

# “CS 374” Fall 2014 — Homework 0

Due Tuesday, September 2, 2014 at noon

---

## ••• Some important course policies •••

---

- **Each student must submit individual solutions for this homework.** You may use any source at your disposal—paper, electronic, or human—but you *must* cite *every* source that you use. See the academic integrity policies on the course web site for more details. For all future homeworks, groups of up to three students will be allowed to submit joint solutions.
- **Submit your solutions on standard printer/copier paper.** At the top of each page, please clearly print your name and NetID, and indicate your registered discussion section. Use both sides of the paper. If you plan to write your solutions by hand, please print the last three pages of this homework as templates. If you plan to typeset your homework, you can find a  $\text{\LaTeX}$  template on the course web site; well-typeset homework will get a small amount of extra credit.
- **Submit your solutions in the drop boxes outside 1404 Siebel.** There is a separate drop box for each numbered problem. Don’t staple your entire homework together. Don’t give your homework to Jeff in class; he is fond of losing important pieces of paper.
- **Avoid the Three Deadly Sins!** There are a few dangerous writing (and thinking) habits that will trigger an automatic zero on any homework or exam problem. Yes, we are completely serious.
  - Give complete solutions, not just examples.
  - Declare all your variables.
  - Never use weak induction.
- Answering any homework or exam problem (or subproblem) in this course with “I don’t know” *and nothing else* is worth 25% partial credit. We will accept synonyms like “No idea” or “WTF”, but you must write *something*.

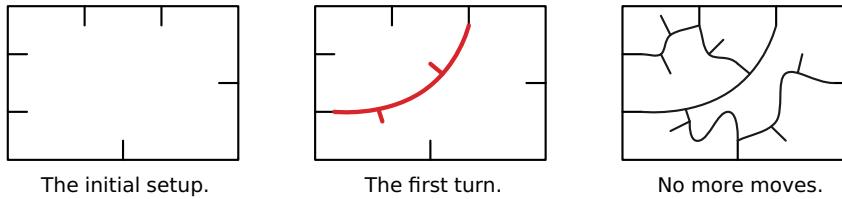
---

**See the course web site for more information.**

If you have any questions about these policies,  
please don’t hesitate to ask in class, in office hours, or on Piazza.

---

1. The Terminal Game is a two-person game played with pen and paper. The game begins by drawing a rectangle with  $n$  “terminals” protruding into the rectangle, for some positive integer  $n$ , as shown in the figure below. On a player’s turn, she selects two terminals, draws a simple curve from one to the other without crossing any other curve (or itself), and finally draws a new terminal on each side of the curve. A player loses if it is her turn and no moves are possible, that is, if no two terminals may be connected without crossing at least one other curve.



Analyze this game, answering the following questions (and any more that you determine the answers to): When is it better to play first, and when it is better to play second? Is there always a winning strategy? What is the smallest number of moves in which you can defeat your opponent? Prove your answers are correct.

2. Herr Professor Doktor Georg von den Dschungel has a 23-node binary tree, in which each node is labeled with a unique letter of the German alphabet, which is just like the English alphabet with four extra letters: Ä, Ö, Ü, and ß. (Don’t confuse these with A, O, U, and B!) Preorder and postorder traversals of the tree visit the nodes in the following order:

- Preorder: B K Ü E H L Z I Ö R C B T S O A Ä D F M N U G
- Postorder: H I Ö Z R L E C Ü S O T A ß K D M U G N F Ä B

- (a) List the nodes in Professor von den Dschungel’s tree in the order visited by an inorder traversal.  
 (b) Draw Professor von den Dschungel’s tree.

3. Recursively define a set  $L$  of strings over the alphabet {0, 1} as follows:

- The empty string  $\epsilon$  is in  $L$ .
- For any two strings  $x$  and  $y$  in  $L$ , the string 0x1y0 is also in  $L$ .
- These are the only strings in  $L$ .

- (a) Prove that the string 000010101010010100 is in  $L$ .  
 (b) Prove by induction that every string in  $L$  has exactly twice as many 0s as 1s.  
 (c) Give an example of a string with exactly twice as many 0s as 1s that is *not* in  $L$ .

Let  $\#(a, w)$  denote the number of times symbol  $a$  appears in string  $w$ ; for example,

$$\#(0, \textcolor{red}{000010101010010100}) = 12 \quad \text{and} \quad \#(1, \textcolor{red}{000010101010010100}) = 6.$$

You may assume without proof that  $\#(a, xy) = \#(a, x) + \#(a, y)$  for any symbol  $a$  and any strings  $x$  and  $y$ .

4. This is an extra credit problem. Submit your solutions in the drop box for problem 2 (but don’t staple your solutions for 2 and 4 together).

A *perfect riffle shuffle*, also known as a *Faro shuffle*, is performed by cutting a deck of cards exactly in half and then *perfectly interleaving* the two halves. There are two different types of perfect shuffles, depending on whether the top card of the resulting deck comes from the top half or the bottom half of the original deck. An *out-shuffle* leaves the top card of the deck unchanged. After an in-shuffle, the original top card becomes the second card from the top. For example:

$$\text{OutShuffle(A\spadesuit 2\spadesuit 3\spadesuit 4\spadesuit 5\heartsuit 6\heartsuit 7\heartsuit 8\heartsuit)} = \text{A\spadesuit 5\heartsuit 2\spadesuit 6\heartsuit 3\spadesuit 7\heartsuit 4\spadesuit 8\heartsuit}$$

$$\text{InShuffle(A\spadesuit 2\spadesuit 3\spadesuit 4\spadesuit 5\heartsuit 6\heartsuit 7\heartsuit 8\heartsuit)} = \text{5\heartsuit A\spadesuit 6\heartsuit 2\spadesuit 7\heartsuit 3\spadesuit 8\heartsuit 4\spadesuit}$$

(If you are unfamiliar with playing cards, please refer to the Wikipedia article [https://en.wikipedia.org/wiki/Standard\\_52-card\\_deck](https://en.wikipedia.org/wiki/Standard_52-card_deck).)

Suppose we start with a deck of  $2^n$  distinct cards, for some non-negative integer  $n$ . What is the effect of performing exactly  $n$  perfect in-shuffles on this deck? Prove your answer is correct!

# “CS 374” Fall 2014 — Homework 1

## Due Tuesday, September 9, 2014 at noon

---

Groups of up to three students may submit common solutions for each problem in this homework and in all future homeworks. You are responsible for forming your own groups; you are welcome to advertise for group members on Piazza. You need not use the same group for every homework, or even for every problem in a single homework. Please clearly print the names and NetIDs of each of your group members at the top of each submitted solution, along with *one* discussion section where we should return your graded work. If you submit hand-written solutions, please use the last three pages of this homework as templates.

---

1. Give regular expressions for each of the following languages over the alphabet  $\{0, 1\}$ . You do not need to prove your answers are correct.
  - (a) All strings with an odd number of 1s.
  - (b) All strings with at most three 0s.
  - (c) All strings that do not contain the substring 010.
  - (d) All strings in which every occurrence of the substring 00 occurs before every occurrence of the substring 11.

2. Recall that the *reversal*  $w^R$  of a string  $w$  is defined recursively as follows:

$$w^R = \begin{cases} \epsilon & \text{if } w = \epsilon \\ x \bullet a & \text{if } w = ax \text{ for some symbol } a \text{ and some string } x \end{cases}$$

The reversal  $L^R$  of a language  $L$  is defined as the set of reversals of all strings in  $L$ :

$$L^R := \{w^R \mid w \in L\}$$

- (a) Prove that  $(L^*)^R = (L^R)^*$  for every language  $L$ .
- (b) Prove that the reversal of any regular language is also a regular language. (You may assume part (a) even if you haven't proved it yet.)

You may assume the following facts without proof:

- $L^* \bullet L^* = L^*$  for every language  $L$ .
- $(w^R)^R = w$  for every string  $w$ .
- $(x \bullet y)^R = y^R \bullet x^R$  for all strings  $x$  and  $y$ .

[Hint: Yes, all three proofs use induction, but induction on what? And yes, all three proofs.]

3. Describe context-free grammars for each of the following languages over the alphabet  $\{0, 1\}$ . Explain *briefly* why your grammars are correct; in particular, describe *in English* the language generated by each non-terminal in your grammars. (We are not looking for full formal proofs of correctness, but convincing evidence that *you* understand why your answers are correct.)
  - (a) The set of all strings with more than twice as many 0s as 1s.
  - (b) The set of all strings that are *not* palindromes.
  - \*(c) [Extra credit] The set of all strings that are *not* equal to  $ww$  for any string  $w$ .

[Hint:  $a + b = b + a$ .]

# “CS 374” Fall 2014 — Homework 2

Due Tuesday, September 16, 2014 at noon

---

Groups of up to three students may submit common solutions for each problem in this homework and in all future homeworks. You are responsible for forming your own groups; you are welcome to advertise for group members on Piazza. You need not use the same group for every homework, or even for every problem in a single homework. Please clearly print the names and NetIDs of each of your group members at the top of each submitted solution, along with **one** discussion section where we should return your graded work. If you submit hand-written solutions, please use the last three pages of this homework as templates.

---

1. **C comments** are the set of strings over alphabet  $\Sigma = \{*, /, A, \diamond, \downarrow\}$  that form a proper comment in the C program language and its descendants, like C++ and Java. Here  $\downarrow$  represents the newline character,  $\diamond$  represents any other whitespace character (like the space and tab characters), and **A** represents any non-whitespace character other than  $*$  or  $/$ .<sup>1</sup> There are two types of C comments:

- Line comments: Strings of the form  $// \dots \downarrow$ .
- Block comments: Strings of the form  $/* \dots */$ .

Following the C99 standard, we explicitly disallow *nesting* comments of the same type. A line comment starts with  $//$  and ends at the first  $\downarrow$  after the opening  $//$ . A block comment starts with  $/*$  and ends at the the first  $*/$  completely after the opening  $/*$ ; in particular, every block comment has at least two  $*$ s. For example, the following strings are all valid C comments:

- $/***/$
- $//\diamond//\diamond\downarrow$
- $/*//\diamond*\diamond\downarrow**/$
- $/*\diamond//\diamond\downarrow\diamond*/$

On the other hand, the following strings are *not* valid C comments:

- $/*/$
- $//\diamond//\diamond\downarrow\diamond\downarrow$
- $/*\diamond/*\diamond*/\diamond*/$

- (a) Describe a DFA that accepts the set of all C comments.
- (b) Describe a DFA that accepts the set of all strings composed entirely of blanks( $\diamond$ ), newlines( $\downarrow$ ), and C comments.

You must explain *in English* how your DFAs work. Drawings or formal descriptions without English explanations will receive no credit, even if they are correct.

2. Construct a DFA for the following language over alphabet {0, 1}:

$$L = \left\{ w \in \{0, 1\}^* \mid \begin{array}{l} \text{the number represented by binary string } w \text{ is divisible} \\ \text{by 19, but the length of } w \text{ is not a multiple of 23} \end{array} \right\}.$$

You must explain in English how your DFA works. A formal description without an English explanation will receive no credit, even if it is correct. Don't even try to draw the DFA.

3. Prove that each of the following languages is *not* regular.

- (a)  $\{w \in \{0\}^* \mid \text{length of } w \text{ is a perfect square; that is, } |w| = k^2 \text{ for some integer } k\}$ .
- (b)  $\{w \in \{0, 1\}^* \mid \text{the number represented by } w \text{ as a binary string is a perfect square}\}$ .

\*4. [Extra credit] Suppose  $L$  is a regular language which guarantees to contain at least one palindrome. Prove that if an  $n$ -state DFA  $M$  accepts  $L$ , then  $L$  contains a palindrome of length polynomial in  $n$ . What is the polynomial bound you get?

---

<sup>1</sup>The actual C commenting syntax is considerably more complex than described here, because of character and string literals.

- The opening /\* or // of a comment must not be inside a string literal ("...") or a (multi-)character literal ('...').
- The opening double-quote of a string literal must not be inside a character literal ('') or a comment.
- The closing double-quote of a string literal must not be escaped (\")
- The opening single-quote of a character literal must not be inside a string literal ("...'"...) or a comment.
- The closing single-quote of a character literal must not be escaped (\')
- A backslash escapes the next symbol if and only if it is not itself escaped (\\\) or inside a comment.

For example, the string /\*\\/\*/\*/\*\\/\*/\* is a valid string literal (representing the 5-character string /\*\"\*/), which is itself a valid block comment!) followed immediately by a valid block comment. For this homework question, just pretend that the characters ', ", and \ don't exist.

The C++ commenting is even more complicated, thanks to the addition of raw string literals. Don't ask.

Some C and C++ compilers do support nested block comments, in violation of the language specification. A few other languages, like OCaml, explicitly allow nesting comments.

# “CS 374” Fall 2014 — Homework 3

Due Tuesday, September 23, 2014 at noon

---

- As usual, groups of up to three students may submit common solutions for this assignment. Each group should submit exactly *one* solution for each problem. Please clearly print the names and NetIDs of each of your group members at the top of each submitted solution, along with *one* discussion section where we should return your graded work. If you submit hand-written solutions, please use the last three pages of this handout as templates.
  - If a question asks you to construct an NFA, you are welcome to use  $\epsilon$ -transitions.
- 

1. For each of the following regular expressions, describe or draw two finite-state machines:

- An NFA that accepts the same language, using Thompson’s algorithm (described in class and in the notes)
  - An equivalent DFA, using the incremental subset construction described in class. For each state in your DFA, identify the corresponding subset of states in your NFA. Your DFA should have no unreachable states.
- (a)  $(\textcolor{red}{0}1 + \textcolor{red}{1}0)^*(\textcolor{red}{0} + \textcolor{red}{1} + \epsilon)$   
(b)  $\textcolor{red}{1}^* + (\textcolor{red}{1}0)^* + (\textcolor{red}{1}00)^*$

2. Prove that for any regular language  $L$ , the following languages are also regular:

- (a)  $\text{SUBSTRINGS}(L) := \{x \mid wxy \in L \text{ for some } w, y \in \Sigma^*\}$   
(b)  $\text{HALF}(L) := \{w \mid ww \in L\}$

[Hint: Describe how to transform a DFA for  $L$  into NFAs for  $\text{SUBSTRINGS}(L)$  and  $\text{HALF}(L)$ . What do your NFAs have to guess? Don’t forget to explain **in English** how your NFAs work.]

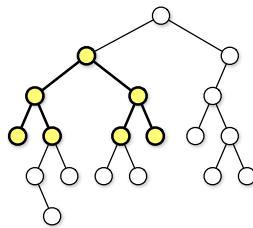
3. Which of the following languages over the alphabet  $\Sigma = \{\textcolor{red}{0}, \textcolor{red}{1}\}$  are regular and which are not? Prove your answers are correct. Recall that  $\Sigma^+$  denotes the set of all *nonempty* strings over  $\Sigma$ .

- (a)  $\{wxw \mid w, x \in \Sigma^+\}$   
(b)  $\{wxx \mid w, x \in \Sigma^+\}$   
(c)  $\{wxwy \mid w, x, y \in \Sigma^+\}$   
(d)  $\{wxxxy \mid w, x, y \in \Sigma^+\}$

# “CS 374” Fall 2014 — Homework 4

Due Tuesday, October 7, 2014 at noon

- 
- For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth. Describe and analyze a recursive algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return both the root and the depth of this subtree.



The largest complete subtree of this binary tree has depth 2.

- Consider the following cruel and unusual sorting algorithm.

<u>CRUEL(<math>A[1..n]</math>):</u> if $n > 1$ CRUEL( $A[1..n/2]$ ) CRUEL( $A[n/2 + 1..n]$ ) UNUSUAL( $A[1..n]$ )
<u>UNUSUAL(<math>A[1..n]</math>):</u> if $n = 2$ if $A[1] > A[2]$ ⟨⟨the only comparison!⟩⟩ swap $A[1] \leftrightarrow A[2]$ else for $i \leftarrow 1$ to $n/4$ ⟨⟨swap 2nd and 3rd quarters⟩⟩ swap $A[i + n/4] \leftrightarrow A[i + n/2]$ UNUSUAL( $A[1..n/2]$ )                                ⟨⟨recurse on left half⟩⟩ UNUSUAL( $A[n/2 + 1..n]$ )                                ⟨⟨recurse on right half⟩⟩ UNUSUAL( $A[n/4 + 1..3n/4]$ )                                ⟨⟨recurse on middle half⟩⟩

Notice that the comparisons performed by the algorithm do not depend at all on the values in the input array; such a sorting algorithm is called *oblivious*. Assume for this problem that the input size  $n$  is always a power of 2.

- Prove by induction that CRUEL correctly sorts any input array. [Hint: Consider an array that contains  $n/4$  1s,  $n/4$  2s,  $n/4$  3s, and  $n/4$  4s. Why is this special case enough?]
- Prove that CRUEL would *not* correctly sort if we removed the for-loop from UNUSUAL.
- Prove that CRUEL would *not* correctly sort if we swapped the last two lines of UNUSUAL.
- What is the running time of UNUSUAL? Justify your answer.
- What is the running time of CRUEL? Justify your answer.

3. In the early 20th century, a German mathematician developed a variant of the Towers of Hanoi game, which quickly became known in the American literature as “Liberty Towers”.<sup>1</sup> In this variant, there is a row of  $k \geq 3$  pegs, numbered in order from 1 to  $k$ . In a single turn, for any index  $i$ , you can move the smallest disk on peg  $i$  to either peg  $i - 1$  or peg  $i + 1$ , subject to the usual restriction that you cannot place a bigger disk on a smaller disk. Your mission is to move a stack of  $n$  disks from peg 1 to peg  $k$ .
- (a) Describe and analyze a recursive algorithm for the case  $k = 3$ . *Exactly* how many moves does your algorithm perform?
  - (b) Describe and analyze a recursive algorithm for the case  $k = n + 1$  that requires at most  $O(n^3)$  moves. To simplify the algorithm, assume that  $n$  is a power of 2. [Hint: Use part (a).]
  - (c) **[Extra credit]** Describe and analyze a recursive algorithm for the case  $k = n + 1$  that requires at most  $O(n^2)$  moves. Do *not* assume that  $n$  is a power of 2. [Hint: Don’t use part (a).]
  - (d) **[Extra credit]** Describe and analyze a recursive algorithm for the case  $k = \sqrt{n} + 1$  that requires at most a polynomial number of moves. To simplify the algorithm, assume that  $n$  is a power of 4. What polynomial bound do you get? [Hint: Use part (a)!]
  - \*(e) **[Extra extra credit]** Describe and analyze a recursive algorithm for arbitrary  $n$  and  $k$ . How small must  $k$  be (as a function of  $n$ ) so that the number of moves is bounded by a polynomial in  $n$ ? (This is actually an open research problem, a phrase which here means “Nobody knows the best answer.”)

---

<sup>1</sup>No, not really. During World War I, many German-derived or Germany-related names were changed to more patriotic variants. For example, sauerkraut became “liberty cabbage”, hamburgers became “liberty sandwiches”, frankfurters became “Liberty sausages” or “hot dogs”, German measles became “liberty measles”, dachsunds became “liberty pups”, German shepherds became “Alsatians”, and pinochle (the card game) became “Liberty”. For more recent anti-French examples, see “freedom fries”, “freedom toast”, and “liberty lip lock”. Americans are weird.

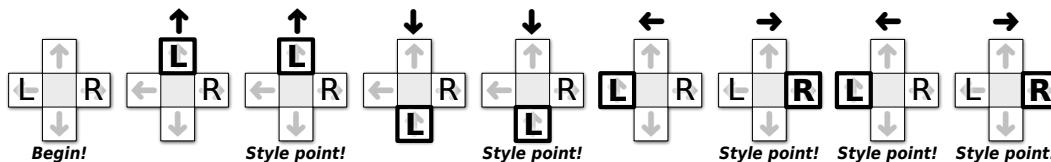
# “CS 374” Fall 2014 — Homework 5

Due Tuesday, October 14, 2014 at noon

1. **Dance Dance Revolution** is a dance video game, first introduced in Japan by Konami in 1998. Players stand on a platform marked with four arrows, pointing forward, back, left, and right, arranged in a cross pattern. During play, the game plays a song and scrolls a sequence of  $n$  arrows ( $\leftarrow$ ,  $\uparrow$ ,  $\downarrow$ , or  $\rightarrow$ ) from the bottom to the top of the screen. At the precise moment each arrow reaches the top of the screen, the player must step on the corresponding arrow on the dance platform. (The arrows are timed so that you’ll step with the beat of the song.)

You are playing a variant of this game called “Vogue Vogue Revolution”, where the goal is to play perfectly but move as little as possible. When an arrow reaches the top of the screen, if one of your feet is already on the correct arrow, you are awarded one style point for maintaining your current pose. If neither foot is on the right arrow, you must move one (and *only* one) of your feet from its current location to the correct arrow on the platform. If you ever step on the wrong arrow, or fail to step on the correct arrow, or move more than one foot at a time, or move either foot when you are already standing on the correct arrow, or insult Beyoncé, all your style points are immediately taken away and you lose.

How should you move your feet to maximize your total number of style points? For purposes of this problem, assume you always start with your left foot on  $\leftarrow$  and your right foot on  $\rightarrow$ , and that you’ve memorized the entire sequence of arrows. For example, if the sequence is  $\uparrow\uparrow\downarrow\downarrow\leftarrow\rightarrow\leftarrow\rightarrow$ , you can earn 5 style points by moving your feet as shown below:



Describe and analyze an efficient algorithm to find the maximum number of style points you can earn during a given VVR routine. Your input is an array  $Arrow[1..n]$  containing the sequence of arrows.

2. Recall that a *palindrome* is any string that is exactly the same as its reversal, like **I**, or **DEED**, or **RACECAR**, or **AMANAPLANACATACANALPANAMA**.

Any string can be decomposed into a sequence of palindrome substrings. For example, the string **BUBBASEESABANANA** (“Bubba sees a banana.”) can be broken into palindromes in the following ways (among many others):

**BUB • BASEESAB • ANANA**  
**B • U • BB • A • SEES • ABA • NAN • A**  
**B • U • BB • A • SEES • A • B • ANANA**  
**B • U • B • B • A • S • E • E • S • A • B • ANA • N • A**

Describe and analyze an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string **BUBBASEESABANANA**, your algorithm would return the integer 3.

3. Suppose you are given a DFA  $M = (\{0, 1\}, Q, s, A, \delta)$  and a binary string  $w \in \{0, 1\}^*$ .
- (a) Describe and analyze an algorithm that computes the longest subsequence of  $w$  that is accepted by  $M$ , or correctly reports that  $M$  does not accept any subsequence of  $w$ .
  - \*(b) [Extra credit] Describe and analyze an algorithm that computes the *shortest supersequence* of  $w$  that is accepted by  $M$ , or correctly reports that  $M$  does not accept any supersequence of  $w$ . (Recall that a string  $x$  is a supersequence of  $w$  if and only if  $w$  is a subsequence of  $x$ .)

Analyze both of your algorithms in terms of the parameters  $n = |w|$  and  $k = |Q|$ .

**Rubric (for all dynamic programming problems):** As usual, a score of  $x$  on the following 10-point scale corresponds to a score of  $\lceil x/3 \rceil$  on the 4-point homework scale.

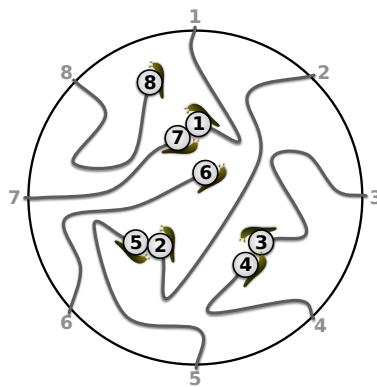
- 6 points for a correct recurrence, described either using mathematical notation or as pseudocode for a recursive algorithm.
  - + 1 point for a clear **English** description of the function you are trying to evaluate. (Otherwise, we don't even know what you're *trying* to do.) **Automatic zero if the English description is missing.**
  - + 1 point for stating how to call your function to get the final answer.
  - + 1 point for base case(s).  $-1/2$  for one *minor* bug, like a typo or an off-by-one error.
  - + 3 points for recursive case(s).  $-1$  for each *minor* bug, like a typo or an off-by-one error. **No credit for the rest of the problem if the recursive case(s) are incorrect.**
- 4 points for details of the dynamic programming algorithm
  - + 1 point for describing the memoization data structure
  - + 2 points for describing a correct evaluation order; a clear picture is sufficient. If you use nested loops, be sure to specify the nesting order.
  - + 1 point for time analysis
- It is *not* necessary to state a space bound.
- For problems that ask for an algorithm that computes an optimal *structure*—such as a subset, partition, subsequence, or tree—an algorithm that computes only the *value* or *cost* of the optimal structure is sufficient for full credit.
- Official solutions usually include pseudocode for the final iterative dynamic programming algorithm, **but this is not required for full credit**. If your solution includes iterative pseudocode, you do not need to separately describe the recurrence, data structure, or evaluation order. (But you still need to describe the underlying recursive function in English.)
- Official solutions will provide target time bounds. Algorithms that are faster than this target are worth more points; slower algorithms are worth fewer points, typically by 2 or 3 points for each factor of  $n$ . Partial credit is scaled to the new maximum score, and all points above 10 are recorded as extra credit.

We rarely include these target time bounds in the actual questions, because when we do include them, significantly more students turn in algorithms that meet the target time bound but don't work (earning 0/10) instead of correct algorithms that are slower than the target time bound (earning 8/10).

# “CS 374” Fall 2014 — Homework 6

Due Tuesday, October 21, 2014 at noon

1. Every year, as part of its annual meeting, the Antarctic Snail Lovers of Upper Glacierville hold a Round Table Mating Race. Several high-quality breeding snails are placed at the edge of a round table. The snails are numbered in order around the table from 1 to  $n$ . During the race, each snail wanders around the table, leaving a trail of slime behind it. The snails have been specially trained never to fall off the edge of the table or to cross a slime trail, even their own. If two snails meet, they are declared a breeding pair, removed from the table, and whisked away to a romantic hole in the ground to make little baby snails. Note that some snails may never find a mate, even if the race goes on forever.



The end of a typical Antarctic SLUG race. Snails 6 and 8 never find mates.

The organizers must pay  $M[3,4] + M[2,5] + M[1,7]$ .

For every pair of snails, the Antarctic SLUG race organizers have posted a monetary reward, to be paid to the owners if that pair of snails meets during the Mating Race. Specifically, there is a two-dimensional array  $M[1..n, 1..n]$  posted on the wall behind the Round Table, where  $M[i, j] = M[j, i]$  is the reward to be paid if snails  $i$  and  $j$  meet. Rewards may be positive, negative, or zero.

Describe and analyze an algorithm to compute the maximum total reward that the organizers could be forced to pay, given the array  $M$  as input.

2. Consider a weighted version of the class scheduling problem, where different classes offer different number of credit hours, which are of course totally unrelated to the duration of the class lectures. Given arrays  $S[1..n]$  of start times, an array  $F[1..n]$  of finishing times, and an array  $H[1..n]$  of credit hours as input, your goal is to choose a set of non-overlapping classes with the largest possible number of credit hours.

  - (a) Prove that the greedy algorithm described in class — Choose the class that ends first and recurse — does *not* always return the best schedule.
  - (b) Describe an efficient algorithm to compute the best schedule.

In addition to submitting a solution on paper as usual, please *individually* submit an electronic solution for this problem on CrowdGrader. Please see the course web page for detailed instructions.

3. Suppose you have just purchased a new type of hybrid car that uses fuel extremely efficiently, but can only travel 100 miles on a single battery. The car’s fuel is stored in a single-use battery, which must be replaced after at most 100 miles. The actual fuel is virtually free, but the batteries are expensive and can only be installed by licensed battery-replacement technicians. Thus, even if you decide to replace your battery early, you must still pay full price for the new battery to be installed. Moreover, because these batteries are in high demand, no one can afford to own more than one battery at a time.

Suppose you are trying to get from San Francisco to New York City on the new Inter-Continental Super-Highway, which runs in a direct line between these two cities. There are several fueling stations along the way; each station charges a different price for installing a new battery. Before you start your trip, you carefully print the Wikipedia page listing the locations and prices of every fueling station on the ICSH. Given this information, how do you decide the best places to stop for fuel?

More formally, suppose you are given two arrays  $D[1..n]$  and  $C[1..n]$ , where  $D[i]$  is the distance from the start of the highway to the  $i$ th station, and  $C[i]$  is the cost to replace your battery at the  $i$ th station. Assume that your trip starts and ends at fueling stations (so  $D[1] = 0$  and  $D[n]$  is the total length of your trip), and that your car starts with an empty battery (so you must install a new battery at station 1).

- (a) Describe and analyze a greedy algorithm to find the minimum number of refueling stops needed to complete your trip. Don’t forget to prove that your algorithm is correct.
- (b) But what you really want to minimize is the total *cost* of travel. Show that your greedy algorithm in part (a) does *not* produce an optimal solution when extended to this setting.
- (c) Describe an efficient algorithm to compute the locations of the fuel stations you should stop at to minimize the total cost of travel.



## “CS 374” Fall 2014 — Homework 8

Due Tuesday, November 4, 2014 at noon

---

1. After a grueling algorithms midterm, you decide to take the bus home. Since you planned ahead, you have a schedule that lists the times and locations of every stop of every bus in Champaign-Urbana. Champaign-Urbana is currently suffering from a plague of zombies, so even though the bus stops have fences that *supposedly* keep the zombies out, you’d still like to spend as little time waiting at bus stops as possible. Unfortunately, there isn’t a single bus that visits both your exam building and your home; you must transfer between buses at least once.

Describe and analyze an algorithm to determine a sequence of bus rides from Siebel to your home, that minimizes the total time you spend waiting at bus stops. You can assume that there are  $b$  different bus lines, and each bus stops  $n$  times per day. Assume that the buses run exactly on schedule, that you have an accurate watch, and that walking between bus stops is too dangerous to even contemplate.

2. It is well known that the global economic collapse of 2017 was caused by computer scientists indiscriminately abusing weaknesses in the currency exchange market. *Arbitrage* was a money-making scheme that takes advantage of inconsistencies in currency exchange rates. Suppose a currency trader with \$1,000,000 discovered that 1 US dollar could be traded for 120 Japanese yen, 1 yen could be traded for 0.01 euros, and 1 euro could be traded for 1.2 US dollars. Then by converting his money from dollars to yen, then from yen to euros, and finally from euros back to dollars, the trader could instantly turn his \$1,000,000 into \$1,440,000! The cycle of currencies  $\$ \rightarrow ¥ \rightarrow € \rightarrow \$$  was called an *arbitrage cycle*. Finding and exploiting arbitrage cycles before the prices were corrected required extremely fast algorithms. Of course, now that the entire world uses plastic bags as currency, such abuse is impossible.

Suppose  $n$  different currencies are traded in the global currency market. You are given a two-dimensional array  $Exch[1..n, 1..n]$  of exchange rates between every pair of currencies; for all indices  $i$  and  $j$ , one unit of currency  $i$  buys  $Exch[i, j]$  units of currency  $j$ . (Do *not* assume that  $Exch[i, j] \cdot Exch[j, i] = 1$ .)

- (a) Describe an algorithm that computes an array  $Most[1..n]$ , where  $Most[i]$  is the largest amount of currency  $i$  that you can obtain by trading, starting with one unit of currency 1, assuming there are no arbitrage cycles.  
(b) Describe an algorithm to determine whether the given array of currency exchange rates creates an arbitrage cycle.
3. Describe and analyze an algorithm to find the *second smallest spanning tree* of a given undirected graph  $G$  with weighted edges, that is, the spanning tree of  $G$  with smallest total weight except for the minimum spanning tree. Because the minimum spanning tree is haunted, or something.

# “CS 374” Fall 2014 — Homework 9

Due Tuesday, November 18, 2014 at noon

---

The following questions ask you to describe various Turing machines. In each problem, give *both* a formal description of your Turing machine in terms of specific states, tape symbols, and transition functions *and* explain in English how your Turing machine works. In particular:

- Clearly specify what variant of Turing machine you are using: Number of tapes, number of heads, allowed head motions, halting conditions, and so on.
  - Include the type signature of your machine’s transition function. The standard model uses a transition function whose signature is  $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{-1, +1\}$ .
  - If necessary, break your Turing machine into smaller functional pieces, and describe those pieces separately (both formally and in English).
  - Use state names that convey their meaning/purpose.
- 

1. Describe a Turing machine that computes the function  $\lceil \log_2 n \rceil$ . Given the string  $1^n$  as input, for any positive integer  $n$ , your machine should return the string  $1^{\lceil \log_2 n \rceil}$  as output. For example, given the input string  $1111111111111$  (thirteen 1s), your machine should output the string  $1111$ , because  $2^3 < 13 \leq 2^4$ .
2. A *binary-tree* Turing machine uses an infinite binary tree as its tape; that is, *every* cell in the tape has a left child and a right child. At each step, the head moves from its current cell to its *Parent*, its *Left* child, or to its *Right* child. Thus, the transition function of such a machine has the form  $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{P, L, R\}$ . The input string is initially given along the left spine of the tape.

Prove that any binary-tree Turing machine can be simulated by a standard Turing machine. That is, given any binary-tree Turing machine  $M = (\Gamma, \square, \Sigma, Q, \text{start}, \text{accept}, \text{reject}, \delta)$ , describe a standard Turing machine  $M' = (\Gamma', \square', \Sigma, Q', \text{start}', \text{accept}', \text{reject}', \delta')$  that accepts and rejects exactly the same input strings as  $M$ . Be sure to describe how a single transition of  $M$  is simulated by  $M'$ .

**In addition to submitting paper solutions, please also electronically submit your solution to this problem on CrowdGrader.**

## 3. [Extra credit]

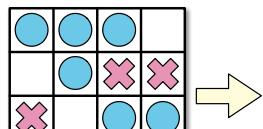
A *tag*-Turing machine has two heads: one can only read, the other can only write. Initially, the read head is located at the left end of the tape, and the write head is located at the first blank after the input string. At each transition, the read head can either move one cell to the right or stay put, but the write head *must* write a symbol to its current cell and move one cell to the right. Neither head can ever move to the left.

Prove that any standard Turing machine can be simulated by a tag-Turing machine. That is, given any standard Turing machine  $M$ , formally describe a tag-Turing machine  $M'$  that accepts and rejects exactly the same strings as  $M$ . Be sure to describe how a single transition of  $M$  is simulated by  $M'$ .

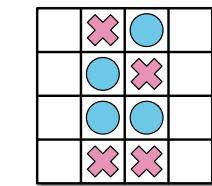
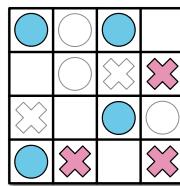
# “CS 374” Fall 2014 — Homework 10

Due Tuesday, December 2, 2014 at noon

1. Consider the following problem, called BoxDEPTH: Given a set of  $n$  axis-aligned rectangles in the plane, how big is the largest subset of these rectangles that contain a common point?
  - (a) Describe a polynomial-time reduction from BoxDEPTH to MAXCLIQUE.
  - (b) Describe and analyze a polynomial-time algorithm for BoxDEPTH. [Hint: Don’t try to optimize the running time;  $O(n^3)$  is good enough.]
  - (c) Why don’t these two results imply that P=NP?
2. Consider the following solitaire game. The puzzle consists of an  $n \times m$  grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) every row contains at least one stone, and (2) no column contains stones of both colors. For some initial configurations of stones, reaching this goal is impossible.



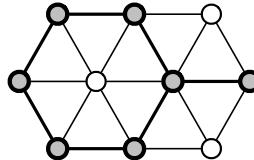
A solvable puzzle and one of its many solutions.



An unsolvable puzzle.

Prove that it is NP-hard to determine, given an initial configuration of red and blue stones, whether this puzzle can be solved.

3. A subset  $S$  of vertices in an undirected graph  $G$  is called **triangle-free** if, for every triple of vertices  $u, v, w \in S$ , at least one of the three edges  $uv, uw, vw$  is *absent* from  $G$ . Prove that finding the size of the largest triangle-free subset of vertices in a given undirected graph is NP-hard.



A triangle-free subset of 7 vertices.

This is **not** the largest triangle-free subset in this graph.

In addition to submitting paper solutions, please also electronically submit your solution to this problem on CrowdGrader.

4. [Extra credit] Describe a direct polynomial-time reduction from 4COLOR to 3COLOR. (This is significantly harder than the opposite direction, which you’ll see in lab on Wednesday. Don’t go through the Cook-Levin Theorem.)

# “CS 374” Fall 2014 ✦ Homework 11

Due Tuesday, December 9, 2014 at noon

---

- Recall that  $w^R$  denotes the reversal of string  $w$ ; for example,  $\text{TURING}^R = \text{GNIRUT}$ . Prove that the following language is undecidable.

$$\text{REVACCEPT} := \{\langle M \rangle \mid M \text{ accepts } \langle M \rangle^R\}$$

- Let  $M$  be a Turing machine, let  $w$  be an arbitrary input string, and let  $s$  be an integer. We say that  $M$  accepts  $w$  in space  $s$  if, given  $w$  as input,  $M$  accesses only the first  $s$  cells on the tape and eventually accepts.

- Prove that the following language is decidable:

$$\{\langle M, w \rangle \mid M \text{ accepts } w \text{ in space } |w|^2\}$$

- Prove that the following language is undecidable:

$$\{\langle M \rangle \mid M \text{ accepts at least one string } w \text{ in space } |w|^2\}$$

- [Extra credit] For each of the following languages, either prove that the language is decidable, or prove that the language is undecidable.

- $L_0 = \{\langle M \rangle \mid \text{given any input string, } M \text{ eventually leaves its start state}\}$
- $L_1 = \{\langle M \rangle \mid M \text{ decides } L_0\}$
- $L_2 = \{\langle M \rangle \mid M \text{ decides } L_1\}$
- $L_3 = \{\langle M \rangle \mid M \text{ decides } L_2\}$
- $L_4 = \{\langle M \rangle \mid M \text{ decides } L_3\}$

1. Prove that every non-negative integer can be represented as the sum of distinct powers of 2. (“Write it in binary” is not a proof; it’s just a restatement of what you have to prove.)
2. Suppose you and your 8-year-old cousin Elmo decide to play a game with a rectangular bar of chocolate, which has been scored into an  $n \times m$  grid of squares. You and Elmo alternate turns. On each turn, you or Elmo choose one of the available pieces of chocolate and break it along one of the grid lines into two smaller rectangles. Thus, at all times, each piece of chocolate is an  $a \times b$  rectangle for some positive integers  $a$  and  $b$ ; in particular, a  $1 \times 1$  piece cannot be broken into smaller pieces. The game ends when all the pieces are individual squares. The winner is the player who breaks the last piece.

Describe a strategy for winning this game. When should you take the first move, and when should you offer it to Elmo? On each turn, how do you decide which piece to break and where? Prove your answers are correct. [*Hint: Let’s play a  $3 \times 3$  game. You go first. Oh, and I’m kinda busy right now, so could you just play for me whenever it’s my turn? Thanks.*]

3. [To think about later] Now consider a variant of the previous chocolate-bar game, where on each turn you can *either* break a piece into two smaller pieces *or eat a  $1 \times 1$  piece*. This game ends when all the chocolate is gone. The winner is the player who eats the last bite of chocolate (*not* the player who eats the *most* chocolate). Describe a strategy for winning this game, and prove that your strategy works.

These lab problems ask you to prove some simple claims about recursively-defined string functions and concatenation. In each case, we want a self-contained proof by induction that relies on the formal recursive definitions, *not* on intuition. In particular, your proofs must refer to the formal recursive definition of string concatenation:

$$w \bullet z := \begin{cases} z & \text{if } w = \varepsilon \\ a \cdot (x \bullet z) & \text{if } w = ax \text{ for some symbol } a \text{ and some string } x \end{cases}$$

You may also use any of the following facts, which we proved in class:

**Lemma 1:** Concatenating nothing does nothing: For every string  $w$ , we have  $w \bullet \varepsilon = w$ .

**Lemma 2:** Concatenation adds length:  $|w \bullet x| = |w| + |x|$  for all strings  $w$  and  $x$ .

**Lemma 3:** Concatenation is associative:  $(w \bullet x) \bullet y = w \bullet (x \bullet y)$  for all strings  $w$ ,  $x$ , and  $y$ .

1. Let  $\#(a, w)$  denote the number of times symbol  $a$  appears in string  $w$ ; for example,

$$\#(\mathbf{0}, \mathbf{000010101010010100}) = 12 \quad \text{and} \quad \#(\mathbf{1}, \mathbf{000010101010010100}) = 6.$$

- (a) Give a formal recursive definition of  $\#(a, w)$ .
- (b) Prove by induction that  $\#(a, w \bullet z) = \#(a, w) + \#(a, z)$  for any symbol  $a$  and any strings  $w$  and  $z$ .

2. The *reversal*  $w^R$  of a string  $w$  is defined recursively as follows:

$$w^R := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ x^R \bullet a & \text{if } w = a \cdot x \end{cases}$$

- (a) Prove that  $(w \bullet x)^R = x^R \bullet w^R$  for all strings  $w$  and  $x$ .
- (b) Prove that  $(w^R)^R = w$  for every string  $w$ .

---

Give regular expressions that describe each of the following languages over the alphabet  $\{0, 1\}$ . We won’t get to all of these in section.

---

1. All strings containing at least three **0s**.
2. All strings containing at least two **0s** and at least one **1**.
3. All strings containing the substring **000**.
4. All strings *not* containing the substring **000**.
5. All strings in which every run of **0s** has length at least 3.
6. All strings such that every substring **000** appears after every **1**.
7. Every string except **000**. [*Hint: Don’t try to be clever.*]
8. All strings  $w$  such that *in every prefix of  $w$* , the number of **0s** and **1s** differ by at most 1.
- \*9. All strings  $w$  such that *in every prefix of  $w$* , the number of **0s** and **1s** differ by at most 2.
- ★10. All strings in which the substring **000** appears an even number of times.  
(For example, **0001000** and **0000** are in this language, but **00000** is not.)

Jeff showed the context-free grammars in class on Tuesday; in each example, the grammar itself is on the left; the explanation for each non-terminal is on the right.

- Properly nested strings of parentheses.

$$S \rightarrow \epsilon \mid S(S) \quad \text{properly nested parentheses}$$

Here is a different grammar for the same language:

$$S \rightarrow \epsilon \mid (S) \mid SS \quad \text{properly nested parentheses}$$

- $\{\mathbf{0}^m \mathbf{1}^n \mid m \neq n\}$ . This is the set of all binary strings composed of some number of **0**s followed by a different number of **1**s.

$S \rightarrow A \mid B$	all strings $\mathbf{0}^m \mathbf{1}^n$ where $m \neq n$
$A \rightarrow \mathbf{0}A \mid \mathbf{0}C$	all strings $\mathbf{0}^m \mathbf{1}^n$ where $m > n$
$B \rightarrow B\mathbf{1} \mid C\mathbf{1}$	all strings $\mathbf{0}^m \mathbf{1}^n$ where $m < n$
$C \rightarrow \epsilon \mid \mathbf{0}C\mathbf{1}$	all strings $\mathbf{0}^n \mathbf{1}^n$ for some integer $n$

Give context-free grammars for each of the following languages. For each grammar, describe *in English* the language for each non-terminal, and in the examples above. As usual, we won’t get to all of these in section.

1. Binary palindromes: Strings over  $\{\mathbf{0}, \mathbf{1}\}$  that are equal to their reversals. For example: **00111100** and **0100010**, but not **01100**.
2.  $\{\mathbf{0}^{2n} \mathbf{1}^n \mid n \geq 0\}$
3.  $\{\mathbf{0}^m \mathbf{1}^n \mid m \neq 2n\}$
4.  $\{\mathbf{0}, \mathbf{1}\}^* \setminus \{\mathbf{0}^{2n} \mathbf{1}^n \mid n \geq 0\}$
5. Strings of properly nested parentheses **( )**, brackets **[ ]**, and braces **{ }**. For example, the string **([]){}** is in this language, but the string **([)]** is not, because the left and right delimiters don’t match.
6. Strings over  $\{\mathbf{0}, \mathbf{1}\}$  where the number of **0**s is equal to the number of **1**s.
7. Strings over  $\{\mathbf{0}, \mathbf{1}\}$  where the number of **0**s is *not* equal to the number of **1**s.

Construct DFA that accept each of the following languages over the alphabet  $\{0, 1\}$ . We won’t get to all of these in section.

1. (a)  $(0 + 1)^*$   
(b)  $\emptyset$   
(c)  $\{\epsilon\}$
2. Every string except **000**.
3. All strings containing the substring **000**.
4. All strings *not* containing the substring **000**.
5. All strings in which the reverse of the string is the binary representation of a integer divisible by 3.
6. All strings  $w$  such that *in every prefix of  $w$* , the number of **0s** and **1s** differ by at most 2.

Prove that each of the following languages is not regular.

1. Binary palindromes: Strings over  $\{0, 1\}$  that are equal to their reversals. For example: **00111100** and **0100010**, but not **01100**. [Hint: We did this in class.]
2.  $\{0^{2n}1^n \mid n \geq 0\}$
3.  $\{0^m1^n \mid m \neq 2n\}$
4. Strings over  $\{0, 1\}$  where the number of **0**s is exactly twice the number of **1**s.
5. Strings of properly nested parentheses **( )**, brackets **[ ]**, and braces **{ }**. For example, the string **([ ]){}{}** is in this language, but the string **([ )]** is not, because the left and right delimiters don't match.
6.  $\{0^{2^n} \mid n \geq 0\}$  — Strings of **0**s whose length is a power of 2.
7. Strings of the form  $w_1\#w_2\#\cdots\#w_n$  for some  $n \geq 2$ , where each substring  $w_i$  is a string in  $\{0, 1\}^*$ , and some pair of substrings  $w_i$  and  $w_j$  are equal.

For each of the following languages over the alphabet  $\Sigma = \{\texttt{0}, \texttt{1}\}$ , either prove the language is regular (by giving an equivalent regular expression, DFA, or NFA) or prove that the language is not regular (using a fooling set argument). Exactly half of these languages are regular.

1.  $\{\texttt{0}^n\texttt{1}\texttt{0}^n \mid n \geq 0\}$
2.  $\{\texttt{0}^n\texttt{1}\texttt{0}^n w \mid n \geq 0 \text{ and } w \in \Sigma^*\}$
3.  $\{w\texttt{0}^n\texttt{1}\texttt{0}^n x \mid w \in \Sigma^* \text{ and } n \geq 0 \text{ and } x \in \Sigma^*\}$
4. Strings in which the number of **0**s and the number of **1**s differ by at most 2.
5. Strings such that *in every prefix*, the number of **0**s and the number of **1**s differ by at most 2.
6. Strings such that *in every substring*, the number of **0**s and the number of **1**s differ by at most 2.

Here are several problems that are easy to solve in  $O(n)$  time, essentially by brute force. Your task is to design algorithms for these problems that are significantly faster.

1. (a) Suppose  $A[1..n]$  is an array of  $n$  distinct integers, sorted so that  $A[1] < A[2] < \dots < A[n]$ . Each integer  $A[i]$  could be positive, negative, or zero. Describe a fast algorithm that either computes an index  $i$  such that  $A[i] = i$  or correctly reports that no such index exists..
- (b) Now suppose  $A[1..n]$  is a sorted array of  $n$  distinct ***positive*** integers. Describe an even faster algorithm that either computes an index  $i$  such that  $A[i] = i$  or correctly reports that no such index exists. [*Hint: This is really easy.*]
2. Suppose we are given an array  $A[1..n]$  such that  $A[1] \geq A[2]$  and  $A[n-1] \leq A[n]$ . We say that an element  $A[x]$  is a ***local minimum*** if both  $A[x-1] \geq A[x]$  and  $A[x] \leq A[x+1]$ . For example, there are exactly six local minima in the following array:

9	7	7	2	1	3	7	5	4	7	3	3	4	8	6	9
▲	▲			▲			▲	▲	▲	▲	▲			▲	

Describe and analyze a fast algorithm that returns the index of one local minimum. For example, given the array above, your algorithm could return the integer 5, because  $A[5]$  is a local minimum. [*Hint: With the given boundary conditions, any array must contain at least one local minimum. Why?*]

3. (a) Suppose you are given two sorted arrays  $A[1..n]$  and  $B[1..n]$  containing distinct integers. Describe a fast algorithm to find the median (meaning the  $n$ th smallest element) of the union  $A \cup B$ . For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \quad B[1..8] = [2, 4, 5, 8, 17, 19, 21, 23]$$

your algorithm should return the integer 9. [*Hint: What can you learn by comparing one element of A with one element of B?*]

- (b) **To think about on your own:** Now suppose you are given two sorted arrays  $A[1..m]$  and  $B[1..n]$  and an integer  $k$ . Describe a fast algorithm to find the  $k$ th smallest element in the union  $A \cup B$ . For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \quad B[1..5] = [2, 5, 7, 17, 19] \quad k = 6$$

your algorithm should return the integer 7.

Here are several problems that are easy to solve in  $O(n)$  time, essentially by brute force. Your task is to design algorithms for these problems that are significantly faster, and *prove* that your algorithm is correct.

1. (a) Suppose  $A[1..n]$  is an array of  $n$  distinct integers, sorted so that  $A[1] < A[2] < \dots < A[n]$ . Each integer  $A[i]$  could be positive, negative, or zero. Describe a fast algorithm that either computes an index  $i$  such that  $A[i] = i$  or correctly reports that no such index exists..
- (b) Now suppose  $A[1..n]$  is a sorted array of  $n$  distinct *positive* integers. Describe an even faster algorithm that either computes an index  $i$  such that  $A[i] = i$  or correctly reports that no such index exists. [*Hint: This is really easy.*]
2. Suppose we are given an array  $A[1..n]$  such that  $A[1] \geq A[2]$  and  $A[n-1] \leq A[n]$ . We say that an element  $A[x]$  is a *local minimum* if both  $A[x-1] \geq A[x]$  and  $A[x] \leq A[x+1]$ . For example, there are exactly six local minima in the following array:

9	7	7	2	1	3	7	5	4	7	3	3	4	8	6	9
▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲

Describe and analyze a fast algorithm that returns the index of one local minimum. For example, given the array above, your algorithm could return the integer 5, because  $A[5]$  is a local minimum. [*Hint: With the given boundary conditions, any array must contain at least one local minimum. Why?*]

3. (a) Suppose you are given two sorted arrays  $A[1..n]$  and  $B[1..n]$  containing distinct integers. Describe a fast algorithm to find the median (meaning the  $n$ th smallest element) of the union  $A \cup B$ . For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \quad B[1..8] = [2, 4, 5, 8, 17, 19, 21, 23]$$

your algorithm should return the integer 9. [*Hint: What can you learn by comparing one element of A with one element of B?*]

- (b) **To think about on your own:** Now suppose you are given two sorted arrays  $A[1..m]$  and  $B[1..n]$  and an integer  $k$ . Describe a fast algorithm to find the  $k$ th smallest element in the union  $A \cup B$ . For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \quad B[1..5] = [2, 5, 7, 17, 19] \quad k = 6$$

your algorithm should return the integer 7.

A *subsequence* of a sequence (for example, an array, linked list, or string), obtained by removing zero or more elements and keeping the rest in the same sequence order. A subsequence is called a *substring* if its elements are contiguous in the original sequence. For example:

- **SUBSEQUENCE**, **UBSEQU**, and the empty string  $\epsilon$  are all substrings of the string **SUBSEQUENCE**;
  - **SBSQNC**, **UEQUE**, and **EEE** are all subsequences of **SUBSEQUENCE** but not substrings;
  - **QUEUE**, **SSS**, and **FOOBAR** are not subsequences of **SUBSEQUENCE**.
- 

Describe recursive backtracking algorithms for the following problems. *Don’t worry about running times.*

- Given an array  $A[1..n]$  of integers, compute the length of a *longest increasing subsequence*. A sequence  $B[1..\ell]$  is *increasing* if  $B[i] > B[i-1]$  for every index  $i \geq 2$ . For example, given the array

$$\langle 3, \underline{1}, \underline{4}, 1, \underline{5}, 9, 2, \underline{6}, 5, 3, 5, \underline{8}, \underline{9}, 7, 9, 3, 2, 3, 8, 4, 6, 2, 7 \rangle$$

your algorithm should return the integer 6, because  $\langle 1, 4, 5, 6, 8, 9 \rangle$  is a longest increasing subsequence (one of many).

- Given an array  $A[1..n]$  of integers, compute the length of a *longest decreasing subsequence*. A sequence  $B[1..\ell]$  is *decreasing* if  $B[i] < B[i-1]$  for every index  $i \geq 2$ . For example, given the array

$$\langle 3, 1, 4, 1, 5, \underline{9}, 2, \underline{6}, 5, 3, \underline{5}, 8, 9, 7, 9, 3, 2, 3, 8, \underline{4}, 6, \underline{2}, 7 \rangle$$

your algorithm should return the integer 5, because  $\langle 9, 6, 5, 4, 2 \rangle$  is a longest decreasing subsequence (one of many).

- Given an array  $A[1..n]$  of integers, compute the length of a *longest alternating subsequence*. A sequence  $B[1..\ell]$  is *alternating* if  $B[i] < B[i-1]$  for every even index  $i \geq 2$ , and  $B[i] > B[i-1]$  for every odd index  $i \geq 3$ . For example, given the array

$$\langle \underline{3}, \underline{1}, \underline{4}, \underline{1}, \underline{5}, 9, \underline{2}, \underline{6}, \underline{5}, 3, 5, \underline{8}, 9, \underline{7}, \underline{9}, \underline{3}, 2, 3, \underline{8}, \underline{4}, \underline{6}, \underline{2}, 7 \rangle$$

your algorithm should return the integer 17, because  $\langle 3, 1, 4, 1, 5, 2, 6, 5, 8, 7, 9, 3, 8, 4, 6, 2, 7 \rangle$  is a longest alternating subsequence (one of many).

A **subsequence** of a sequence (for example, an array, a linked list, or a string), obtained by removing zero or more elements and keeping the rest in the same sequence order. A subsequence is called a **substring** if its elements are contiguous in the original sequence. For example:

- **SUBSEQUENCE**, **UBSEQU**, and the empty string  $\epsilon$  are all substrings of the string **SUBSEQUENCE**;
  - **SBSQNC**, **UEQUE**, and **EEE** are all subsequences of **SUBSEQUENCE** but not substrings;
  - **QUEUE**, **SSS**, and **FOOBAR** are not subsequences of **SUBSEQUENCE**.
- 

Describe and analyze **dynamic programming** algorithms for the following problems. For the first three, use the backtracking algorithms you developed on Wednesday.

1. Given an array  $A[1..n]$  of integers, compute the length of a longest **increasing** subsequence of  $A$ . A sequence  $B[1..\ell]$  is **increasing** if  $B[i] > B[i - 1]$  for every index  $i \geq 2$ .
2. Given an array  $A[1..n]$  of integers, compute the length of a longest **decreasing** subsequence of  $A$ . A sequence  $B[1..\ell]$  is **decreasing** if  $B[i] < B[i - 1]$  for every index  $i \geq 2$ .
3. Given an array  $A[1..n]$  of integers, compute the length of a longest **alternating** subsequence of  $A$ . A sequence  $B[1..\ell]$  is **alternating** if  $B[i] < B[i - 1]$  for every even index  $i \geq 2$ , and  $B[i] > B[i - 1]$  for every odd index  $i \geq 3$ .
4. Given an array  $A[1..n]$  of integers, compute the length of a longest **convex** subsequence of  $A$ . A sequence  $B[1..\ell]$  is **convex** if  $B[i] - B[i - 1] > B[i - 1] - B[i - 2]$  for every index  $i \geq 3$ .
5. Given an array  $A[1..n]$ , compute the length of a longest **palindrome** subsequence of  $A$ . Recall that a sequence  $B[1..\ell]$  is a **palindrome** if  $B[i] = B[\ell - i + 1]$  for every index  $i$ .

## Basic steps in developing a dynamic programming algorithm

1. **Formulate the problem recursively.** This is the hard part. There are two distinct but equally important things to include in your formulation.
  - (a) **Specification.** First, give a clear and precise English description of the problem you are claiming to solve. Don’t describe *how* to solve the problem at this stage; just describe *what* the problem actually is. Otherwise, the reader has no way to know what your recursive algorithm is *supposed* to compute.
  - (b) **Solution.** Second, give a clear recursive formula or algorithm for the whole problem in terms of the answers to smaller instances of *exactly* the same problem. It generally helps to think in terms of a recursive definition of your inputs and outputs. If you discover that you need a solution to a *similar* problem, or a slightly *related* problem, you’re attacking the wrong problem; go back to step 1.
2. **Build solutions to your recurrence from the bottom up.** Write an algorithm that starts with the base cases of your recurrence and works its way up to the final solution, by considering intermediate subproblems in the correct order. This stage can be broken down into several smaller, relatively mechanical steps:
  - (a) **Identify the subproblems.** What are all the different ways can your recursive algorithm call itself, starting with some initial input? For example, the argument to REC<sub>FIBO</sub> is always an integer between 0 and  $n$ .
  - (b) **Analyze space and running time.** The number of possible distinct subproblems determines the space complexity of your memoized algorithm. To compute the time complexity, add up the running times of all possible subproblems, *ignoring the recursive calls*. For example, if we already know  $F_{i-1}$  and  $F_{i-2}$ , we can compute  $F_i$  in  $O(1)$  time, so computing the first  $n$  Fibonacci numbers takes  $O(n)$  time.
  - (c) **Choose a data structure to memoize intermediate results.** For most problems, each recursive subproblem can be identified by a few integers, so you can use a multidimensional array. For some problems, however, a more complicated data structure is required.
  - (d) **Identify dependencies between subproblems.** Except for the base cases, every recursive subproblem depends on other subproblems—which ones? Draw a picture of your data structure, pick a generic element, and draw arrows from each of the other elements it depends on. Then formalize your picture.
  - (e) **Find a good evaluation order.** Order the subproblems so that each subproblem comes *after* the subproblems it depends on. Typically, this means you should consider the base cases first, then the subproblems that depends only on base cases, and so on. More formally, the dependencies you identified in the previous step define a partial order over the subproblems; in this step, you need to find a linear extension of that partial order. *Be careful!*
  - (f) **Write down the algorithm.** You know what order to consider the subproblems, and you know how to solve each subproblem. So do that! If your data structure is an array, this usually means writing a few nested for-loops around your original recurrence.

1. It’s almost time to show off your flippin’ sweet dancing skills! Tomorrow is the big dance contest you’ve been training for your entire life, except for that summer you spent with your uncle in Alaska hunting wolverines. You’ve obtained an advance copy of the the list of  $n$  songs that the judges will play during the contest, in chronological order.

You know all the songs, all the judges, and your own dancing ability extremely well. For each integer  $k$ , you know that if you dance to the  $k$ th song on the schedule, you will be awarded exactly  $\text{Score}[k]$  points, but then you will be physically unable to dance for the next  $\text{Wait}[k]$  songs (that is, you cannot dance to songs  $k + 1$  through  $k + \text{Wait}[k]$ ). The dancer with the highest total score at the end of the night wins the contest, so you want your total score to be as high as possible.

Describe and analyze an efficient algorithm to compute the maximum total score you can achieve. The input to your sweet algorithm is the pair of arrays  $\text{Score}[1 .. n]$  and  $\text{Wait}[1 .. n]$ .

2. A *shuffle* of two strings  $X$  and  $Y$  is formed by interspersing the characters into a new string, keeping the characters of  $X$  and  $Y$  in the same order. For example, the string **BANANAANANAS** is a shuffle of the strings **BANANA** and **ANANAS** in several different ways.

**BANANA ANANAS**      **BAN ANA ANA NAS**      **B AN AN A NA NA S**

Similarly, the strings **PRODGYRNAMAMMIINCG** and **DYPRONGARMAMMAGICNG** are both shuffles of **DYNAMIC** and **PROGRAMMING**:

**PRO D G Y R N A M A M M I I N C G**      **D Y P R O N G A R M A M M I C I N G**

Describe and analyze an efficient algorithm to determine, given three strings  $A[1 .. m]$ ,  $B[1 .. n]$ , and  $C[1 .. m + n]$ , whether  $C$  is a shuffle of  $A$  and  $B$ .

## Basic steps in developing a dynamic programming algorithm

1. **Formulate the problem recursively.** This is the hard part. There are two distinct but equally important things to include in your formulation.
  - (a) **Specification.** First, give a clear and precise English description of the problem you are claiming to solve. Don’t describe *how* to solve the problem at this stage; just describe *what* the problem actually is. Otherwise, the reader has no way to know what your recursive algorithm is *supposed* to compute.
  - (b) **Solution.** Second, give a clear recursive formula or algorithm for the whole problem in terms of the answers to smaller instances of *exactly* the same problem. It generally helps to think in terms of a recursive definition of your inputs and outputs. If you discover that you need a solution to a *similar* problem, or a slightly *related* problem, you’re attacking the wrong problem; go back to step 1.
2. **Build solutions to your recurrence from the bottom up.** Write an algorithm that starts with the base cases of your recurrence and works its way up to the final solution, by considering intermediate subproblems in the correct order. This stage can be broken down into several smaller, relatively mechanical steps:
  - (a) **Identify the subproblems.** What are all the different ways can your recursive algorithm call itself, starting with some initial input? For example, the argument to REC<sub>FIBO</sub> is always an integer between 0 and  $n$ .
  - (b) **Analyze space and running time.** The number of possible distinct subproblems determines the space complexity of your memoized algorithm. To compute the time complexity, add up the running times of all possible subproblems, *ignoring the recursive calls*. For example, if we already know  $F_{i-1}$  and  $F_{i-2}$ , we can compute  $F_i$  in  $O(1)$  time, so computing the first  $n$  Fibonacci numbers takes  $O(n)$  time.
  - (c) **Choose a data structure to memoize intermediate results.** For most problems, each recursive subproblem can be identified by a few integers, so you can use a multidimensional array. For some problems, however, a more complicated data structure is required.
  - (d) **Identify dependencies between subproblems.** Except for the base cases, every recursive subproblem depends on other subproblems—which ones? Draw a picture of your data structure, pick a generic element, and draw arrows from each of the other elements it depends on. Then formalize your picture.
  - (e) **Find a good evaluation order.** Order the subproblems so that each subproblem comes *after* the subproblems it depends on. Typically, this means you should consider the base cases first, then the subproblems that depends only on base cases, and so on. More formally, the dependencies you identified in the previous step define a partial order over the subproblems; in this step, you need to find a linear extension of that partial order. *Be careful!*
  - (f) **Write down the algorithm.** You know what order to consider the subproblems, and you know how to solve each subproblem. So do that! If your data structure is an array, this usually means writing a few nested for-loops around your original recurrence.

1. Suppose you are given a sequence of non-negative integers separated by + and  $\times$  signs; for example:

$$2 \times 3 + 0 \times 6 \times 1 + 4 \times 2$$

You can change the value of this expression by adding parentheses in different places. For example:

$$2 \times (3 + (0 \times (6 \times (1 + (4 \times 2))))) = 6$$

$$((((2 \times 3) + 0) \times 6) \times 1) + 4) \times 2 = 80$$

$$((2 \times 3) + (0 \times 6)) \times (1 + (4 \times 2)) = 108$$

$$((2 \times 3) + 0) \times 6) \times ((1 + 4) \times 2) = 360$$

Describe and analyze an algorithm to compute, given a list of integers separated by + and  $\times$  signs, the largest possible value we can obtain by inserting parentheses.

Your input is an array  $A[0..2n]$  where each  $A[i]$  is an integer if  $i$  is even and + or  $\times$  if  $i$  is odd. Assume any arithmetic operation in your algorithm takes  $O(1)$  time.

## Basic steps in developing a dynamic programming algorithm

1. **Formulate the problem recursively.** This is the hard part. There are two distinct but equally important things to include in your formulation.
  - (a) **Specification.** First, give a clear and precise English description of the problem you are claiming to solve. Don’t describe *how* to solve the problem at this stage; just describe *what* the problem actually is. Otherwise, the reader has no way to know what your recursive algorithm is *supposed* to compute.
  - (b) **Solution.** Second, give a clear recursive formula or algorithm for the whole problem in terms of the answers to smaller instances of *exactly* the same problem. It generally helps to think in terms of a recursive definition of your inputs and outputs. If you discover that you need a solution to a *similar* problem, or a slightly *related* problem, you’re attacking the wrong problem; go back to step 1.
2. **Build solutions to your recurrence from the bottom up.** Write an algorithm that starts with the base cases of your recurrence and works its way up to the final solution, by considering intermediate subproblems in the correct order. This stage can be broken down into several smaller, relatively mechanical steps:
  - (a) **Identify the subproblems.** What are all the different ways can your recursive algorithm call itself, starting with some initial input? For example, the argument to REC<sub>FIBO</sub> is always an integer between 0 and  $n$ .
  - (b) **Analyze space and running time.** The number of possible distinct subproblems determines the space complexity of your memoized algorithm. To compute the time complexity, add up the running times of all possible subproblems, *ignoring the recursive calls*. For example, if we already know  $F_{i-1}$  and  $F_{i-2}$ , we can compute  $F_i$  in  $O(1)$  time, so computing the first  $n$  Fibonacci numbers takes  $O(n)$  time.
  - (c) **Choose a data structure to memoize intermediate results.** For most problems, each recursive subproblem can be identified by a few integers, so you can use a multidimensional array. For some problems, however, a more complicated data structure is required.
  - (d) **Identify dependencies between subproblems.** Except for the base cases, every recursive subproblem depends on other subproblems—which ones? Draw a picture of your data structure, pick a generic element, and draw arrows from each of the other elements it depends on. Then formalize your picture.
  - (e) **Find a good evaluation order.** Order the subproblems so that each subproblem comes *after* the subproblems it depends on. Typically, this means you should consider the base cases first, then the subproblems that depends only on base cases, and so on. More formally, the dependencies you identified in the previous step define a partial order over the subproblems; in this step, you need to find a linear extension of that partial order. *Be careful!*
  - (f) **Write down the algorithm.** You know what order to consider the subproblems, and you know how to solve each subproblem. So do that! If your data structure is an array, this usually means writing a few nested for-loops around your original recurrence.

Recall the class scheduling problem described in lecture on Tuesday. We are given two arrays  $S[1..n]$  and  $F[1..n]$ , where  $S[i] < F[i]$  for each  $i$ , representing the start and finish times of  $n$  classes. Your goal is to find the largest number of classes you can take without ever taking two classes simultaneously. We showed in class that the following greedy algorithm constructs an optimal schedule:

Choose the course that *ends first*, discard all conflicting classes, and recurse.

But this is not the only greedy strategy we could have tried. For each of the following alternative greedy algorithms, either prove that the algorithm always constructs an optimal schedule, or describe a small input example for which the algorithm does not produce an optimal schedule. Assume that all algorithms break ties arbitrarily (that is, in a manner that is completely out of your control).

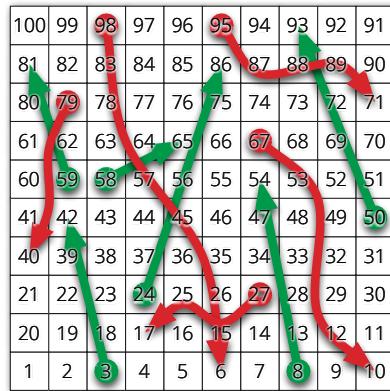
[Hint: Exactly three of these greedy strategies actually work.]

1. Choose the course  $x$  that *ends last*, discard classes that conflict with  $x$ , and recurse.
2. Choose the course  $x$  that *starts first*, discard all classes that conflict with  $x$ , and recurse.
3. Choose the course  $x$  that *starts last*, discard all classes that conflict with  $x$ , and recurse.
4. Choose the course  $x$  with *shortest duration*, discard all classes that conflict with  $x$ , and recurse.
5. Choose a course  $x$  that *conflicts with the fewest other courses*, discard all classes that conflict with  $x$ , and recurse.
6. If no classes conflict, choose them all. Otherwise, discard the course with *longest duration* and recurse.
7. If no classes conflict, choose them all. Otherwise, discard a course that *conflicts with the most other courses* and recurse.
8. Let  $x$  be the class with the *earliest start time*, and let  $y$  be the class with the *second earliest start time*.
  - If  $x$  and  $y$  are disjoint, choose  $x$  and recurse on everything but  $x$ .
  - If  $x$  completely contains  $y$ , discard  $x$  and recurse.
  - Otherwise, discard  $y$  and recurse.
9. If any course  $x$  completely contains another course, discard  $x$  and recurse. Otherwise, choose the course  $y$  that *ends last*, discard all classes that conflict with  $y$ , and recurse.

For each of the problems below, transform the input into a graph and apply a standard graph algorithm that you’ve seen in class. Whenever you use a standard graph algorithm, you **must** provide the following information. (I recommend actually using a bulleted list.)

- What are the vertices?
- What are the edges? Are they directed or undirected?
- If the vertices and/or edges have associated values, what are they?
- What problem do you need to solve on this graph?
- What standard algorithm are you using to solve that problem?
- What is the running time of your entire algorithm, *including* the time to build the graph, *as a function of the original input parameters*?

1. **Snakes and Ladders** is a classic board game, originating in India no later than the 16th century. The board consists of an  $n \times n$  grid of squares, numbered consecutively from 1 to  $n^2$ , starting in the bottom left corner and proceeding row by row from bottom to top, with rows alternating to the left and right. Certain pairs of squares, always in different rows, are connected by either “snakes” (leading down) or “ladders” (leading up). Each square can be an endpoint of at most one snake or ladder.



A typical Snakes and Ladders board.  
Upward straight arrows are ladders; downward wavy arrows are snakes.

You start with a token in cell 1, in the bottom left corner. In each move, you advance your token up to  $k$  positions, for some fixed constant  $k$  (typically 6). If the token ends the move at the *top* end of a snake, you **must** slide the token down to the bottom of that snake. If the token ends the move at the *bottom* end of a ladder, you **may** move the token up to the top of that ladder.

Describe and analyze an algorithm to compute the smallest number of moves required for the token to reach the last square of the grid.

2. Let  $G$  be a connected undirected graph. Suppose we start with two coins on two arbitrarily chosen vertices of  $G$ . At every step, each coin *must* move to an adjacent vertex. Describe and analyze an algorithm to compute the minimum number of steps to reach a configuration where both coins are on the same vertex, or to report correctly that no such configuration is reachable. The input to your algorithm consists of a graph  $G = (V, E)$  and two vertices  $u, v \in V$  (which may or may not be distinct).

For each of the problems below, transform the input into a graph and apply a standard graph algorithm that you’ve seen in class. Whenever you use a standard graph algorithm, you **must** provide the following information. (I recommend actually using a bulleted list.)

- What are the vertices?
  - What are the edges? Are they directed or undirected?
  - If the vertices and/or edges have associated values, what are they?
  - What problem do you need to solve on this graph?
  - What standard algorithm are you using to solve that problem?
  - What is the running time of your entire algorithm, *including* the time to build the graph, *as a function of the original input parameters*?
- 

1. Inspired by the previous lab, you decided to organize a Snakes and Ladders competition with  $n$  participants. In this competition, each game of Snakes and Ladders involves three players. After the game is finished, they are ranked first, second and third. Each player may be involved in any (non-negative) number of games, and the number needs not be equal among players.

At the end of the competition,  $m$  games have been played. You realized that you had forgotten to implement a proper rating system, and therefore decided to produce the overall ranking of all  $n$  players as you see fit. However, to avoid being too suspicious, if player  $A$  ranked better than player  $B$  in any game, then  $A$  must rank better than  $B$  in the overall ranking.

You are given the list of players involved and the ranking in each of the  $m$  games. Describe and analyze an algorithm to produce an overall ranking of the  $n$  players that satisfies the condition, or correctly reports that it is impossible.

2. There are  $n$  galaxies connected by  $m$  intergalactic teleport-ways. Each teleport-way joins two galaxies and can be traversed in both directions. Also, each teleport-way  $e$  has an associated toll of  $c_e$  dollars, where  $c_e$  is a positive integer. A teleport-way can be used multiple times, but the toll must be paid every time it is used.

Judy wants to travel from galaxy  $u$  to galaxy  $v$ , but teleportation is not very pleasant and she would like to minimize the number of times she needs to teleport. However, she wants the total cost to be a multiple of five dollars, because carrying small bills is not pleasant either.

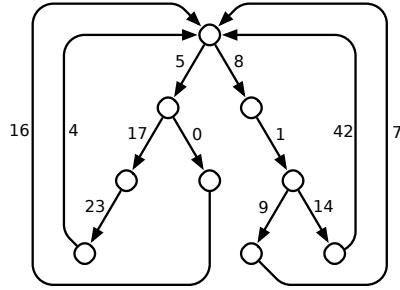
- (a) Describe and analyze an algorithm to compute the smallest number of times Judy needs to teleport to travel from galaxy  $u$  to galaxy  $v$  while the total cost is a multiple of five dollars.
- (b) Solve (a), but now assume that Judy has a coupon that allows her to waive the toll once.

Suppose we are given both an undirected graph  $G$  with weighted edges and a minimum spanning tree  $T$  of  $G$ .

1. Describe an efficient algorithm to update the minimum spanning tree when the weight of one edge  $e \in T$  is decreased.
2. Describe an efficient algorithm to update the minimum spanning tree when the weight of one edge  $e \notin T$  is increased.
3. Describe an efficient algorithm to update the minimum spanning tree when the weight of one edge  $e \in T$  is increased.
4. Describe an efficient algorithm to update the minimum spanning tree when the weight of one edge  $e \notin T$  is decreased.

In all cases, the input to your algorithm is the edge  $e$  and its new weight; your algorithms should modify  $T$  so that it is still a minimum spanning tree. Of course, we could just recompute the minimum spanning tree from scratch in  $O(E \log V)$  time, but you can do better.

1. A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has non-negative weight.



A looped tree.

- (a) How much time would Dijkstra’s algorithm require to compute the shortest path between two vertices  $u$  and  $v$  in a looped tree with  $n$  nodes?
  - (b) Describe and analyze a faster algorithm.
2. After graduating you accept a job with Aerophobes-R-Us, the leading traveling agency for people who hate to fly. Your job is to build a system to help customers plan airplane trips from one city to another. All of your customers are afraid of flying (and by extension, airports), so any trip you plan needs to be as short as possible. You know all the departure and arrival times of all the flights on the planet.

Suppose one of your customers wants to fly from city  $X$  to city  $Y$ . Describe an algorithm to find a sequence of flights that minimizes the *total time in transit*—the length of time from the initial departure to the final arrival, including time at intermediate airports waiting for connecting flights. [Hint: Build an appropriate graph from the input data and apply Dijkstra’s algorithm.]

Describe Turing machines that compute the following functions.

In particular, specify the transition functions  $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{-1, +1\}$  for each machine either by writing out a table or by drawing a graph. Recall that  $\delta(p, \$) = (q, @, +1)$  means that if the Turing machine is in state  $p$  and reads the symbol  $\$$  from the tape, then it will change to state  $q$ , write the symbol  $@$  to the tape, and move one step to the right. In a *drawing* of a Turing machine, this transition is indicated by an edge from  $p$  to  $q$  with the label “ $\$/@, +1$ ”.

***Give your states short mnemonic names that suggest their purpose.*** Naming your states well won’t just make it easier to understand; it will also make it easier to design.

---

1. DOUBLE: Given a string  $w \in \{0, 1\}^*$  as input, return the string  $ww$  as output.
  2. POWER: Given a string of the form  $1^n$  as input, return the string  $1^{2^n}$  as output.
-

Describe how to simulate an arbitrary Turing machine to make it *error-tolerant*. Specifically, given an arbitrary Turing machine  $M$ , describe a new Turing machine  $M'$  that accepts and rejects exactly the same strings as  $M$ , even though an evil pixie named Lenny will move the head of  $M'$  to an *arbitrary* location on the tape some finite number of *unknown* times during the execution of  $M'$ .

You do not have to describe  $M'$  in complete detail, but do give enough details that a seasoned Turing machine programmer could work out the remaining mechanical details.

---

**As stated, this problem has no solution!** If  $M$  halts on all inputs after a finite number of steps, then Lenny can make any substring of the input string completely invisible to  $M$ . For example, if the true input string is **INPUT-STRING**, Lenny can make  $M$  believe the input string is actually **IMPING**, by moving the head to the second **I** whenever it tries to move to **R**, and by moving the head to **P** when it tries to move to **U**. Because  $M$  halts after a finite number of steps, Lenny only has a finite number of opportunities to move the head.

In fact, with more care, Lenny can make  $M$  think the input string is *any* string that uses only symbols from the actual input string; if the true input string is **INPUT-STRING**, Lenny can make  $M$  believe the input string is actually **GRINNING-PUTIN-IS-GRINNING**.)

However, there are several different ways to rescue the problem. For each of the following restrictions on Lenny’s behavior, and for any Turing machine  $M$ , one can design a Turing machine  $M'$  that simulates  $M$  despite Lenny’s interference.

- Lenny can move the head only a *bounded* number of times. For example: Lenny can move the head at most 374 times.
  - Whenever Lenny moves the head, he changes the state of the machine to a special error state **lenny**.
  - Whenever Lenny moves the head, he moves it to the left end of the tape.
  - Whenever Lenny moves the head, he moves it to a blank cell to the right of all non-blank cells.
  - Whenever Lenny moves the head, he moves it to a cell containing a particular symbol in the input alphabet, say **0**.
-

Describe algorithms for the following problems. The input for each problem is string  $\langle M, w \rangle$  that encodes a standard (one-tape, one-track, one-head) Turing machine  $M$  whose tape alphabet is  $\{\text{0}, \text{1}, \square\}$  and a string  $w \in \{\text{0}, \text{1}\}^*$ .

1. Does  $M$  accept  $w$  after at most  $|w|^2$  steps?
  2. If we run  $M$  with input  $w$ , does  $M$  ever move its head to the right?
  - $2\frac{1}{2}$ . If we run  $M$  with input  $w$ , does  $M$  ever move its head to the right twice in a row?
  - $2\frac{3}{4}$ . If we run  $M$  with input  $w$ , does  $M$  move its head to the right more than  $2^{|w|}$  times?
  3. If we run  $M$  with input  $w$ , does  $M$  ever change a symbol on the tape?
  - $3\frac{1}{2}$ . If we run  $M$  with input  $w$ , does  $M$  ever change a  $\square$  on the tape to either  $\text{0}$  or  $\text{1}$ ?
  4. If we run  $M$  with input  $w$ , does  $M$  ever leave its **start** state?
- 
- 

In contrast, as we will see later, the following problems are all undecidable!

1. Does  $M$  accept  $w$ ?
- $1\frac{1}{2}$ . If we run  $M$  with input  $w$ , does  $M$  ever halt?
2. If we run  $M$  with input  $w$ , does  $M$  ever move its head to the right three times in a row?
3. If we run  $M$  with input  $w$ , does  $M$  ever change a  $\square$  on the tape to  $\text{1}$ ?
- $3\frac{1}{2}$ . If we run  $M$  with input  $w$ , does  $M$  ever change either  $\text{0}$  or  $\text{1}$  on the tape to  $\square$ ?
4. If we run  $M$  with input  $w$ , does  $M$  ever reenter its **start** state?

1. Suppose you are given a magic black box that somehow answers the following decision problem in *polynomial time*:

- INPUT: A boolean circuit  $K$  with  $n$  inputs and one output .
- OUTPUT: TRUE if there are input values  $x_1, x_2, \dots, x_n \in \{\text{TRUE}, \text{FALSE}\}$  that make  $K$  output TRUE, and FALSE otherwise.

Using this black box as a subroutine, describe an algorithm that solves the following related search problem *in polynomial time*:

- INPUT: A boolean circuit  $K$  with  $n$  inputs and one output.
- OUTPUT: Input values  $x_1, x_2, \dots, x_n \in \{\text{TRUE}, \text{FALSE}\}$  that make  $K$  output TRUE, or NONE if there are no such inputs.

[Hint: You can use the magic box more than once.]

2. Formally, **valid 3-coloring** of a graph  $G = (V, E)$  is a function  $c : V \rightarrow \{1, 2, 3\}$  such that  $c(u) \neq c(v)$  for all  $uv \in E$ . Less formally, a valid 3-coloring assigns each vertex a color, which is either red, green, or blue, such that the endpoints of every edge have different colors.

Suppose you are given a magic black box that somehow answers the following problem *in polynomial time*:

- INPUT: An undirected graph  $G$ .
- OUTPUT: TRUE if  $G$  has a valid 3-coloring, and FALSE otherwise.

Using this black box as a subroutine, describe an algorithm that solves the **3-coloring problem** *in polynomial time*:

- INPUT: An undirected graph  $G$ .
- OUTPUT: A valid 3-coloring of  $G$ , or NONE if there is no such coloring.

[Hint: You can use the magic box more than once. The input to the magic box is a graph and **only** a graph, meaning **only** vertices and edges.]

Proving that a problem  $X$  is NP-hard requires several steps:

- Choose a problem  $Y$  that you already know is NP-hard.
  - Describe an algorithm to solve  $Y$ , using an algorithm for  $X$  as a subroutine. Typically this algorithm has the following form: Given an instance of  $Y$ , transform it into an instance of  $X$ , and then call the magic black-box algorithm for  $X$ .
  - Prove that your algorithm is correct. This almost always requires two separate steps:
    - Prove that your algorithm transforms “good” instances of  $Y$  into “good” instances of  $X$ .
    - Prove that your algorithm transforms “bad” instances of  $Y$  into “bad” instances of  $X$ . Equivalently: Prove that if your transformation produces a “good” instance of  $X$ , then it was given a “good” instance of  $Y$ .
  - Argue that your algorithm for  $Y$  runs in polynomial time.
- 

1. Recall the following  $k$ COLOR problem: Given an undirected graph  $G$ , can its vertices be colored with  $k$  colors, so that every edge touches vertices with two different colors?
  - (a) Describe a direct polynomial-time reduction from 3COLOR to 4COLOR.
  - (b) Prove that  $k$ COLOR problem is NP-hard for any  $k \geq 3$ .
2. Recall that a *Hamiltonian cycle* in a graph  $G$  is a cycle that goes through every vertex of  $G$  exactly once. Now, a **tonian cycle** in a graph  $G$  is a cycle that goes through at least *half* of the vertices of  $G$ , and a **Hamilhamiltonian circuit** in a graph  $G$  is a closed walk that goes through every vertex in  $G$  exactly *twice*.
  - (a) Prove that it is NP-hard to determine whether a given graph contains a tonian cycle.
  - (b) Prove that it is NP-hard to determine whether a given graph contains a Hamilhamiltonian circuit.

Proving that a problem  $X$  is NP-hard requires several steps:

- Choose a problem  $Y$  that you already know is NP-hard.
  - Describe an algorithm to solve  $Y$ , using an algorithm for  $X$  as a subroutine. Typically this algorithm has the following form: Given an instance of  $Y$ , transform it into an instance of  $X$ , and then call the magic black-box algorithm for  $X$ .
  - Prove that your algorithm is correct. This almost always requires two separate steps:
    - Prove that your algorithm transforms “good” instances of  $Y$  into “good” instances of  $X$ .
    - Prove that your algorithm transforms “bad” instances of  $Y$  into “bad” instances of  $X$ . Equivalently: Prove that if your transformation produces a “good” instance of  $X$ , then it was given a “good” instance of  $Y$ .
  - Argue that your algorithm for  $Y$  runs in polynomial time.
- 

Recall that a *Hamiltonian cycle* in a graph  $G$  is a cycle that visits every vertex of  $G$  exactly once.

1. In class on Thursday, Jeff proved that it is NP-hard to determine whether a given *directed* graph contains a Hamiltonian cycle. Prove that it is NP-hard to determine whether a given *undirected* graph contains a Hamiltonian cycle.
2. A *double Hamiltonian circuit* in a graph  $G$  is a closed walk that goes through every vertex in  $G$  exactly *twice*. Prove that it is NP-hard to determine whether a given *undirected* graph contains a double Hamiltonian circuit.

Proving that a language  $L$  is undecidable by reduction requires several steps:

- Choose a language  $L'$  that you already know is undecidable. Typical choices for  $L'$  include:

$$\begin{aligned} \text{ACCEPT} &:= \{\langle M, w \rangle \mid M \text{ accepts } w\} \\ \text{REJECT} &:= \{\langle M, w \rangle \mid M \text{ rejects } w\} \\ \text{HALT} &:= \{\langle M, w \rangle \mid M \text{ halts on } w\} \\ \text{DIVERGE} &:= \{\langle M, w \rangle \mid M \text{ diverges on } w\} \\ \text{NEVERACCEPT} &:= \{\langle M \rangle \mid \text{ACCEPT}(M) = \emptyset\} \\ \text{NEVERREJECT} &:= \{\langle M \rangle \mid \text{REJECT}(M) = \emptyset\} \\ \text{NEVERHALT} &:= \{\langle M \rangle \mid \text{HALT}(M) = \emptyset\} \\ \text{NEVERDIVERGE} &:= \{\langle M \rangle \mid \text{DIVERGE}(M) = \emptyset\} \end{aligned}$$

- Describe an algorithm (really a Turing machine)  $M'$  that decides  $L'$ , using a Turing machine  $M$  that decides  $L$  as a black box. Typically this algorithm has the following form:

Given a string  $w$ , transform it into another string  $x$ ,  
such that  $M$  accepts  $x$  if and only if  $w \in L'$ .

- Prove that your Turing machine is correct. This almost always requires two separate steps:
  - Prove that if  $M$  accepts  $w$  then  $w \in L'$ .
  - Prove that if  $M$  rejects  $w$  then  $w \notin L'$ .

Prove that the following languages are undecidable:

1. ACCEPTILLINI :=  $\{\langle M \rangle \mid M \text{ accepts the string ILLINI}\}$
2. ACCEPTTHREE :=  $\{\langle M \rangle \mid M \text{ accepts exactly three strings}\}$
3. ACCEPTPALINDROME :=  $\{\langle M \rangle \mid M \text{ accepts at least one palindrome}\}$

Prove that the following languages are undecidable *using Rice’s Theorem*:

**Rice’s Theorem.** Let  $\mathcal{X}$  be any nonempty proper subset of the set of acceptable languages. The language  $\text{ACCEPTIN}\mathcal{X} := \{\langle M \rangle \mid \text{ACCEPT}(M) \in \mathcal{X}\}$  is undecidable.

1.  $\text{ACCEPTREGULAR} := \{\langle M \rangle \mid \text{ACCEPT}(M) \text{ is regular}\}$
2.  $\text{ACCEPTILLINI} := \{\langle M \rangle \mid M \text{ accepts the string } \text{ILLINI}\}$
3.  $\text{ACCEPTPALINDROME} := \{\langle M \rangle \mid M \text{ accepts at least one palindrome}\}$
4.  $\text{ACCEPTTHREE} := \{\langle M \rangle \mid M \text{ accepts exactly three strings}\}$
5.  $\text{ACCEPTUNDECIDABLE} := \{\langle M \rangle \mid \text{ACCEPT}(M) \text{ is undecidable}\}$

**To think about later.** Which of the following languages are undecidable? How do you prove it?

1.  $\text{ACCEPT}\{\{\varepsilon\}\} := \{\langle M \rangle \mid M \text{ only accepts the string } \varepsilon, \text{ i.e. } \text{ACCEPT}(M) = \{\varepsilon\}\}$
2.  $\text{ACCEPT}\{\emptyset\} := \{\langle M \rangle \mid M \text{ does not accept any strings, i.e. } \text{ACCEPT}(M) = \emptyset\}$
3.  $\text{ACCEPT}\emptyset := \{\langle M \rangle \mid \text{ACCEPT}(M) \text{ is not an acceptable language}\}$
4.  $\text{ACCEPT}=\text{REJECT} := \{\langle M \rangle \mid \text{ACCEPT}(M) = \text{REJECT}(M)\}$
5.  $\text{ACCEPT}\neq\text{REJECT} := \{\langle M \rangle \mid \text{ACCEPT}(M) \neq \text{REJECT}(M)\}$
6.  $\text{ACCEPT}\cup\text{REJECT} := \{\langle M \rangle \mid \text{ACCEPT}(M) \cup \text{REJECT}(M) = \Sigma^*\}$

**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

1. For each statement below, check “True” if the statement is *always* true and “False” otherwise. Each correct answer is worth +1 point; each incorrect answer is worth  $-\frac{1}{2}$  point; checking “I don’t know” is worth  $+\frac{1}{4}$  point; and flipping a coin is (on average) worth  $+\frac{1}{4}$  point. You do **not** need to prove your answer is correct.

**Read each statement *very carefully*.** Some of these are deliberately subtle.

- (a) If  $2 + 2 = 5$ , then Jeff is the Queen of England.
  - (b) For all languages  $L_1$  and  $L_2$ , the language  $L_1 \cup L_2$  is regular.
  - (c) For all languages  $L \subseteq \Sigma^*$ , if  $L$  is not regular, then  $\Sigma^* \setminus L$  cannot be represented by a regular expression.
  - (d) For all languages  $L_1$  and  $L_2$ , if  $L_1 \subseteq L_2$  and  $L_1$  is regular, then  $L_2$  is regular.
  - (e) For all languages  $L_1$  and  $L_2$ , if  $L_1 \subseteq L_2$  and  $L_1$  is not regular, then  $L_2$  is not regular.
  - (f) For all languages  $L$ , if  $L$  is regular, then  $L$  has no infinite fooling set.
  - (g) The language  $\{\textcolor{red}{0}^m 1^n \mid 0 \leq m + n \leq 374\}$  is regular.
  - (h) The language  $\{\textcolor{red}{0}^m 1^n \mid 0 \leq m - n \leq 374\}$  is regular.
  - (i) For every language  $L$ , if the language  $L^R = \{w^R \mid w \in L\}$  is regular, then  $L$  is also regular.  
(Here  $w^R$  denotes the reversal of string  $w$ ; for example,  $(\text{BACKWARD})^R = \text{DRAWK CAB}$ .)
  - (j) Every context-free language is regular.
- 
2. Let  $L$  be the set of strings in  $\{0, 1\}^*$  in which every run of consecutive 0s has even length and every run of consecutive 1s has odd length.
    - (a) Give a regular expression that represents  $L$ .
    - (b) Construct a DFA that recognizes  $L$ .
- You do **not** need to prove that your answers are correct.
- 
3. For each of the following languages over the alphabet  $\{0, 1\}$ , either *prove* that the language is regular or *prove* that the language is not regular. *Exactly one of these two languages is regular.*
    - (a) The set of all strings in which the substrings  $00$  and  $11$  appear the same number of times.
    - (b) The set of all strings in which the substrings  $01$  and  $10$  appear the same number of times.

For example, both of these languages contain the string  $\textcolor{red}{1100001101101}$ .

4. Consider the following recursive function:

$$\text{stutter}(w) := \begin{cases} \epsilon & \text{if } w = \epsilon \\ aa \cdot \text{stutter}(x) & \text{if } w = ax \text{ for some } a \in \Sigma \text{ and } x \in \Sigma^* \end{cases}$$

For example,  $\text{stutter}(00101) = 0000110011$ .

**Prove** that for any regular language  $L$ , the following languages are also regular.

- (a)  $\text{STUTTER}(L) := \{\text{stutter}(w) \mid w \in L\}$ .
- (b)  $\text{STUTTER}^{-1}(L) := \{w \mid \text{stutter}(w) \in L\}$ .

5. Recall that string concatenation and string reversal are formally defined as follows:

$$w \bullet y := \begin{cases} y & \text{if } w = \epsilon \\ a \cdot (x \bullet y) & \text{if } w = ax \text{ for some } a \in \Sigma \text{ and } x \in \Sigma^* \end{cases}$$

$$w^R := \begin{cases} \epsilon & \text{if } w = \epsilon \\ x^R \bullet a & \text{if } w = ax \text{ for some } a \in \Sigma \text{ and } x \in \Sigma^* \end{cases}$$

**Prove** that  $(w \bullet x)^R = x^R \bullet w^R$ , for all strings  $w$  and  $x$ . Your proof should be complete, concise, formal, and self-contained.

**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

- For each statement below, check “True” if the statement is *always* true and “False” otherwise. Each correct answer is worth +1 point; each incorrect answer is worth  $-\frac{1}{2}$  point; checking “I don’t know” is worth  $+\frac{1}{4}$  point; and flipping a coin is (on average) worth  $+\frac{1}{4}$  point. You do **not** need to prove your answer is correct.

**Read each statement very carefully.** Some of these are deliberately subtle.

- If  $2 + 2 = 5$ , then Jeff is not the Queen of England.
  - For all languages  $L$ , the language  $L^*$  is regular.
  - For all languages  $L \subseteq \Sigma^*$ , if  $L$  can be represented by a regular expression, then  $\Sigma^* \setminus L$  can also be represented by a regular expression.
  - For all languages  $L_1$  and  $L_2$ , if  $L_2$  is regular and  $L_1 \subseteq L_2$ , then  $L_1$  is regular.
  - For all languages  $L_1$  and  $L_2$ , if  $L_2$  is not regular and  $L_1 \subseteq L_2$ , then  $L_1$  is not regular.
  - For all languages  $L$ , if  $L$  is not regular, then every fooling set for  $L$  is infinite.
  - The language  $\{\texttt{0}^m \texttt{1}\texttt{0}^n \mid 0 \leq n - m \leq 374\}$  is regular.
  - The language  $\{\texttt{0}^m \texttt{1}\texttt{0}^n \mid 0 \leq n + m \leq 374\}$  is regular.
  - For every language  $L$ , if  $L$  is not regular, then the language  $L^R = \{w^R \mid w \in L\}$  is also not regular. (Here  $w^R$  denotes the reversal of string  $w$ ; for example,  $(\text{BACKWARD})^R = \text{DRAWKCAB}$ .)
  - Every context-free language is regular.
- Let  $L$  be the set of strings in  $\{\texttt{0}, \texttt{1}\}^*$  in which every run of consecutive **0**s has odd length and the total number of **1**s is even.

For example, the string **11110000010111000** is in  $L$ , because it has eight **1**s and three runs of consecutive **0**s, with lengths 5, 1, and 3.

- Give a regular expression that represents  $L$ .
- Construct a DFA that recognizes  $L$ .

You do **not** need to prove that your answers are correct.

- For each of the following languages over the alphabet  $\{\texttt{0}, \texttt{1}\}$ , either *prove* that the language is regular or *prove* that the language is not regular. *Exactly one of these two languages is regular.*

- The set of all strings in which the substrings **10** and **01** appear the same number of times.
- The set of all strings in which the substrings **00** and **01** appear the same number of times.

For example, both of these languages contain the string **1100001101101**.

4. Consider the following recursive function:

$$\text{odds}(w) := \begin{cases} w & \text{if } |w| \leq 1 \\ a \cdot \text{odds}(x) & \text{if } w = abx \text{ for some } a, b \in \Sigma \text{ and } x \in \Sigma^* \end{cases}$$

Intuitively, *odds* removes every other symbol from the input string, starting with the second symbol. For example,  $\text{odds}(\textcolor{red}{0101110}) = \textcolor{red}{0010}$ .

**Prove** that for any regular language  $L$ , the following languages are also regular.

- (a)  $\text{ODDS}(L) := \{ \text{odds}(w) \mid w \in L \}$ .
- (b)  $\text{ODDS}^{-1}(L) := \{ w \mid \text{odds}(w) \in L \}$ .

5. Recall that string concatenation and string reversal are formally defined as follows:

$$w \bullet y := \begin{cases} y & \text{if } w = \epsilon \\ a \cdot (x \bullet y) & \text{if } w = ax \text{ for some } a \in \Sigma \text{ and } x \in \Sigma^* \end{cases}$$

$$w^R := \begin{cases} \epsilon & \text{if } w = \epsilon \\ x^R \bullet a & \text{if } w = ax \text{ for some } a \in \Sigma \text{ and } x \in \Sigma^* \end{cases}$$

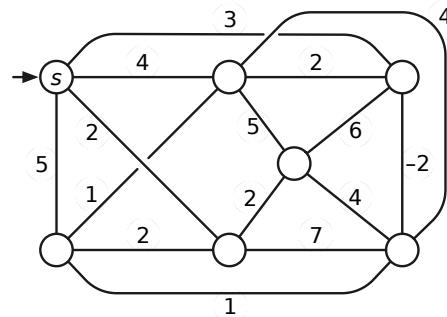
**Prove** that  $(w \bullet x)^R = x^R \bullet w^R$ , for all strings  $w$  and  $x$ . Your proof should be complete, concise, formal, and self-contained. You may assume the following identities, which we proved in class:

- $w \bullet (x \bullet y) = (w \bullet x) \bullet y$  for all strings  $w$ ,  $x$ , and  $y$ .
- $|w \bullet x| = |w| + |x|$  for all strings  $w$  and  $x$ .

**Write your answers in the separate answer booklet.**  
 Please return this question sheet and your cheat sheet with your answers.

1. *Clearly* indicate the edges of the following spanning trees of the weighted graph pictured below. (Pretend that the person grading your exam has bad eyesight.) Some of these subproblems have more than one correct answer. Yes, that edge on the right has negative weight.

- (a) A depth-first spanning tree rooted at  $s$
- (b) A breadth-first spanning tree rooted at  $s$
- (c) A shortest-path tree rooted at  $s$
- (d) A minimum spanning tree



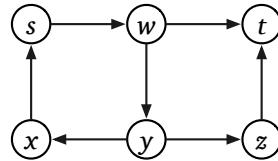
2. An array  $A[0..n-1]$  of  $n$  distinct numbers is **bitonic** if there are unique indices  $i$  and  $j$  such that  $A[(i-1) \bmod n] < A[i] > A[(i+1) \bmod n]$  and  $A[(j-1) \bmod n] > A[j] < A[(j+1) \bmod n]$ . In other words, a bitonic sequence either consists of an increasing sequence followed by a decreasing sequence, or can be circularly shifted to become so. For example,

4   6   9   8   7   5   1   2   3	is bitonic, but
3   6   9   8   7   5   1   2   4	is <i>not</i> bitonic.

Describe and analyze an algorithm to find the index of the *smallest* element in a given bitonic array  $A[0..n-1]$  in  $O(\log n)$  time. You may assume that the numbers in the input array are distinct. For example, given the first array above, your algorithm should return 6, because  $A[6] = 1$  is the smallest element in that array.

3. Suppose you are given a directed graph  $G = (V, E)$  and two vertices  $s$  and  $t$ . Describe and analyze an algorithm to determine if there is a walk in  $G$  from  $s$  to  $t$  (possibly repeating vertices and/or edges) whose length is divisible by 3.

For example, given the graph below, with the indicated vertices  $s$  and  $t$ , your algorithm should return TRUE, because the walk  $s \rightarrow w \rightarrow y \rightarrow x \rightarrow s \rightarrow w \rightarrow t$  has length 6.



[Hint: Build a (different) graph.]

4. The new swap-puzzle game *Candy Swap Saga XIII* involves  $n$  cute animals numbered 1 through  $n$ . Each animal holds one of three types of candy: circus peanuts, Heath bars, and Cioccolateria Gardini chocolate truffles. You also have a candy in your hand; at the start of the game, you have a circus peanut.

To earn points, you visit each of the animals in order from 1 to  $n$ . For each animal, you can either keep the candy in your hand or exchange it with the candy the animal is holding.

- If you swap your candy for another candy of the *same* type, you earn one point.
- If you swap your candy for a candy of a *different* type, you lose one point. (Yes, your score can be negative.)
- If you visit an animal and decide not to swap candy, your score does not change.

You *must* visit the animals in order, and once you visit an animal, you can never visit it again.

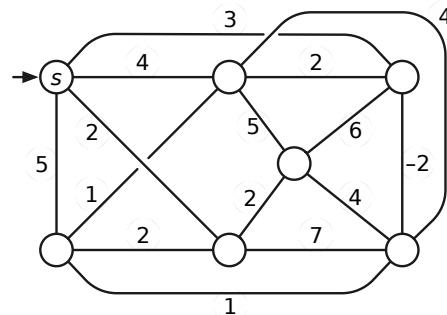
Describe and analyze an efficient algorithm to compute your maximum possible score. Your input is an array  $C[1..n]$ , where  $C[i]$  is the type of candy that the  $i$ th animal is holding.

5. Let  $G$  be a directed graph with weighted edges, and let  $s$  be a vertex of  $G$ . Suppose every vertex  $v \neq s$  stores a pointer  $\text{pred}(v)$  to another vertex in  $G$ . Describe and analyze an algorithm to determine whether these predecessor pointers correctly define a single-source shortest path tree rooted at  $s$ . Do **not** assume that  $G$  has no negative cycles.

**Write your answers in the separate answer booklet.**  
Please return this question sheet and your cheat sheet with your answers.

1. *Clearly* indicate the edges of the following spanning trees of the weighted graph pictured below. (Pretend that the person grading your exam has bad eyesight.) Some of these subproblems have more than one correct answer. Yes, that edge on the right has negative weight.

- (a) A depth-first spanning tree rooted at  $s$
- (b) A breadth-first spanning tree rooted at  $s$
- (c) A shortest-path tree rooted at  $s$
- (d) A minimum spanning tree



2. Farmers Boggis, Bunce, and Bean have set up an obstacle course for Mr. Fox. The course consists of a row of  $n$  booths, each with an integer painted on the front with bright red paint, which could be positive, negative, or zero. Let  $A[i]$  denote the number painted on the front of the  $i$ th booth. Everyone has agreed to the following rules:

- At each booth, Mr. Fox *must* say either “Ring!” or “Ding!”.
- If Mr. Fox says “Ring!” at the  $i$ th booth, he earns a reward of  $A[i]$  chickens. (If  $A[i] < 0$ , Mr. Fox pays a penalty of  $-A[i]$  chickens.)
- If Mr. Fox says “Ding!” at the  $i$ th booth, he pays a penalty of  $A[i]$  chickens. (If  $A[i] < 0$ , Mr. Fox earns a reward of  $-A[i]$  chickens.)
- Mr. Fox is forbidden to say the same word more than three times in a row. For example, if he says “Ring!” at booths 6, 7, and 8, then he *must* say “Ding!” at booth 9.
- All accounts will be settled at the end; Mr. Fox does not actually have to carry chickens through the obstacle course.
- If Mr. Fox violates any of the rules, or if he ends the obstacle course owing the farmers chickens, the farmers will shoot him.

Describe and analyze an algorithm to compute the largest number of chickens that Mr. Fox can earn by running the obstacle course, given the array  $A[1..n]$  of booth numbers as input.

3. Let  $G$  be a directed graph with weighted edges, and let  $s$  be a vertex of  $G$ . Suppose every vertex  $v \neq s$  stores a pointer  $\text{pred}(v)$  to another vertex in  $G$ . Describe and analyze an algorithm to determine whether these predecessor pointers correctly define a single-source shortest path tree rooted at  $s$ . Do **not** assume that  $G$  has no negative cycles.

4. An array  $A[0..n-1]$  of  $n$  distinct numbers is **bitonic** if there are unique indices  $i$  and  $j$  such that  $A[(i-1) \bmod n] < A[i] > A[(i+1) \bmod n]$  and  $A[(j-1) \bmod n] > A[j] < A[(j+1) \bmod n]$ . In other words, a bitonic sequence either consists of an increasing sequence followed by a decreasing sequence, or can be circularly shifted to become so. For example,

4   6   9   8   7   5   1   2   3	is bitonic, but
-----------------------------------	-----------------

3   6   9   8   7   5   1   2   4	is <i>not</i> bitonic.
-----------------------------------	------------------------

Describe and analyze an algorithm to find the index of the *smallest* element in a given bitonic array  $A[0..n-1]$  in  $O(\log n)$  time. You may assume that the numbers in the input array are distinct. For example, given the first array above, your algorithm should return 6, because  $A[6] = 1$  is the smallest element in that array.

5. Suppose we are given an undirected graph  $G$  in which every *vertex* has a positive weight.
- Describe and analyze an algorithm to find a *spanning tree* of  $G$  with minimum total weight. (The total weight of a spanning tree is the sum of the weights of its vertices.)
  - Describe and analyze an algorithm to find a *path* in  $G$  from one given vertex  $s$  to another given vertex  $t$  with minimum total weight. (The total weight of a path is the sum of the weights of its vertices.)

**“CS 374”: Algorithms and Models of Computation, Fall 2014**  
**Final Exam — Version A — December 16, 2014**

Name:					
NetID:					
Section:	1	2	3		

#	1	2	3	4	5	6	Total
Score							
Max	20	10	10	10	10	10	70
Grader							

- 
- ***Don’t panic!***
  - Please print your name and your NetID and circle your discussion section in the boxes above.
  - This is a closed-book, closed-notes, closed-electronics exam. If you brought anything except your writing implements and your two double-sided 8½" × 11" cheat sheets, please put it away for the duration of the exam. In particular, you may not use *any* electronic devices.
  - **Please read the entire exam before writing anything.** Please ask for clarification if any question is unclear.
  - **You have 180 minutes.**
  - If you run out of space for an answer, continue on the back of the page, or on the blank pages at the end of this booklet, **but please tell us where to look.** Alternatively, feel free to tear out the blank pages and use them as scratch paper.
  - **Please return your cheat sheets and all scratch paper with your answer booklet.**
  - If you use a greedy algorithm, you must prove that it is correct to receive credit. **Otherwise, proofs are required only if we specifically ask for them.**
  - As usual, answering any (sub)problem with “I don’t know” (and nothing else) is worth 25% partial credit. **Yes, even for problem 1.** Correct, complete, but suboptimal solutions are *always* worth more than 25%. A blank answer is not the same as “I don’t know”.
  - **Good luck!** And have a great winter break!
-

1. For each of the following questions, indicate *every* correct answer by marking the “Yes” box, and indicate *every* incorrect answer by marking the “No” box. **Assume P  $\neq$  NP**. If there is any other ambiguity or uncertainty, mark the “No” box. For example:

<input checked="" type="checkbox"/> Yes	No
Yes	<input checked="" type="checkbox"/> No
Yes	<input checked="" type="checkbox"/> No
<input checked="" type="checkbox"/> Yes	No

 $2 + 2 = 4$  $x + y = 5$ 

3SAT can be solved in polynomial time.

Jeff is not the Queen of England.

There are 40 yes/no choices altogether, each worth  $\frac{1}{2}$  point.

---

- (a) Which of the following statements is true for *every* language  $L \subseteq \{\text{0, 1}\}^*$ ?

Yes	No

 $L$  is non-empty. $L$  is decidable or  $L$  is infinite (or both). $L$  is accepted by some DFA with 42 states if and only if  $L$  is accepted by some NFA with 42 states.If  $L$  is regular, then  $L \in \text{NP}$ . $L$  is decidable if and only if its complement  $\overline{L}$  is undecidable.

- 
- (b) Which of the following computational models can be simulated by a deterministic Turing machine with three read/write heads, with at most polynomial slow-down in time, assuming  $P \neq NP$ ?

Yes	No

A Java program

A deterministic Turing machine with one head

A deterministic Turing machine with 3 tapes, each with 5 heads

A nondeterministic Turing machine with one head

A nondeterministic finite-state automaton (NFA)

- (c) Which of the following languages are decidable?

<input type="checkbox"/>	<input type="checkbox"/>	$\emptyset$
<input type="checkbox"/>	<input type="checkbox"/>	$\{0^n 1^n 0^n 1^n \mid n \geq 0\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M \rangle \mid M \text{ is a Turing machine}\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M \rangle \mid M \text{ accepts } \langle M \rangle \bullet \langle M \rangle\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M \rangle \mid M \text{ accepts a finite number of non-palindromes}\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M \rangle \mid M \text{ accepts } \emptyset\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M, w \rangle \mid M \text{ accepts } w^R\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M, w \rangle \mid M \text{ accepts } w \text{ after at most }  w ^2 \text{ transitions}\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M, w \rangle \mid M \text{ changes a blank on the tape to a non-blank, given input } w\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M, w \rangle \mid M \text{ changes a non-blank on the tape to a blank, given input } w\}$

- (d) Let  $M$  be a standard Turing machine (with a single one-track tape and a single head) that *decides* the regular language  $0^*1^*$ . Which of the following *must* be true?

<input type="checkbox"/>	<input type="checkbox"/>	Given an empty initial tape, $M$ eventually halts.
<input type="checkbox"/>	<input type="checkbox"/>	$M$ accepts the string $1111$ .
<input type="checkbox"/>	<input type="checkbox"/>	$M$ rejects the string $0110$ .
<input type="checkbox"/>	<input type="checkbox"/>	$M$ moves its head to the right at least once, given input $1100$ .
<input type="checkbox"/>	<input type="checkbox"/>	$M$ moves its head to the right at least once, given input $0101$ .
<input type="checkbox"/>	<input type="checkbox"/>	$M$ never accepts before reading a blank.
<input type="checkbox"/>	<input type="checkbox"/>	For some input string, $M$ moves its head to the left at least once.
<input type="checkbox"/>	<input type="checkbox"/>	For some input string, $M$ changes at least one symbol on the tape.
<input type="checkbox"/>	<input type="checkbox"/>	$M$ always halts.
<input type="checkbox"/>	<input type="checkbox"/>	If $M$ accepts a string $w$ , it does so after at most $O( w ^2)$ steps.

(e) Consider the following pair of languages:

- HAMILTONIANPATH :=  $\{G \mid G \text{ contains a Hamiltonian path}\}$
- CONNECTED :=  $\{G \mid G \text{ is connected}\}$

Which of the following **must** be true, assuming P  $\neq$  NP?

<input type="checkbox"/> Yes	<input type="checkbox"/> No	CONNECTED $\in$ NP
<input type="checkbox"/> Yes	<input type="checkbox"/> No	HAMILTONIANPATH $\in$ NP
<input type="checkbox"/> Yes	<input type="checkbox"/> No	HAMILTONIANPATH is decidable.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	There is no polynomial-time reduction from HAMILTONIANPATH to CONNECTED.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	There is no polynomial-time reduction from CONNECTED to HAMILTONIANPATH.

(f) Suppose we want to prove that the following language is undecidable.

$$\text{ALWAYSHALTS} := \{\langle M \rangle \mid M \text{ halts on every input string}\}$$

Rocket J. Squirrel suggests a reduction from the standard halting language

$$\text{HALT} := \{\langle M, w \rangle \mid M \text{ halts on inputs } w\}.$$

Specifically, suppose there is a Turing machine  $AH$  that decides ALWAYSHALTS. Rocky claims that the following Turing machine  $H$  decides HALT. Given an arbitrary encoding  $\langle M, w \rangle$  as input, machine  $H$  writes the encoding  $\langle M' \rangle$  of a new Turing machine  $M'$  to the tape and passes it to  $AH$ , where  $M'$  implements the following algorithm:

<u><math>M'(x):</math></u>
if $M$ accepts $w$
reject
if $M$ rejects $w$
accept

Which of the following statements is true for all inputs  $\langle M, w \rangle$ ?

<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $M$ accepts $w$ , then $M'$ halts on every input string.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $M$ diverges on $w$ , then $M'$ halts on every input string.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $M$ accepts $w$ , then $AH$ accepts $\langle M' \rangle$ .
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $M$ rejects $w$ , then $H$ rejects $\langle M, w \rangle$ .
<input type="checkbox"/> Yes	<input type="checkbox"/> No	$H$ decides the language HALT. (That is, Rocky's reduction is correct.)

2. A *relaxed 3-coloring* of a graph  $G$  assigns each vertex of  $G$  one of three colors (for example, red, green, and blue), such that *at most one* edge in  $G$  has both endpoints the same color.
  - (a) Give an example of a graph that has a relaxed 3-coloring, but does not have a proper 3-coloring (where every edge has endpoints of different colors).
  - (b) *Prove* that it is NP-hard to determine whether a given graph has a relaxed 3-coloring.

3. Give a complete, formal, self-contained description of a DFA that accepts all strings in  $\{0, 1\}^*$  containing at least ten 0s and at most ten 1s. Specifically:

- (a) What are the states of your DFA?
- (b) What is the start state of your DFA?
- (c) What are the accepting states of your DFA?
- (d) What is your DFA's transition function?

4. Suppose you are given three strings  $A[1..n]$ ,  $B[1..n]$ , and  $C[1..n]$ . Describe and analyze an algorithm to find the maximum length of a common subsequence of all three strings. For example, given the input strings

$$A = \text{AxxBxxCDxEF}, \quad B = \text{yyABCDyEyFy}, \quad C = \text{zAzzBCDzEFz},$$

your algorithm should output the number 6, which is the length of the longest common subsequence  $\text{ABCDEF}$ .

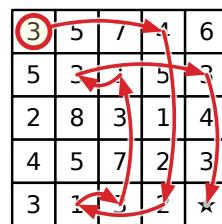
5. For each of the following languages over the alphabet  $\Sigma = \{0, 1\}$ , either *prove* that the language is regular, or *prove* that the language is not regular.

- (a)  $\{www \mid w \in \Sigma^*\}$
- (b)  $\{wxw \mid w, x \in \Sigma^*\}$

6. A **number maze** is an  $n \times n$  grid of positive integers. A token starts in the upper left corner; your goal is to move the token to the lower-right corner. On each turn, you are allowed to move the token up, down, left, or right; the distance you may move the token is determined by the number on its current square. For example, if the token is on a square labeled 3, then you may move the token three steps up, three steps down, three steps left, or three steps right. However, you are never allowed to move the token off the edge of the board.

Describe and analyze an efficient algorithm that either returns the minimum number of moves required to solve a given number maze, or correctly reports that the maze has no solution. For example, given the maze shown below, your algorithm would return the number 8.

3	5	7	4	6
5	3	1	5	3
2	8	3	1	4
4	5	7	2	3
3	1	3	2	★



A  $5 \times 5$  number maze that can be solved in eight moves.

(scratch paper)

(scratch paper)

**You may assume the following problems are NP-hard:**

**CIRCUITSAT:** Given a boolean circuit, are there any input values that make the circuit output TRUE?

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

**MAXINDEPENDENTSET:** Given an undirected graph  $G$ , what is the size of the largest subset of vertices in  $G$  that have no edges among them?

**MAXCLIQUE:** Given an undirected graph  $G$ , what is the size of the largest complete subgraph of  $G$ ?

**MINVERTEXCOVER:** Given an undirected graph  $G$ , what is the size of the smallest subset of vertices that touch every edge in  $G$ ?

**3COLOR:** Given an undirected graph  $G$ , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

**HAMILTONIANPATH:** Given an undirected graph  $G$ , is there a path in  $G$  that visits every vertex exactly once?

**HAMILTONIANCYCLE:** Given an undirected graph  $G$ , is there a cycle in  $G$  that visits every vertex exactly once?

**DIRECTEDHAMILTONIANCYCLE:** Given an directed graph  $G$ , is there a directed cycle in  $G$  that visits every vertex exactly once?

**TRAVELINGSALESMAN:** Given a graph  $G$  (either directed or undirected) with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in  $G$ ?

**DRAUGHTS:** Given an  $n \times n$  international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

**SUPER MARIO:** Given an  $n \times n$  level for Super Mario Brothers, can Mario reach the castle?

**You may assume the following languages are undecidable:**

SELFREJECT :=  $\{\langle M \rangle \mid M \text{ rejects } \langle M \rangle\}$

SELFACCEPT :=  $\{\langle M \rangle \mid M \text{ accepts } \langle M \rangle\}$

SELFHALT :=  $\{\langle M \rangle \mid M \text{ halts on } \langle M \rangle\}$

SELFDIVERGE :=  $\{\langle M \rangle \mid M \text{ does not halt on } \langle M \rangle\}$

REJECT :=  $\{\langle M, w \rangle \mid M \text{ rejects } w\}$

ACCEPT :=  $\{\langle M, w \rangle \mid M \text{ accepts } w\}$

HALT :=  $\{\langle M, w \rangle \mid M \text{ halts on } w\}$

DIVERGE :=  $\{\langle M, w \rangle \mid M \text{ does not halt on } w\}$

NEVERREJECT :=  $\{\langle M \rangle \mid \text{REJECT}(M) = \emptyset\}$

NEVERACCEPT :=  $\{\langle M \rangle \mid \text{ACCEPT}(M) = \emptyset\}$

NEVERHALT :=  $\{\langle M \rangle \mid \text{HALT}(M) = \emptyset\}$

NEVERDIVERGE :=  $\{\langle M \rangle \mid \text{DIVERGE}(M) = \emptyset\}$

**“CS 374”: Algorithms and Models of Computation, Fall 2014**  
**Final Exam (Version B) — December 16, 2014**

Name:			
NetID:			
Section:	1	2	3

#	1	2	3	4	5	6	Total
Score							
Max	20	10	10	10	10	10	70
Grader							

- 
- ***Don’t panic!***
  - Please print your name and your NetID and circle your discussion section in the boxes above.
  - This is a closed-book, closed-notes, closed-electronics exam. If you brought anything except your writing implements and your two double-sided 8½" × 11" cheat sheets, please put it away for the duration of the exam. In particular, you may not use *any* electronic devices.
  - **Please read the entire exam before writing anything.** Please ask for clarification if any question is unclear.
  - **You have 180 minutes.**
  - If you run out of space for an answer, continue on the back of the page, or on the blank pages at the end of this booklet, **but please tell us where to look.** Alternatively, feel free to tear out the blank pages and use them as scratch paper.
  - **Please return your cheat sheets and all scratch paper with your answer booklet.**
  - If you use a greedy algorithm, you must prove that it is correct to receive credit. **Otherwise, proofs are required only if we specifically ask for them.**
  - As usual, answering any (sub)problem with “I don’t know” (and nothing else) is worth 25% partial credit. **Yes, even for problem 1.** Correct, complete, but suboptimal solutions are *always* worth more than 25%. A blank answer is not the same as “I don’t know”.
  - **Good luck!** And have a great winter break!
-

1. For each of the following questions, indicate *every* correct answer by marking the “Yes” box, and indicate *every* incorrect answer by marking the “No” box. **Assume P  $\neq$  NP**. If there is any other ambiguity or uncertainty, mark the “No” box. For example:

<input checked="" type="checkbox"/> Yes	No	$2 + 2 = 4$
Yes	<input checked="" type="checkbox"/> No	$x + y = 5$
Yes	<input checked="" type="checkbox"/> No	3SAT can be solved in polynomial time.
<input checked="" type="checkbox"/> Yes	No	Jeff is not the Queen of England.

There are 40 yes/no choices altogether, each worth  $\frac{1}{2}$  point.

- (a) Which of the following statements is true for *every* language  $L \subseteq \{0, 1\}^*$ ?

<input type="checkbox"/> Yes	<input type="checkbox"/> No	$L$ is non-empty.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	$L$ is decidable or $L$ is infinite (or both).
<input type="checkbox"/> Yes	<input type="checkbox"/> No	$L$ is accepted by some DFA with 42 states if and only if $L$ is accepted by some NFA with 42 states.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $L$ is regular, then $L \in \text{NP}$ .
<input type="checkbox"/> Yes	<input type="checkbox"/> No	$L$ is decidable if and only if its complement $\overline{L}$ is undecidable.

- (b) Which of the following computational models can simulate a deterministic Turing machine with three read/write heads, with at most polynomial slow-down in time, assuming P  $\neq$  NP?

<input type="checkbox"/> Yes	<input type="checkbox"/> No	A C++ program
<input type="checkbox"/> Yes	<input type="checkbox"/> No	A deterministic Turing machine with one head
<input type="checkbox"/> Yes	<input type="checkbox"/> No	A deterministic Turing machine with 3 tapes, each with 5 heads
<input type="checkbox"/> Yes	<input type="checkbox"/> No	A nondeterministic Turing machine with one head
<input type="checkbox"/> Yes	<input type="checkbox"/> No	A nondeterministic finite-state automaton (NFA)

- (c) Which of the following languages are decidable?

<input type="checkbox"/>	<input type="checkbox"/>	$\emptyset$
<input type="checkbox"/>	<input type="checkbox"/>	$\{ww \mid w \text{ is a palindrome}\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M \rangle \mid M \text{ is a Turing machine}\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M \rangle \mid M \text{ accepts } \langle M \rangle \bullet \langle M \rangle\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M \rangle \mid M \text{ accepts an infinite number of palindromes}\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M \rangle \mid M \text{ accepts } \emptyset\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M, w \rangle \mid M \text{ accepts } www\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M, w \rangle \mid M \text{ accepts } w \text{ after at least }  w ^2 \text{ transitions}\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M, w \rangle \mid M \text{ changes a non-blank on the tape to a blank, given input } w\}$
<input type="checkbox"/>	<input type="checkbox"/>	$\{\langle M, w \rangle \mid M \text{ changes a blank on the tape to a non-blank, given input } w\}$

- (d) Let  $M$  be a standard Turing machine (with a single one-track tape and a single head) such that  $\text{ACCEPT}(M)$  is the regular language  $0^*1^*$ . Which of the following *must* be true?

<input type="checkbox"/>	<input type="checkbox"/>	Given an empty initial tape, $M$ eventually halts.
<input type="checkbox"/>	<input type="checkbox"/>	$M$ accepts the string $1111$ .
<input type="checkbox"/>	<input type="checkbox"/>	$M$ rejects the string $0110$ .
<input type="checkbox"/>	<input type="checkbox"/>	$M$ moves its head to the right at least once, given input $1100$ .
<input type="checkbox"/>	<input type="checkbox"/>	$M$ moves its head to the right at least once, given input $0101$ .
<input type="checkbox"/>	<input type="checkbox"/>	$M$ must read a blank before it accepts.
<input type="checkbox"/>	<input type="checkbox"/>	For some input string, $M$ moves its head to the left at least once.
<input type="checkbox"/>	<input type="checkbox"/>	For some input string, $M$ changes at least one symbol on the tape.
<input type="checkbox"/>	<input type="checkbox"/>	$M$ always halts.
<input type="checkbox"/>	<input type="checkbox"/>	If $M$ accepts a string $w$ , it does so after at most $O( w ^2)$ steps.

(e) Consider the following pair of languages:

- HAMILTONIANPATH :=  $\{G \mid G \text{ contains a Hamiltonian path}\}$
- CONNECTED :=  $\{G \mid G \text{ is connected}\}$

Which of the following *must* be true, assuming P  $\neq$  NP?

- |     |    |
|-----|----|
| Yes | No |
|-----|----|
- $\text{CONNECTED} \in \text{NP}$
- 
- |     |    |
|-----|----|
| Yes | No |
|-----|----|
- $\text{HAMILTONIANPATH} \in \text{NP}$
- 
- |     |    |
|-----|----|
| Yes | No |
|-----|----|
- $\text{HAMILTONIANPATH}$  is undecidable.
- 
- |     |    |
|-----|----|
| Yes | No |
|-----|----|
- There is a polynomial-time reduction from  $\text{HAMILTONIANPATH}$  to  $\text{CONNECTED}$ .
- 
- |     |    |
|-----|----|
| Yes | No |
|-----|----|
- There is a polynomial-time reduction from  $\text{CONNECTED}$  to  $\text{HAMILTONIANPATH}$ .
- 

(f) Suppose we want to prove that the following language is undecidable.

$$\text{ALWAYSHALTS} := \{\langle M \rangle \mid M \text{ halts on every input string}\}$$

Bullwinkle J. Moose suggests a reduction from the standard halting language

$$\text{HALT} := \{\langle M, w \rangle \mid M \text{ halts on inputs } w\}.$$

Specifically, suppose there is a Turing machine  $AH$  that decides  $\text{ALWAYSHALTS}$ . Bullwinkle claims that the following Turing machine  $H$  decides  $\text{HALT}$ . Given an arbitrary encoding  $\langle M, w \rangle$  as input, machine  $H$  writes the encoding  $\langle M' \rangle$  of a new Turing machine  $M'$  to the tape and passes it to  $AH$ , where  $M'$  implements the following algorithm:

$M'(x)$ :

if  $M$  accepts  $w$   
    reject  
if  $M$  rejects  $w$   
    accept

Which of the following statements is true for all inputs  $\langle M, w \rangle$ ?

- |     |    |
|-----|----|
| Yes | No |
|-----|----|
- If  $M$  accepts  $w$ , then  $M'$  halts on every input string.
- 
- |     |    |
|-----|----|
| Yes | No |
|-----|----|
- If  $M$  rejects  $w$ , then  $M'$  halts on every input string.
- 
- |     |    |
|-----|----|
| Yes | No |
|-----|----|
- If  $M$  rejects  $w$ , then  $H$  rejects  $\langle M, w \rangle$ .
- 
- |     |    |
|-----|----|
| Yes | No |
|-----|----|
- If  $M$  diverges on  $w$ , then  $H$  diverges on  $\langle M, w \rangle$ .
- 
- |     |    |
|-----|----|
| Yes | No |
|-----|----|
- $H$  does not correctly decide the language  $\text{HALT}$ . (That is, Bullwinkle's reduction is incorrect.)
-

2. A *near-Hamiltonian cycle* in a graph  $G$  is a closed walk in  $G$  that visits one vertex exactly twice and every other vertex exactly once.
- Give an example of a graph that contains a near-Hamiltonian cycle, but does not contain a Hamiltonian cycle (which visits every vertex exactly once).
  - Prove* that it is NP-hard to determine whether a given graph contains a near-Hamiltonian cycle.

3. Give a complete, formal, self-contained description of a DFA that accepts all strings in  $\{0, 1\}^*$  such that every fifth bit is 0 and the length is *not* divisible by 12. For example, your DFA should accept the strings **11110111101** and **11**. Specifically:

- (a) What are the states of your DFA?
- (b) What is the start state of your DFA?
- (c) What are the accepting states of your DFA?
- (d) What is your DFA's transition function?

4. Suppose you are given three strings  $A[1..n]$ ,  $B[1..n]$ , and  $C[1..n]$ . Describe and analyze an algorithm to find the maximum length of a common subsequence of all three strings. For example, given the input strings

$$A = \text{AxxBxxCDxEF}, \quad B = \text{yyABCDyEyFy}, \quad C = \text{zAzzBCDzEFz},$$

your algorithm should output the number 6, which is the length of the longest common subsequence  $\text{ABCDEF}$ .

5. For each of the following languages over the alphabet  $\Sigma = \{0, 1\}$ , either *prove* that the language is regular, or *prove* that the language is not regular.

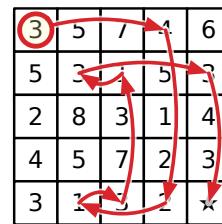
- (a)  $\{www \mid w \in \Sigma^*\}$
- (b)  $\{wxw \mid w, x \in \Sigma^*\}$

6. A **number maze** is an  $n \times n$  grid of positive integers. A token starts in the upper left corner; your goal is to move the token to the lower-right corner.

- On each turn, you are allowed to move the token up, down, left, or right.
- The distance you may move the token is determined by the number on its current square. For example, if the token is on a square labeled 3, then you may move the token three steps up, three steps down, three steps left, or three steps right.
- However, you are never allowed to move the token off the edge of the board. In particular, if the current number is too large, you may not be able to move at all.

Describe and analyze an efficient algorithm that either returns the minimum number of moves required to solve a given number maze, or correctly reports that the maze has no solution. For example, given the maze shown below, your algorithm would return the number 8.

3	5	7	4	6
5	3	1	5	3
2	8	3	1	4
4	5	7	2	3
3	1	3	2	★



A  $5 \times 5$  number maze that can be solved in eight moves.

(scratch paper)

(scratch paper)

**You may assume the following problems are NP-hard:**

**CIRCUITSAT:** Given a boolean circuit, are there any input values that make the circuit output TRUE?

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

**MAXINDEPENDENTSET:** Given an undirected graph  $G$ , what is the size of the largest subset of vertices in  $G$  that have no edges among them?

**MAXCLIQUE:** Given an undirected graph  $G$ , what is the size of the largest complete subgraph of  $G$ ?

**MINVERTEXCOVER:** Given an undirected graph  $G$ , what is the size of the smallest subset of vertices that touch every edge in  $G$ ?

**3COLOR:** Given an undirected graph  $G$ , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

**HAMILTONIANPATH:** Given an undirected graph  $G$ , is there a path in  $G$  that visits every vertex exactly once?

**HAMILTONIANCYCLE:** Given an undirected graph  $G$ , is there a cycle in  $G$  that visits every vertex exactly once?

**DIRECTEDHAMILTONIANCYCLE:** Given an directed graph  $G$ , is there a directed cycle in  $G$  that visits every vertex exactly once?

**TRAVELINGSALESMAN:** Given a graph  $G$  (either directed or undirected) with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in  $G$ ?

**DRAUGHTS:** Given an  $n \times n$  international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

**SUPER MARIO:** Given an  $n \times n$  level for Super Mario Brothers, can Mario reach the castle?

**You may assume the following languages are undecidable:**

SELFREJECT :=  $\{\langle M \rangle \mid M \text{ rejects } \langle M \rangle\}$

SELFACCEPT :=  $\{\langle M \rangle \mid M \text{ accepts } \langle M \rangle\}$

SELFHALT :=  $\{\langle M \rangle \mid M \text{ halts on } \langle M \rangle\}$

SELFDIVERGE :=  $\{\langle M \rangle \mid M \text{ does not halt on } \langle M \rangle\}$

REJECT :=  $\{\langle M, w \rangle \mid M \text{ rejects } w\}$

ACCEPT :=  $\{\langle M, w \rangle \mid M \text{ accepts } w\}$

HALT :=  $\{\langle M, w \rangle \mid M \text{ halts on } w\}$

DIVERGE :=  $\{\langle M, w \rangle \mid M \text{ does not halt on } w\}$

NEVERREJECT :=  $\{\langle M \rangle \mid \text{REJECT}(M) = \emptyset\}$

NEVERACCEPT :=  $\{\langle M \rangle \mid \text{ACCEPT}(M) = \emptyset\}$

NEVERHALT :=  $\{\langle M \rangle \mid \text{HALT}(M) = \emptyset\}$

NEVERDIVERGE :=  $\{\langle M \rangle \mid \text{DIVERGE}(M) = \emptyset\}$

# ∞ New CS 473: Algorithms, Spring 2015 ∞

## Homework 0

Due Tuesday, January 27, 2015 at 5pm

---

- This homework tests your familiarity with **prerequisite material**: designing, describing, and analyzing elementary algorithms (at the level of CS 225); fundamental graph problems and algorithms (again, at the level of CS 225); and especially facility with recursion and induction. Notes on most of this prerequisite material are available on the course web page.
  - **Each student must submit individual solutions for this homework.** For all future homeworks, groups of up to three students will be allowed to submit joint solutions.
- 

### ☞ Some important course policies ☝

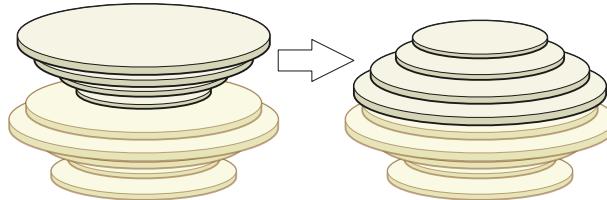
- **You may use any source at your disposal**—paper, electronic, or human—but you **must** cite **every** source that you use, and you **must** write everything yourself in your own words. See the academic integrity policies on the course web site for more details.
  - **Submit your solutions on standard printer/copier paper.** Please use both sides of the paper. Please clearly print your name and NetID at the top of each page. If you plan to write your solutions by hand, please print the last four pages of this homework as templates. If you plan to typeset your homework, you can find a `LATEX` template on the course web site; well-typeset homework will get a small amount of extra credit.
  - **Start your solution to each numbered problem on a new sheet of paper.** Do not staple your entire homework together.
  - **Submit your solutions in the drop boxes outside 1404 Siebel labeled “New CS 473”.** There is a separate drop box for each numbered problem; if you put your solution in the wrong drop box, we won’t grade it. Don’t give your homework to Jeff in class; he is fond of losing important pieces of paper.
  - **Avoid the Three Deadly Sins!** There are a few dangerous writing (and thinking) habits that will trigger an automatic zero on any homework or exam problem, unless your solution is nearly perfect otherwise. Yes, we are completely serious.
    - Always give complete solutions, not just examples.
    - Always declare all your variables.
    - Never use weak induction.
  - Answering any homework or exam problem (or subproblem) in this course with “I don’t know” *and nothing else* is worth 25% partial credit. We will accept synonyms like “No idea” or “WTF”, but you must write *something*.
- 

**See the course web site for more information.**

If you have any questions about these policies,  
please don’t hesitate to ask in class, in office hours, or on Piazza.

---

1. Suppose you are given a stack of  $n$  pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a *flip*—insert a spatula under the top  $k$  pancakes, for some integer  $k$  between 1 and  $n$ , and flip them all over.



Flipping the top four pancakes.

- (a) Describe an algorithm to sort an arbitrary stack of  $n$  pancakes, which uses as few flips as possible in the worst case. *Exactly* how many flips does your algorithm perform in the worst case?
- (b) Now suppose one side of each pancake is burned. Describe an algorithm to sort an arbitrary stack of  $n$  pancakes, so that the burned side of every pancake is facing down, using as few flips as possible in the worst case. *Exactly* how many flips does your algorithm perform in the worst case?

2. [From last semester's CS 374 final exam] A **number maze** is an  $n \times n$  grid of positive integers. A token starts in the upper left corner; your goal is to move the token to the lower-right corner.

- On each turn, you are allowed to move the token up, down, left, or right.
- The distance you may move the token is determined by the number on its current square. For example, if the token is on a square labeled 3, then you may move the token three steps up, three steps down, three steps left, or three steps right.
- However, you are never allowed to move the token off the edge of the board. In particular, if the current number is too large, you may not be able to move at all.

Describe and analyze an efficient algorithm that either returns the minimum number of moves required to solve a given number maze, or correctly reports that the maze has no solution. For example, given the maze shown below, your algorithm would return the number 8.

3	5	7	4	6
5	3	1	5	3
2	8	3	1	4
4	5	7	2	3
3	1	3	2	★

3	5	7	4	6
5	3	1	5	3
2	8	3	1	4
4	5	7	2	3
3	1	3	2	★

A  $5 \times 5$  number maze that can be solved in eight moves.

[Hint: Build a graph. What are the vertices? What are the edges? Is the graph directed or undirected? Do the vertices or edges have weights? If so, what are they? What textbook problem do you need to solve on this graph? What textbook algorithm should you use to solve that problem? What is the running time of that algorithm as a function of  $n$ ?]

3. (a) The **Fibonacci numbers**  $F_n$  are defined by the recurrence  $F_n = F_{n-1} + F_{n-2}$ , with base cases  $F_0 = 0$  and  $F_1 = 1$ . Here are the first several Fibonacci numbers:

$F_0$	$F_1$	$F_2$	$F_3$	$F_4$	$F_5$	$F_6$	$F_7$	$F_8$	$F_9$	$F_{10}$
0	1	1	2	3	5	8	13	21	34	55

Prove that every non-negative integer can be written as the sum of distinct, non-consecutive Fibonacci numbers. That is, if the Fibonacci number  $F_i$  appears in the sum, it appears exactly once, and its neighbors  $F_{i-1}$  and  $F_{i+1}$  do not appear at all. For example:

$$17 = F_7 + F_4 + F_2, \quad 42 = F_9 + F_6, \quad 54 = F_9 + F_7 + F_5 + F_3 + F_1.$$

- (b) The Fibonacci sequence can be extended backward to negative indices by rearranging the defining recurrence:  $F_n = F_{n+2} - F_{n+1}$ . Here are the first several negative-index Fibonacci numbers:

$F_{-10}$	$F_{-9}$	$F_{-8}$	$F_{-7}$	$F_{-6}$	$F_{-5}$	$F_{-4}$	$F_{-3}$	$F_{-2}$	$F_{-1}$
-55	34	-21	13	-8	5	-3	2	-1	1

Prove that  $F_{-n} = -F_n$  if and only if  $n$  is even.

- (c) Prove that every integer—positive, negative, or zero—can be written as the sum of distinct, non-consecutive Fibonacci numbers with negative indices. For example:

$$17 = F_{-7} + F_{-5} + F_{-2}, \quad -42 = F_{-10} + F_{-7}, \quad 54 = F_{-9} + F_{-7} + F_{-5} + F_{-3} + F_{-1}.$$

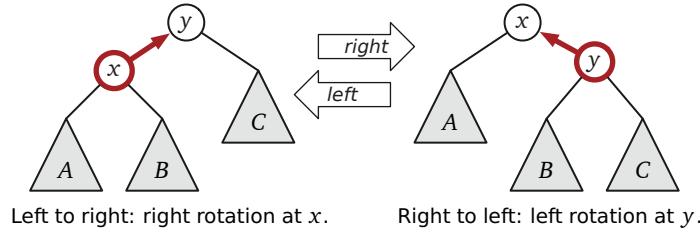
[Hint: Zero is both non-negative and even. Don't even think about using weak induction!]

4. [Extra credit] Let  $T$  be a binary tree whose nodes store distinct numerical values. Recall that  $T$  is a **binary search tree** if and only if either (1)  $T$  is empty, or (2)  $T$  satisfies the following recursive conditions:

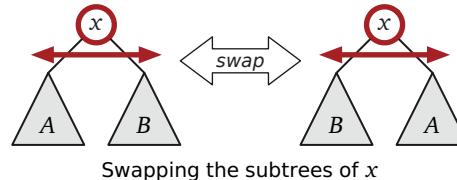
- The left subtree of  $T$  is a binary search tree.
- All values in the left subtree of  $T$  are smaller than the value at the root of  $T$ .
- The right subtree of  $T$  is a binary search tree.
- All values in the right subtree of  $T$  are larger than the value at the root of  $T$ .

Describe and analyze an algorithm to transform an *arbitrary* binary tree  $T$  with distinct node values into a binary search tree, using **only** the following operations:

- Rotate an arbitrary node. Rotation is a local operation that decreases the depth of a node by one and increases the depth of its parent by one.

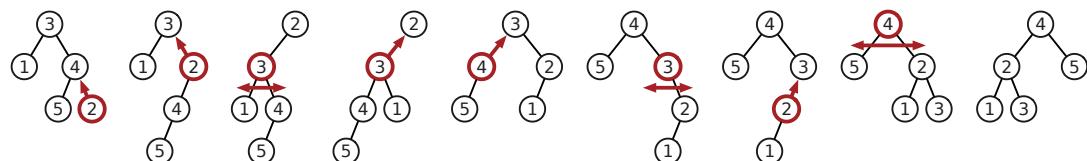


- Swap the left and right subtrees of an arbitrary node.



For both of these operations, some, all, or none of the subtrees  $A$ ,  $B$ , and  $C$  may be empty.

The following example shows a five-node binary tree transforming into a binary search tree in eight operations:



"Sorting" a binary tree in eight steps: rotate 2, rotate 2, swap 3, rotate 3, rotate 4, swap 3, rotate 2, swap 4.

Your algorithm cannot directly modify parent or child pointers, and it cannot allocate new nodes or delete old nodes; the **only** way it can modify  $T$  is using rotations and swaps. On the other hand, you may *compute* anything you like for free, as long as that computation does not modify  $T$ . In other words, the running time of your algorithm is *defined* to be the number of rotations and swaps that it performs.

For full credit, your algorithm should use as few rotations and swaps as possible in the worst case. [Hint:  $O(n^2)$  operations is not too difficult, but we can do better.]

# ∞ New CS 473: Algorithms, Spring 2015 ∞

## Homework 1

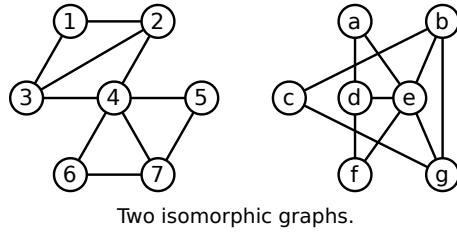
Due Tuesday, February 3, 2015 at 5pm

---

For this and all future homeworks, groups of up to three students can submit joint solutions. Please print (or typeset) the name and NetID of every group member on the first page of each submission.

---

1. Two graphs are **isomorphic** if one can be transformed into the other just by relabeling the vertices. For example, the graphs shown below are isomorphic; the left graph can be transformed into the right graph by the relabeling  $(1, 2, 3, 4, 5, 6, 7) \mapsto (c, g, b, e, a, f, d)$ .



Consider the following related decision problems:

- **GRAPHISOMORPHISM:** Given two graphs  $G$  and  $H$ , determine whether  $G$  and  $H$  are isomorphic.
  - **EVENGRAPHISOMORPHISM:** Given two graphs  $G$  and  $H$ , such that every vertex in  $G$  and every vertex in  $H$  has even degree, determine whether  $G$  and  $H$  are isomorphic.
  - **SUBGRAPHISOMORPHISM:** Given two graphs  $G$  and  $H$ , determine whether  $G$  is isomorphic to a subgraph of  $H$ .
- (a) Describe a polynomial-time reduction from **EVENGRAPHISOMORPHISM** to **GRAPHISOMORPHISM**.
  - (b) Describe a polynomial-time reduction from **GRAPHISOMORPHISM** to **EVENGRAPHISOMORPHISM**.
  - (c) Describe a polynomial-time reduction from **GRAPHISOMORPHISM** to **SUBGRAPHISOMORPHISM**.
  - (d) Prove that **SUBGRAPHISOMORPHISM** is NP-complete.
  - (e) What can you conclude about the NP-hardness of **GRAPHISOMORPHISM**? Justify your answer.

[Hint: These are all easy!]

2. Prove that the following problems are NP-hard.
  - (a) Given an undirected graph  $G$ , does  $G$  have a spanning tree with at most 473 leaves?
  - (b) Given an undirected graph  $G = (V, E)$ , what is the size of the largest subset of vertices  $S \subseteq V$  such that at most 2015 edges in  $E$  have both endpoints in  $S$ ?
3. The Hamiltonian cycle problem has two closely related variants:
  - **UNDIRECTEDHAMCYCLE:** Given an *undirected* graph  $G$ , does  $G$  contain an *undirected* Hamiltonian cycle?
  - **DIRECTEDHAMCYCLE:** Given an *directed* graph  $G$ , does  $G$  contain a *directed* Hamiltonian cycle?

This question asks you to prove that these two problems are essentially equivalent.

- (a) Describe a polynomial-time reduction from **UNDIRECTEDHAMCYCLE** to **DIRECTEDHAMCYCLE**.
  - (b) Describe a polynomial-time reduction from **DIRECTEDHAMCYCLE** to **UNDIRECTEDHAMCYCLE**.
- \*4. **[Extra Credit]** Describe a direct polynomial-time reduction from **4COLOR** to **3COLOR**. (This is a lot harder than the opposite direction!)

# ∞ New CS 473: Algorithms, Spring 2015 ∞

## Homework 2

Due Tuesday, February 10, 2015 at 5pm

---

1. The **maximum  $k$ -cut problem** asks, given a graph  $G$  with edge weights and an integer  $k$  as input, to compute a partition of the vertices of  $G$  into  $k$  disjoint subsets  $S_1, S_2, \dots, S_k$  such that the sum of the weights of the edges that cross the partition (that is, have endpoints in different subsets) is as large as possible.
  - (a) Describe an efficient  $(1 - 1/k)$ -approximation algorithm for this problem.
  - (b) Now suppose we wish to minimize the sum of the weights of edges that do *not* cross the partition. What approximation ratio does your algorithm from part (a) achieve for the new problem? Justify your answer.
2. In the **bin packing** problem, we are given a set of  $n$  items, each with weight between 0 and 1, and we are asked to load the items into as few bins as possible, such that the total weight in each bin is at most 1. It's not hard to show that this problem is NP-Hard; this question asks you to analyze a few common approximation algorithms. In each case, the input is an array  $W[1..n]$  of weights, and the output is the number of bins used.

```
NEXTFIT( $W[1..n]$ ):
  bins ← 0
  Total[0] ← ∞
  for  $i \leftarrow 1$  to  $n$ 
    if  $Total[bins] + W[i] > 1$ 
      bins ← bins + 1
      Total[bins] ←  $W[i]$ 
    else
      Total[bins] ←  $Total[bins] + W[i]$ 
  return bins
```

```
FIRSTFIT( $W[1..n]$ ):
  bins ← 0
  for  $i \leftarrow 1$  to  $n$ 
     $j \leftarrow 1$ ; found ← FALSE
    while  $j \leq bins$  and  $\neg found$ 
      if  $Total[j] + W[i] \leq 1$ 
        Total[j] ←  $Total[j] + W[i]$ 
        found ← TRUE
       $j \leftarrow j + 1$ 
    if  $\neg found$ 
      bins ← bins + 1
      Total[bins] =  $W[i]$ 
  return bins
```

- (a) Prove that NEXTFIT uses at most twice the optimal number of bins.
- (b) Prove that FIRSTFIT uses at most twice the optimal number of bins.
- \*(c) **[Extra Credit]** Prove that if the weight array  $W$  is initially sorted in decreasing order, then FIRSTFIT uses at most  $(4 \cdot OPT + 1)/3$  bins, where  $OPT$  is the optimal number of bins. The following facts may be useful (but you need to prove them if your proof uses them):
  - In the packing computed by FIRSTFIT, every item with weight more than  $1/3$  is placed in one of the first  $OPT$  bins.
  - FIRSTFIT places at most  $OPT - 1$  items outside the first  $OPT$  bins.

3. Consider the following greedy algorithm for the metric traveling salesman problem: Start at an arbitrary vertex, and then repeatedly travel to the closest unvisited vertex, until every vertex has been visited.
- (a) Prove that the approximation ratio for this algorithm is  $O(\log n)$ , where  $n$  is the number of vertices. [*Hint: Argue that the  $k$ th least expensive edge in the tour output by the greedy algorithm has weight at most  $\text{OPT}/(n - k + 1)$ ; try  $k = 1$  and  $k = 2$  first.*]
  - \*(b) **[Extra Credit]** Prove that the approximation ratio for this algorithm is  $\Omega(\log n)$ . That is, describe an infinite family of weighted graphs such that the greedy algorithm returns a Hamiltonian cycle whose weight is  $\Omega(\log n)$  times the weight of the optimal TSP tour.

# ∞ New CS 473: Algorithms, Spring 2015 ∞

## Homework 3

Due Tuesday, February 17, 2015 at 5pm

---

1. A string  $w$  of parentheses ( and ) and brackets [ and ] is *balanced* if it satisfies one of the following conditions:

- $w$  is the empty string.
- $w = (x)$  for some balanced string  $x$
- $w = [x]$  for some balanced string  $x$
- $w = xy$  for some balanced strings  $x$  and  $y$

For example, the string

$$w = (\textcolor{red}{[}\textcolor{green}{(}\textcolor{red}{)}\textcolor{green}{]}\textcolor{red}{[}\textcolor{green}{(}\textcolor{red}{)}\textcolor{green}{]}\textcolor{red}{[}\textcolor{green}{(}\textcolor{red}{)}\textcolor{green}{]}\textcolor{red}{)}$$

is balanced, because  $w = xy$ , where

$$x = \textcolor{red}{(\textcolor{green}{[}\textcolor{red}{)}\textcolor{green}{]}\textcolor{red}{[}\textcolor{green}{(}\textcolor{red}{)})} \quad \text{and} \quad y = \textcolor{red}{[\textcolor{green}{(}\textcolor{red}{)}\textcolor{green}{]}\textcolor{red}{[}\textcolor{green}{(}\textcolor{red}{)}\textcolor{green}{]}.$$

Describe and analyze an algorithm to compute the length of a longest balanced subsequence of a given string of parentheses and brackets. Your input is an array  $w[1..n]$ , where  $w[i] \in \{(, ), [, ]\}$  for every index  $i$ . (You may prefer to use different symbols instead of parentheses and brackets—for example, L, R, l, r or <, >, ▲, ▼—but please tell us what symbols you’re using!)

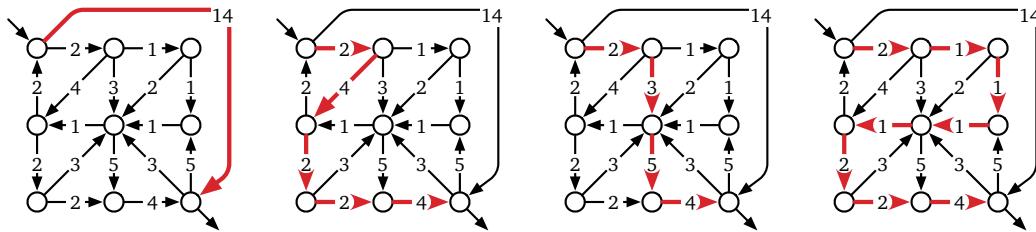
2. Congratulations! You’ve just been hired at internet giant Yeehaw! as the new party czar. The president of the company has asked you to plan the annual holiday party. Your task is to find exactly  $k$  employees to invite, including the president herself. Employees at Yeehaw! are organized into a strict hierarchy—a tree with the president at the root. The all-knowing oracles in Human Resources have determined two numerical values for every employee:

- $With[i]$  measures much fun employee  $i$  would have at the party if their immediate supervisor is also invited.
- $Without[i]$  measures how much fun employee  $i$  would have at the party if their immediate supervisor is *not* also invited.

These values could be positive, negative, or zero, and  $With[i]$  could be greater than, less than, or equal to  $Without[i]$ .

Describe an algorithm that finds the set of  $k$  employees to invite that maximizes the sum of the  $k$  resulting “fun” values. The input to your algorithm is the tree  $T$ , the integer  $k$ , and the  $With$  and  $Without$  values for each employee. Assume that everyone invited to the party actually attends. Do *not* assume that  $T$  is a *binary* tree.

3. Although we typically speak of “the” shortest path between two nodes, a single graph could contain several minimum-length paths with the same endpoints.



Four (of many) equal-length shortest paths.

Describe and analyze an algorithm to determine the *number* of shortest paths from a source vertex  $s$  to a target vertex  $t$  in an arbitrary directed graph  $G$  with weighted edges. You may assume that all edge weights are positive and that all necessary arithmetic operations can be performed in  $O(1)$  time.

*[Hint: Compute shortest path distances from  $s$  to every other vertex. Throw away all edges that cannot be part of a shortest path from  $s$  to another vertex. What's left?]*

# ∞ New CS 473: Algorithms, Spring 2015 ∞

## Homework 4

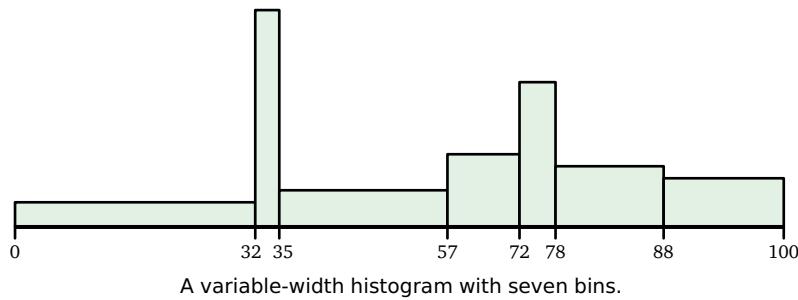
Due Tuesday, February 24, 2015 at 5pm

---

Starting with this assignment, all homework must be submitted electronically via Moodle as separate PDF files, one for each numbered problem. Please see the course web site for more information.

---

1. Suppose we want to summarize a large set  $S$  of values—for example, course averages for students in CS 105—using a variable-width histogram. To construct a histogram, we choose a sorted sequence of *breakpoints*  $b_0 < b_1 < \dots < b_k$ , such that every element of  $S$  lies between  $b_0$  and  $b_k$ . Then for each interval  $[b_{i-1}, b_i]$ , the histogram includes a rectangle whose height is the number of elements of  $S$  that lie inside that interval.



Unlike a standard histogram, which requires the intervals to have equal width, we are free to choose the breakpoints arbitrarily. For statistical purposes, it is useful for the *areas* of the rectangles to be as close to equal as possible. To that end, define the **cost** of a histogram to be the sum of the *squares* of the rectangle areas; we want to compute the histogram with minimum cost.

More formally, suppose we fix a sequence of breakpoints  $b_0 < b_1 < \dots < b_k$ . For each index  $i$ , let  $n_i$  denote the number of input values in the  $i$ th interval:

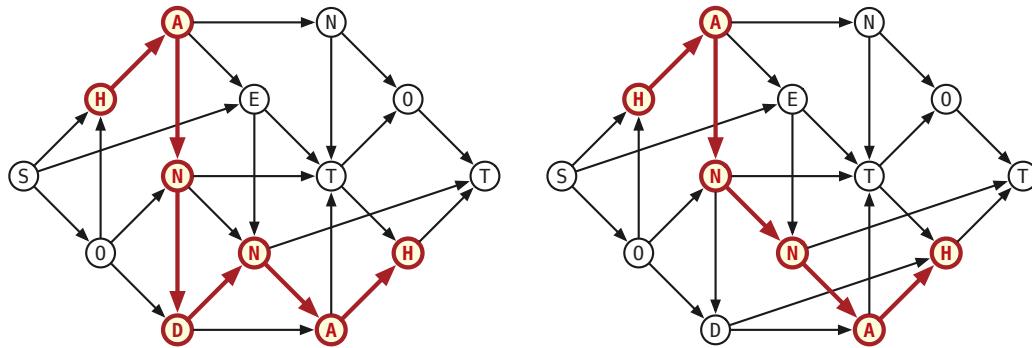
$$n_i := \# \{x \in S \mid b_{i-1} \leq x < b_i\}.$$

Then the **cost** of the resulting histogram is  $\sum_{i=1}^k (n_i(b_i - b_{i-1}))^2$ .

Describe and analyze an algorithm to compute a variable-width histogram with minimum cost for a given set of data values. Your input is an unsorted array  $S[1..n]$  of **distinct** real numbers, all strictly between 0 and 1, and an integer  $k$ . Your algorithm should return a sorted array  $B[0..k]$  of breakpoints that minimizes the cost of the resulting histogram, where  $B[0] = 0$  and  $B[k] = 1$ , **and every other breakpoint  $B[i]$  is equal to some input value  $S[j]$** .

2. Suppose we are given a directed acyclic graph  $G$  with labeled vertices. Every path in  $G$  has a label, which is a string obtained by concatenating the labels of its vertices in order. Recall that a *palindrome* is a string that is equal to its reversal.

Describe and analyze an algorithm to find the length of the longest palindrome that is the label of a path in  $G$ . For example, given the graph on the left below, your algorithm should return the integer 7, which is the length of the palindrome HANDNAH; given the graph on the right, your algorithm should return the integer 6, which is the length of the palindrome HANNAH.



# ∞ New CS 473: Algorithms, Spring 2015 ∞

## Homework 5

Due Tuesday, March 10, 2015 at 5pm

---

All homework must be submitted electronically via Moodle as separate PDF files, one for each numbered problem. Please see the course web site for more information.

---

1. On their long journey from Denmark to England, Rosencrantz and Guildenstern amuse themselves by playing the following game with a fair coin. First Rosencrantz flips the coin over and over until it comes up tails. Then Guildenstern flips the coin over and over until he gets as many heads in a row as Rosencrantz got on his turn. Here are three typical games:

Rosencrantz: **H H T**

Guildenstern: **H T H H**

Rosencrantz: **T**

Guildenstern: (no flips)

Rosencrantz: **H H H T**

Guildenstern: **T H H T H H T H T T H H H**

- (a) What is the *exact* expected number of flips in one of Rosencrantz's turns?
- (b) Suppose Rosencrantz happens to flip  $k$  heads in a row on his turn. What is the *exact* expected number of flips in Guildenstern's next turn?
- (c) What is the *exact* expected total number of flips (by both Rosencrantz and Guildenstern) in a single game?

Include formal proofs that your answers are correct. If you have to appeal to “intuition” or “common sense”, your answer is almost certainly wrong!

2. Recall from class that a **priority search tree** is a binary tree in which every node has both a *search key* and a *priority*, arranged so that the tree is simultaneously a binary search tree for the keys and a min-heap for the priorities. A **heater** is a priority search tree in which the *priorities* are given by the user, and the *search keys* are distributed uniformly and independently at random in the real interval  $[0, 1]$ . Intuitively, a heater is an “anti-treap”.

The following questions consider an  $n$ -node heater  $T$  whose priorities are the integers from 1 to  $n$ . Here we identify each node in  $T$  by its **priority rank**, rather than by the rank of its search keys; for example, “node 5” means the node in  $T$  with the 5th smallest *priority*. In particular, the min-heap property implies that node 1 is the root of  $T$ . Finally, let  $i$  and  $j$  be arbitrary integers such that  $1 \leq i < j \leq n$ .

- (a) What is the *exact* expected depth of node  $j$  in an  $n$ -node heater? Answering the following subproblems will help you:
- i. Prove that in a uniformly random permutation of the  $(i + 1)$ -element set  $\{1, 2, \dots, i, j\}$ , elements  $i$  and  $j$  are adjacent with probability  $2/(i + 1)$ .
  - ii. Prove that node  $i$  is an ancestor of node  $j$  with probability  $2/(i + 1)$ . [Hint: Use the previous question!]
  - iii. What is the probability that node  $i$  is a *descendant* of node  $j$ ? [Hint: Do **not** use the previous question!]
- (b) Describe and analyze an algorithm to insert a new item into a heater. Express the expected running time of the algorithm in terms of the priority rank of the newly inserted item.
- (c) Describe an algorithm to delete the minimum-priority item (the root) from an  $n$ -node heater. What is the expected running time of your algorithm?
3. Suppose we are given a two-dimensional array  $M[1..n, 1..n]$  in which every row and every column is sorted in increasing order and no two elements are equal.
- (a) Describe and analyze an algorithm to solve the following problem in  $O(n)$  time: Given indices  $i, j, i', j'$  as input, compute the number of elements of  $M$  smaller than  $M[i, j]$  and larger than  $M[i', j']$ .
  - (b) Describe and analyze an algorithm to solve the following problem in  $O(n)$  time: Given indices  $i, j, i', j'$  as input, return an element of  $M$  chosen uniformly at random from the elements smaller than  $M[i, j]$  and larger than  $M[i', j']$ . Assume the requested range is always non-empty.
  - (c) Describe and analyze a randomized algorithm to compute the median element of  $M$  in  $O(n \log n)$  expected time.

Assume you have access to a subroutine  $\text{RANDOM}(k)$  that returns an integer chosen independently and uniformly at random from the set  $\{1, 2, \dots, k\}$ , given an arbitrary positive integer  $k$  as input.

# ∞ New CS 473: Algorithms, Spring 2015 ∞

## Homework 6

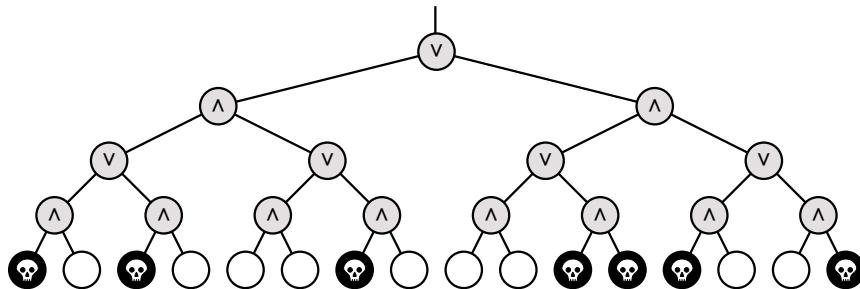
Due Tuesday, March 17, 2015 at 5pm

---

All homework must be submitted electronically via Moodle as separate PDF files, one for each numbered problem. Please see the course web site for more information.

---

1. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with  $4^n$  leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after  $2n$  moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.



You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are AND gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

- (a) Describe and analyze a deterministic algorithm to determine whether or not you can win. [Hint: This is easy, but be specific!]
- \*(b) **[Extra credit]** Prove that any deterministic algorithm that correctly determines whether you can win must examine every leaf in the tree. It follows that any correct algorithm for part (a) must take  $\Omega(4^n)$  time. [Hint: Let Death cheat, but not in a way that the algorithm can detect.]
- (c) Unfortunately, Death won't give you enough time to look at every node in the tree. Describe a randomized algorithm that determines whether you can win in  $O(3^n)$  expected time. [Hint: Consider the case  $n = 1$ .]
- \*(d) **[Extra credit]** Describe and analyze a randomized algorithm that determines whether you can win in  $O(c^n)$  expected time, for some explicit constant  $c < 3$ . Your analysis should yield an exact value for the constant  $c$ . [Hint: You may not need to change your algorithm from part (b) at all!]

2. A *meldable priority queue* stores a set of priorities from some totally-ordered universe (such as the integers) and supports the following operations:

- **MAKEQUEUE**: Return a new priority queue containing the empty set.
- **FINDMIN( $Q$ )**: Return the smallest element of  $Q$  (if any).
- **DELETEMIN( $Q$ )**: Remove the smallest element in  $Q$  (if any).
- **INSERT( $Q, x$ )**: Insert element  $x$  into  $Q$ , if it is not already there.
- **DECREASEPRIORITY( $Q, x, y$ )**: Replace an element  $x \in Q$  with a new element  $y < x$ . (If  $y \geq x$ , the operation fails.) The input includes a pointer directly to the node in  $Q$  containing  $x$ .
- **DELETE( $Q, x$ )**: Delete the element  $x \in Q$ . The input is a pointer directly to the node in  $Q$  containing  $x$ .
- **MELD( $Q_1, Q_2$ )**: Return a new priority queue containing all the elements of  $Q_1$  and  $Q_2$ ; this operation destroys  $Q_1$  and  $Q_2$ . The elements of  $Q_1$  and  $Q_2$  could be arbitrarily intermixed; we do *not* assume, for example, that every element of  $Q_1$  is smaller than every element of  $Q_2$ .

A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a priority, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm. The input consists of pointers to the roots of the two trees.

```
MELD( $Q_1, Q_2$ ):
    if  $Q_1 = \text{NULL}$  then return  $Q_2$ 
    if  $Q_2 = \text{NULL}$  then return  $Q_1$ 
    if  $\text{priority}(Q_1) > \text{priority}(Q_2)$ 
        swap  $Q_1 \leftrightarrow Q_2$ 
    with probability  $1/2$ 
         $\text{left}(Q_1) \leftarrow \text{MELD}(\text{left}(Q_1), Q_2)$ 
    else
         $\text{right}(Q_1) \leftarrow \text{MELD}(\text{right}(Q_1), Q_2)$ 
    return  $Q_1$ 
```

- (a) Prove that for *any* heap-ordered binary trees  $Q_1$  and  $Q_2$  (not just those constructed by the operations listed above), the expected running time of  $\text{MELD}(Q_1, Q_2)$  is  $O(\log n)$ , where  $n = |Q_1| + |Q_2|$ . [Hint: What is the expected length of a random root-to-leaf path in an  $n$ -node binary tree, where each left/right choice is made with equal probability?]
- (b) Prove that  $\text{MELD}(Q_1, Q_2)$  runs in  $O(\log n)$  time with high probability. [Hint: You can use Chernoff bounds, but the simpler identity  $\binom{c}{k} \leq (ce)^k$  is actually sufficient.]
- (c) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and  $O(1)$  additional time. (It follows that every operation takes  $O(\log n)$  time with high probability.)

# ∞ New CS 473: Algorithms, Spring 2015 ∞

## Homework 7

Due Tuesday, March 31, 2015 at 5pm

---

All homework must be submitted electronically via Moodle as separate PDF files, one for each numbered problem. Please see the course web site for more information.

---

1. **Reservoir sampling** is a method for choosing an item uniformly at random from an arbitrarily long stream of data; for example, the sequence of packets that pass through a router, or the sequence of IP addresses that access a given web page. Like all data stream algorithms, this algorithm must process each item in the stream quickly, using very little memory.

```
GETONESAMPLE(stream S):
    ℓ ← 0
    while S is not done
        x ← next item in S
        ℓ ← ℓ + 1
        if RANDOM(ℓ) = 1
            sample ← x      (*)
    return sample
```

At the end of the algorithm, the variable  $\ell$  stores the length of the input stream  $S$ ; this number is *not* known to the algorithm in advance. If  $S$  is empty, the output of the algorithm is (correctly!) undefined.

Consider an arbitrary non-empty input stream  $S$ , and let  $n$  denote the (unknown) length of  $S$ .

- Prove that the item returned by  $\text{GETONESAMPLE}(S)$  is chosen uniformly at random from  $S$ .
- What is the *exact* expected number of times that  $\text{GETONESAMPLE}(S)$  executes line  $(\star)$ ?
- What is the *exact* expected value of  $\ell$  when  $\text{GETONESAMPLE}(S)$  executes line  $(\star)$  for the *last* time?
- What is the *exact* expected value of  $\ell$  when either  $\text{GETONESAMPLE}(S)$  executes line  $(\star)$  for the *second* time (or the algorithm ends, whichever happens first)?
- Describe and analyze an algorithm that returns a subset of  $k$  distinct items chosen uniformly at random from a data stream of length at least  $k$ . The integer  $k$  is given as part of the input to your algorithm. Prove that your algorithm is correct.

For example, if  $k = 2$  and the stream contains the sequence  $\langle \spadesuit, \heartsuit, \diamondsuit, \clubsuit \rangle$ , the algorithm would return the subset  $\{\diamondsuit, \clubsuit\}$  with probability  $1/6$ .

# ∞ New CS 473: Algorithms, Spring 2015 ∞

## Homework 8

Due Tuesday, April 6, 2015 at 5pm

---

All homework must be submitted electronically via Moodle as separate PDF files, one for each numbered problem. Please see the course web site for more information.

---

1. Suppose you are given a directed graph  $G = (V, E)$ , two vertices  $s$  and  $t$ , a capacity function  $c : E \rightarrow \mathbb{R}^+$ , and a second function  $f : E \rightarrow \mathbb{R}$ .
  - (a) Describe and analyze an efficient algorithm to determine whether  $f$  is a maximum  $(s, t)$ -flow in  $G$ .
  - (b) Describe and analyze an efficient algorithm to determine whether  $f$  is the *unique* maximum  $(s, t)$ -flow in  $G$ .

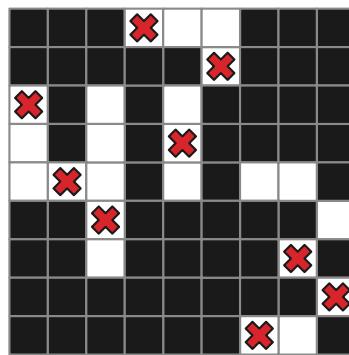
Do not assume anything about the function  $f$ .

2. A new assistant professor, teaching maximum flows for the first time, suggested the following greedy modification to the generic Ford-Fulkerson augmenting path algorithm. Instead of maintaining a residual graph, the greedy algorithm just reduces the capacity of edges along the augmenting path. In particular, whenever the algorithm saturates an edge, that edge is simply removed from the graph.

```
GREEDYFLOW( $G, c, s, t$ ):  
    for every edge  $e$  in  $G$   
         $f(e) \leftarrow 0$   
    while there is a path from  $s$  to  $t$  in  $G$   
         $\pi \leftarrow$  arbitrary path from  $s$  to  $t$  in  $G$   
         $F \leftarrow$  minimum capacity of any edge in  $\pi$   
        for every edge  $e$  in  $\pi$   
             $f(e) \leftarrow f(e) + F$   
            if  $c(e) = F$   
                remove  $e$  from  $G$   
            else  
                 $c(e) \leftarrow c(e) - F$   
    return  $f$ 
```

- (a) Prove that GREEDYFLOW does not always compute a maximum flow.
- (b) Prove that GREEDYFLOW is not even guaranteed to compute a good approximation to the maximum flow. That is, for any constant  $\alpha > 1$ , describe a flow network  $G$  such that the value of the maximum flow is more than  $\alpha$  times the value of the flow computed by GREEDYFLOW. [Hint: Assume that GREEDYFLOW chooses the worst possible path  $\pi$  at each iteration.]

- (c) Prove that for any flow network, if the Greedy Path Fairy tells you precisely which path  $\pi$  to use at each iteration, then GREEDYFLOW does compute a maximum flow. (Sadly, the Greedy Path Fairy does not actually exist.)
3. Suppose we are given an  $n \times n$  square grid, some of whose squares are colored black and the rest white. Describe and analyze an algorithm to determine whether tokens can be placed on the grid so that
- every token is on a white square;
  - every row of the grid contains exactly one token; and
  - every column of the grid contains exactly one token.



Your input is a two dimensional array  $IsWhite[1..n, 1..n]$  of booleans, indicating which squares are white. Your output is a single boolean. For example, given the grid above as input, your algorithm should return TRUE.

# ∞ New CS 473: Algorithms, Spring 2015 ∞

## Homework 9

Due Tuesday, April 21, 2015 at 5pm

---

All homework must be submitted electronically via Moodle as separate PDF files, one for each numbered problem. Please see the course web site for more information.

---

1. Suppose we are given an array  $A[1..m][1..n]$  of real numbers. We want to *round*  $A$  to an integer array, by replacing each entry  $x$  in  $A$  with either  $\lfloor x \rfloor$  or  $\lceil x \rceil$ , without changing the sum of entries in any row or column of  $A$ . For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

Describe and analyze an efficient algorithm that either rounds  $A$  in this fashion, or reports correctly that no such rounding is possible.

2. Suppose we are given a set of boxes, each specified by their height, width, and depth in centimeters. All three side lengths of every box lie strictly between 10cm and 20cm. As you should expect, one box can be placed inside another if the smaller box can be rotated so that its height, width, and depth are respectively smaller than the height, width, and depth of the larger box. Boxes can be nested recursively. Call a box *visible* if it is not inside another box.
  - (a) Describe and analyse an algorithm to find the largest subset of the given boxes that can be nested so that only one box is visible.
  - (b) Describe and analyze an algorithm to nest *all* the given boxes so that the number of visible boxes is as small as possible. [Hint: Do **not** use part (a).]
3. Describe and analyze an algorithm for the following problem, first posed and solved by the German mathematician Carl Jacobi in the early 1800s.□

*Disponantur nn quantitates  $h_k^{(i)}$  quaecunque in schema Quadrati, ita ut k habeantur n series horizontales et n series verticales, quarum quaeque est n terminorum. Ex illis quantitatibus eligantur n transversales, i.e. in seriebus horizontalibus simul atque verticalibus diversis positae, quod fieri potest 1.2...n modis; ex omnibus illis modis querendum est is, qui summam n numerorum electorum suppeditet maximam.*

---

□Carl Gustav Jacob Jacobi. De investigando ordine systematis aequationum differentialium vulgarium cujuscunque. *J. Reine Angew. Math.* 64(4):297–320, 1865. Posthumously published by Carl Borchardt.

For the few students who are not fluent in mid-19th century academic Latin, here is a modern English translation of Jacobi's problem. Suppose we are given an  $n \times n$  matrix  $M$ . Describe and analyze an algorithm that computes a permutation  $\sigma$  that maximizes the sum  $\sum_i M_{i,\sigma(i)}$ , or equivalently, permutes the columns of  $M$  so that the sum of the elements along the diagonal is as large as possible.

Please do not submit your solution in mid-19th century academic Latin.

# ∞ New CS 473: Algorithms, Spring 2015 ∞

## Homework 10

Due Tuesday, April 28, 2015 at 5pm

---

∞ This is the last graded homework. ∞

---

- Given points  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$  in the plane, the **linear regression problem** asks for real numbers  $a$  and  $b$  such that the line  $y = ax + b$  fits the points as closely as possible, according to some criterion. The most common fit criterion is the  **$L_2$  error**, defined as follows:

$$\varepsilon_2(a, b) = \sum_{i=1}^n (y_i - ax_i - b)^2.$$

(This is the error metric (*ordinary/linear*) *least squares*.)

But there are several other ways of measuring how well a line fits a set of points, some of which can be optimized via linear programming.

- The  **$L_1$  error** (or *total absolute deviation*) of the line  $y = ax + b$  is the sum of the vertical distances from the given points to the line:

$$\varepsilon_1(a, b) = \sum_{i=1}^n |y_i - ax_i - b|.$$

Describe a linear program whose solution  $(a, b)$  describes the line with minimum  $L_1$  error.

- The  **$L_\infty$  error** (or *maximum absolute deviation*) of the line  $y = ax + b$  is the maximum vertical distance from any given point to the line::

$$\varepsilon_\infty(a, b) = \max_{i=1}^n |y_i - ax_i - b|.$$

Describe a linear program whose solution  $(a, b)$  describes the line with minimum  $L_\infty$  error.

- Give a linear-programming formulation of the maximum-cardinality bipartite matching problem. The input is a bipartite graph  $G = (L \cup R, E)$ , where every edge connects a vertex in  $L$  (“on the left”) with a vertex in  $R$  (“on the right”). The output is the largest matching in  $G$ . Your linear program should have one variable for each edge.
  - Now dualize the linear program from part (a). What do the dual variables represent? What does the objective function represent? What problem is this!?
- An **integer program** is a linear program with the additional constraint that the variables must take only integer values. Prove that deciding whether a given integer program has a feasible solution is NP-hard. [Hint: Any NP-complete decision problem can be formulated as an integer program. Choose your favorite!]

# ∞ New CS 473: Algorithms, Spring 2015 ∞

## Homework 11

---

∅ This homework will not be graded. ∅

---

1. In Homework 10, we considered several different problems that can be solved by reducing them to a linear programming problem:

- Finding a line that fits a given set of  $n$  points in the plane with minimum  $L_1$  error.
- Finding a line that fits a given set of  $n$  points in the plane with minimum  $L_\infty$  error.
- Finding the largest matching in a bipartite graph.
- Finding the smallest vertex cover in a bipartite graph.

The specific linear programs are described in the homework solutions. For each of these linear programs, answer the following questions in the language of the original problem:

- (a) What is a basis?  
(b) (For the line-fitting problems only:) How many different bases are there?  
(c) What is a *feasible* basis?  
(d) What is a *locally optimal* basis?  
(e) What is a pivot?
2. Let  $G = (V, E)$  be an arbitrary directed graph with weighted vertices; vertex weights may be positive, negative, or zero. A *prefix* of  $G$  is a subset  $P \subseteq V$ , such that there is no edge  $u \rightarrow v$  where  $u \notin P$  but  $v \in P$ . A *suffix* of  $G$  is the complement of a prefix. Finally, an *interval* of  $G$  is the intersection of a prefix of  $G$  and a suffix of  $G$ . The weight of a prefix, suffix, or interval is the sum of the weights of its vertices.
  - (a) Describe a linear program that characterizes the maximum-weight prefix of  $G$ . Your linear program should have one variable per vertex, indicating whether that vertex is or is not in the chosen prefix.
  - (b) Describe a linear program that characterizes the maximum-weight interval of  $G$ .

[Hint: Don't worry about the solutions to your linear programs being integral; they will be. If all vertex weights are negative, the maximum-weight interval is empty; if all vertex weights are positive, the maximum-weight interval contains every vertex.]

**Write your answers in the separate answer booklet.**

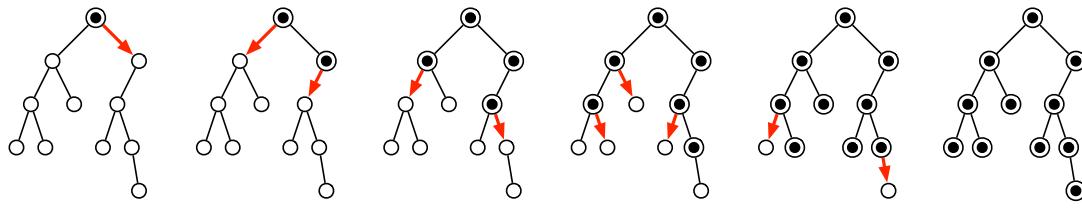
Please return this question sheet and your cheat sheet with your answers.

1. Recall that a boolean formula is in *conjunctive normal form* if it is the conjunction (AND) of a series of *clauses*, each of which is a disjunction (OR) of a series of literals, each of which is either a variable or the negation of a variable. Consider the following variants of SAT:

- **3SAT**: Given a boolean formula  $\Phi$  in conjunctive normal form, such that every clause in  $\Phi$  contains exactly *three* literals, is  $\Phi$  satisfiable?
  - **4SAT**: Given a boolean formula  $\Phi$  in conjunctive normal form, such that every clause in  $\Phi$  contains exactly *four* literals, is  $\Phi$  satisfiable?
- Describe a polynomial-time reduction from 3SAT to 4SAT.
  - Describe a polynomial-time reduction from 4SAT to 3SAT.

Don't forget to *prove* that your reductions are correct!

2. Suppose we need to distribute a message to all the nodes in a given binary tree. Initially, only the root node knows the message. In a single round, each node that knows the message is allowed (but not required) forward it to at most one of its children. Describe and analyze an algorithm to compute the minimum number of rounds required for the message to be delivered to all nodes in the tree.



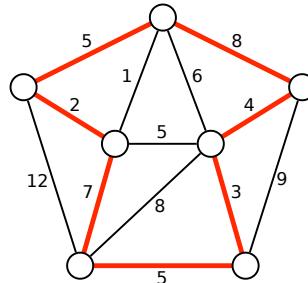
A message being distributed through a binary tree in five rounds.

3. The **maximum acyclic subgraph** problem is defined as follows: The input is a directed graph  $G = (V, E)$  with  $n$  vertices. Our task is to label the vertices from 1 to  $n$  so that the number of edges  $u \rightarrow v$  with  $\text{label}(u) < \text{label}(v)$  is as large as possible. Solving this problem exactly is NP-hard.

- Describe and analyze an efficient 2-approximation algorithm for this problem.
- Prove** that the approximation ratio of your algorithm is at most 2.

[Hint: Find an ordering of the vertices such that at least half of the edges point forward. Why is that enough?]

4. Let  $G$  be an undirected graph with weighted edges. A *heavy Hamiltonian cycle* is a cycle  $C$  that passes through each vertex of  $G$  exactly once, such that the total weight of the edges in  $C$  is at least half of the total weight of all edges in  $G$ . **Prove** that deciding whether a graph has a heavy Hamiltonian cycle is NP-hard.



A heavy Hamiltonian cycle. The cycle has total weight 34; the graph has total weight 67.

5. Lenny Rutenbar, founding dean of the new Maximilian Q. Levchin College of Computer Science, has commissioned a series of snow ramps on the south slope of the Orchard Downs sledding hill<sup>□</sup> and challenged William (Bill) Sanders, head of the Department of Electrical and Computer Engineering, to a sledding contest. Bill and Lenny will both sled down the hill, each trying to maximize their air time. The winner gets to expand their department/college into both Siebel Center and the new ECE Building; the loser has to move their entire department/college under the Boneyard bridge next to Everitt Lab.

Whenever Lenny or Bill reaches a ramp *while on the ground*, they can either use that ramp to jump through the air, possibly flying over one or more ramps, or sled past that ramp and stay on the ground. Obviously, if someone flies over a ramp, they cannot use that ramp to extend their jump.

Suppose you are given a pair of arrays  $Ramp[1..n]$  and  $Length[1..n]$ , where  $Ramp[i]$  is the distance from the top of the hill to the  $i$ th ramp, and  $Length[i]$  is the distance that a sledder who takes the  $i$ th ramp will travel through the air. Describe and analyze an algorithm to determine the maximum total distance that Lenny or Bill can spend in the air.

---

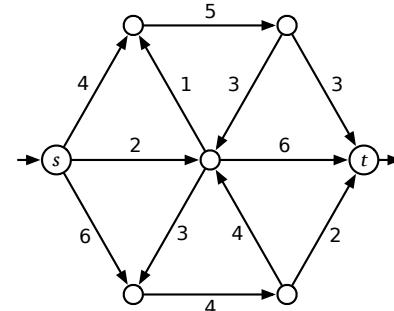
<sup>□</sup>The north slope is faster, but too short for an interesting contest.

**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

1. Clearly indicate the following structures in the directed graph on the right. Some of these subproblems may have more than one correct answer.

- (a) A maximum  $(s, t)$ -flow  $f$ .
- (b) The residual graph of  $f$ .
- (c) A minimum  $(s, t)$ -cut.



2. Recall that a family  $\mathcal{H}$  of hash functions is **universal** if  $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq 1/m$  for all distinct items  $x \neq y$ , where  $m$  is the size of the hash table. For any fixed hash function  $h$ , a **collision** is an unordered pair of distinct items  $x \neq y$  such that  $h(x) = h(y)$ .

Suppose we hash a set of  $n$  items into a table of size  $m = 2n$ , using a hash function  $h$  chosen uniformly at random from some universal family. Assume  $\sqrt{n}$  is an integer.

- (a) *Prove* that the expected number of collisions is at most  $n/4$ .
- (b) *Prove* that the probability that there are at least  $n/2$  collisions is at most  $1/2$ .
- (c) *Prove* that the probability that any subset of more than  $\sqrt{n}$  items all hash to the same address is at most  $1/2$ . [Hint: Use part (b).]
- (d) *[The actual exam question assumed only pairwise independence of hash values; under this weaker assumption, the claimed result is actually false. Everybody got extra credit for this part.]*

Now suppose we choose  $h$  at random from a **strongly 4-universal** family of hash functions, which means for all distinct items  $w, x, y, z$  and all addresses  $i, j, k, l$ , we have

$$\Pr_{h \in \mathcal{H}}[h(w) = i \wedge h(x) = j \wedge h(y) = k \wedge h(z) = l] = \frac{1}{m^4}.$$

*Prove* that the probability that any subset of more than  $\sqrt{n}$  items all hash to the same address is at most  $O(1/n)$ .

[Hint: Use Markov's and Chebyshev's inequalities. All four statements have short elementary proofs.]

3. Suppose we have already computed a maximum flow  $f^*$  in a flow network  $G$  with *integer* capacities. Assume all flow values  $f^*(e)$  are integers.
  - (a) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is *increased* by 1.
  - (b) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is *decreased* by 1.

Your algorithms should be significantly faster than recomputing the maximum flow from scratch.

4. Let  $T$  be a treap with  $n$  vertices.

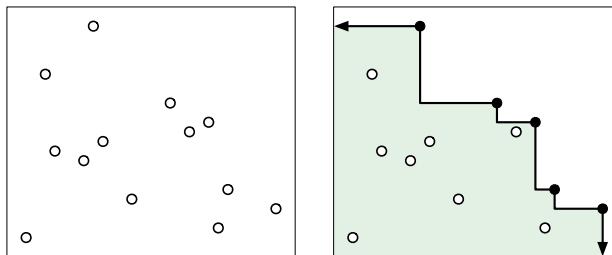
- (a) What is the *exact* expected number of leaves in  $T$ ?
- (b) What is the *exact* expected number of nodes in  $T$  that have two children?
- (c) What is the *exact* expected number of nodes in  $T$  that have exactly one child?

You do *not* need to prove that your answers are correct. [Hint: What is the probability that the node with the  $k$ th smallest search key has no children, one child, or two children?]

5. There is no problem 5.

**Write your answers in the separate answer booklet.**  
 You may take this question sheet with you when you leave.

- Let  $S$  be an arbitrary set of  $n$  points in the plane. A point  $p$  in  $S$  is **Pareto-optimal** if no other point in  $S$  is both above and to the right. The **staircase** of  $S$  is the set of all points in the plane (not just in  $S$ ) that have at least one point in  $S$  both above and to the right. All Pareto-optimal points lie on the boundary of the staircase.



A set of points in the plane and its staircase (shaded), with Pareto-optimal points in black.

- (a) Describe and analyze an algorithm that computes the staircase of  $S$  in  $O(n \log n)$  time.  
 (b) Suppose each point in  $S$  is chosen independently and uniformly at random from the unit square  $[0, 1] \times [0, 1]$ . What is the **exact** expected number of Pareto-optimal points in  $S$ ?

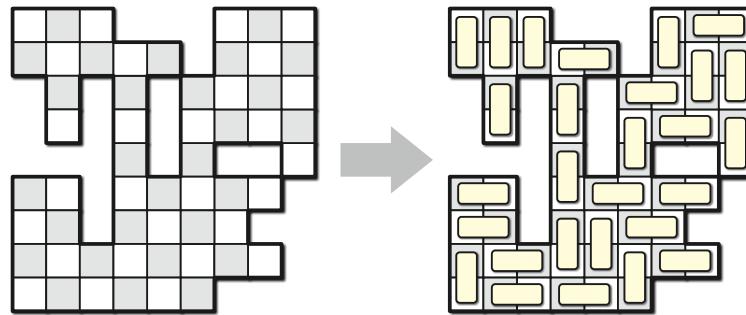
- Let  $G = (V, E)$  be a directed graph, in which every edge has capacity equal to 1 and some arbitrary cost. Edge costs could be positive, negative, or zero. Suppose you have just finished computing the minimum-cost circulation in this graph. Unfortunately, after all that work, now you realize that you recorded the direction of one of the edges incorrectly!

Describe and analyze an algorithm to update the minimum-cost circulation in  $G$  when the direction of an arbitrary edge in  $G$  is reversed. The input to your algorithm consists of the directed graph  $G$ , the costs of edges in  $G$ , the minimum-cost circulation in  $G$ , and the edge to be reversed. Your algorithm should be faster than recomputing the minimum-cost circulation from scratch.

- The **chromatic number**  $\chi(G)$  of an undirected graph  $G$  is the minimum number of colors required to color the vertices, so that every edge has endpoints with different colors. Computing the chromatic number exactly is NP-hard, because 3COLOR is NP-hard.

Prove that the following problem is also NP-hard: Given an arbitrary undirected graph  $G$ , return any integer between  $\chi(G)$  and  $\chi(G) + 473$ .

4. Suppose you are given an  $n \times n$  checkerboard with some of the squares deleted. You have a large set of dominos, just the right size to cover two squares of the checkerboard. Describe and analyze an algorithm to determine whether it is possible to tile the remaining squares with dominos—each domino must cover exactly two undeleted squares, and each undeleted square must be covered by exactly one domino.

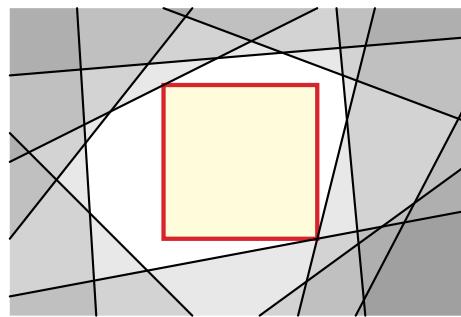


Your input is a two-dimensional array  $Deleted[1..n, 1..n]$  of bits, where  $Deleted[i, j] = \text{TRUE}$  if and only if the square in row  $i$  and column  $j$  has been deleted. Your output is a single bit; you do **not** have to compute the actual placement of dominos. For example, for the board shown above, your algorithm should return TRUE.

5. Recall from Homework 11 that a *prefix* of a directed graph  $G = (V, E)$  is a subset  $P \subseteq V$  of the vertices such that no edge  $u \rightarrow v \in E$  has  $u \notin P$  and  $v \in P$ .

Suppose you are given a rooted tree  $T$ , with all edges directed away from the root; every vertex in  $T$  has a weight, which could be positive, negative, or zero. Describe and analyze a *self-contained* algorithm to compute the prefix of  $T$  with maximum total weight. [Hint: Don't use linear programming.]

6. Suppose we are given a sequence of  $n$  linear inequalities of the form  $a_i x + b_i y \leq c_i$ ; the set of all points  $(x, y)$  that satisfy these inequalities is a convex polygon  $P$  in the  $(x, y)$ -plane. Describe a linear program whose solution describes the largest square with horizontal and vertical sides that lies inside  $P$ . (You can assume that  $P$  is non-empty.)



## CS 473 ✦ Spring 2016

## ❖ Homework 0 ❖

Due Tuesday, January 26, 2016 at 5pm

---

- **This homework tests your familiarity with prerequisite material:** designing, describing, and analyzing elementary algorithms (at the level of CS 225); fundamental graph problems and algorithms (again, at the level of CS 225); and especially facility with recursion and induction. Notes on most of this prerequisite material are available on the course web page.
  - **Each student must submit individual solutions for this homework.** For all future homeworks, groups of up to three students will be allowed to submit joint solutions.
  - **Submit your solutions electronically on the course Moodle site as PDF files.**
    - Submit a separate file for each numbered problem.
    - You can find a  $\text{\LaTeX}$  solution template on the course web site; please use it if you plan to typeset your homework.
    - If you must submit scanned handwritten solutions, use a black pen (not pencil) on blank white printer paper (not notebook or graph paper), use a high-quality scanner (not a phone camera), and print the resulting PDF file on a black-and-white printer to verify readability before you submit.
- 

## ❖ Some important course policies ❖

- **You may use any source at your disposal**—paper, electronic, or human—but you **must** cite **every** source that you use, and you **must** write everything yourself in your own words. See the academic integrity policies on the course web site for more details.
- The answer “*I don’t know*” (and *nothing else*) is worth 25% partial credit on any problem or subproblem, on any homework or exam, except for extra-credit problems. We will accept synonyms like “No idea” or “WTF” or “~\(\bullet\_-\bullet)/~”, but you must write *something*.
- **Avoid the Three Deadly Sins!** There are a few dangerous writing (and thinking) habits that will trigger an **automatic zero** on any homework or exam problem, unless your solution is nearly perfect otherwise. Yes, we are completely serious.
  - Always give complete solutions, not just examples.
  - Always declare all your variables, in English.
  - Never use weak induction.

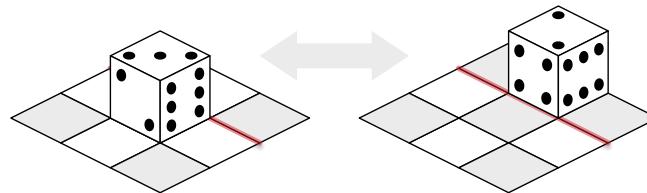
---

**See the course web site for more information.**

If you have any questions about these policies,  
please don’t hesitate to ask in class, in office hours, or on Piazza.

---

1. A *rolling die maze* is a puzzle involving a standard six-sided die (a cube with numbers on each side) and a grid of squares. You should imagine the grid lying on top of a table; the die always rests on and exactly covers one square. In a single step, you can *roll* the die 90 degrees around one of its bottom edges, moving it to an adjacent square one step north, south, east, or west.



Rolling a die.

Some squares in the grid may be *blocked*; the die must never be rolled onto a blocked square. Other squares may be *labeled* with a number; whenever the die rests on a labeled square, the number of pips on the *top* face of the die must equal the label. Squares that are neither labeled nor marked are *free*. You may not roll the die off the edges of the grid. A rolling die maze is *solvable* if it is possible to place a die on the lower left square and roll it to the upper right square under these constraints.

For example, here are two rolling die mazes. Black squares are blocked; empty white squares are free. The maze on the left can be solved by placing the die on the lower left square with 1 pip on the top face, and then rolling it north, then north, then east, then east. The maze on the right is not solvable.



Two rolling die mazes. Only the maze on the left is solvable.

Describe and analyze an efficient algorithm to determine whether a given rolling die maze is solvable. Your input is a two-dimensional array  $Label[1..n, 1..n]$ , where each entry  $Label[i, j]$  stores the label of the square in the  $i$ th row and  $j$ th column, where the label 0 means the square is free, and the label  $-1$  means the square is blocked.

[Hint: Build a graph. What are the vertices? What are the edges? Is the graph directed or undirected? Do the vertices or edges have weights? If so, what are they? What textbook problem do you need to solve on this graph? What textbook algorithm should you use to solve that problem? What is the running time of that algorithm as a function of  $n$ ? What does the number 24 have to do with anything?]

2. Describe and analyze fast algorithms for the following problems. The input for each problem is an unsorted array  $A[1..n]$  of  $n$  arbitrary numbers, which may be positive, negative, or zero, and which are not necessarily distinct.

- (a) Are there two distinct indices  $i < j$  such that  $A[i] + A[j] = 0$ ?
- (b) Are there three distinct indices  $i < j < k$  such that  $A[i] + A[j] + A[k] = 0$ ?

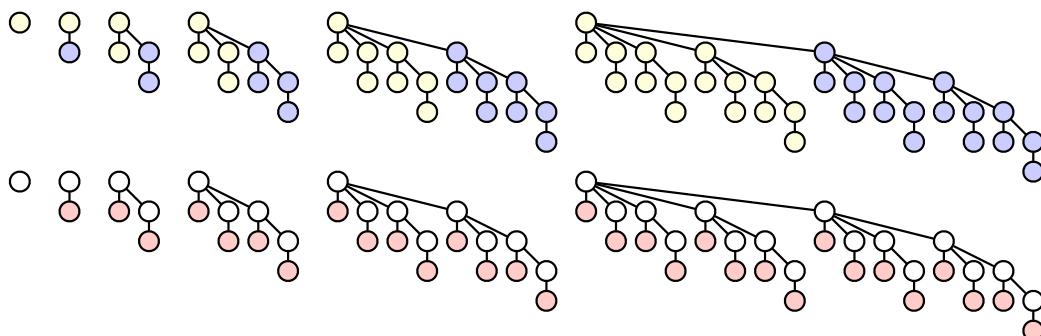
For example, if the input array is  $[2, -1, 0, 4, 0, -1]$ , both algorithms should return TRUE, but if the input array is  $[4, -1, 2, 0]$ , both algorithms should return FALSE. **You do not need to prove that your algorithms are correct.** [Hint: The devil is in the details.]

3. A *binomial tree of order  $k$*  is defined recursively as follows:

- A binomial tree of order 0 is a single node.
- For all  $k > 0$ , a binomial tree of order  $k$  consists of two binomial trees of order  $k - 1$ , with the root of one tree connected as a new child of the root of the other. (See the figure below.)

Prove the following claims:

- (a) For all non-negative integers  $k$ , a binomial tree of order  $k$  has exactly  $2^k$  nodes.
- (b) For all positive integers  $k$ , attaching a new leaf to every node in a binomial tree of order  $k - 1$  results in a binomial tree of order  $k$ .
- (c) For all non-negative integers  $k$  and  $d$ , a binomial tree of order  $k$  has exactly  $\binom{k}{d}$  nodes with depth  $d$ . (Hence the name!)



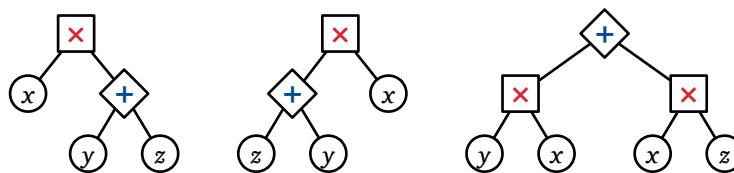
Binomial trees of order 0 through 5.

Top row: The recursive definition. Bottom row: The property claimed in part (b).

- \*4. [*Extra credit*] An *arithmetic expression tree* is a binary tree where every leaf is labeled with a variable, every internal node is labeled with an arithmetic operation, and every internal node has exactly two children. For this problem, assume that the only allowed operations are  $+$  and  $\times$ . Different leaves may or may not represent different variables.

Every arithmetic expression tree represents a function, transforming input values for the leaf variables into an output value for the root, by following two simple rules: (1) The value of any  $+$ -node is the sum of the values of its children. (2) The value of any  $\times$ -node is the product of the values of its children.

Two arithmetic expression trees are *equivalent* if they represent the same function; that is, the same input values for the leaf variables always leads to the same output value at both roots. An arithmetic expression tree is in *normal form* if the parent of every  $+$ -node (if any) is another  $+$ -node.



Three equivalent expression trees. Only the third expression tree is in normal form.

Prove that for any arithmetic expression tree, there is an equivalent arithmetic expression tree in normal form. [Hint: This is harder than it looks.]

## CS 473 ✦ Spring 2016

### ❖ Homework 1 ❖

Due Tuesday, February 2, 2016, at **8pm**

---

- Starting with this homework, groups of up to three students may submit joint solutions. **Group solutions must represent an honest collaborative effort by all members of the group.** Please see the academic integrity policies for more information.
  - You are responsible for forming your own groups. Groups can change from homework to homework, or even from (numbered) problem to problem.
  - Please make sure the names and NetIDs of *all* group members appear prominently at the top of the first page of each submission.
  - Please only upload one submission per group for each problem. In the Online Text box on the problem submission page, you must type in the NetIDs of *all* group members, including the person submitting. See the Homework Policies for examples. **Failure to enter all group NetIDs will delay (if not prevent) giving all group members the grades they deserve.**
- 
- For dynamic programming problems, a full-credit solution must include the following:
    - A clear English specification of the underlying recursive function. (For example: “Let  $Edit(i, j)$  denote the edit distance between  $A[1..i]$  and  $B[1..j]$ .”) **Omitting the English description is a Deadly Sin, which will result in an automatic zero.**
    - **One** of the following:
      - \* A correct recursive function or algorithm that computes the specified function, a clear description of the memoization structure, and a clear description of the iterative evaluation order.
      - \* Pseudocode for the final iterative dynamic programming algorithm.
    - The running time.
  - For problems that ask for an algorithm that computes an optimal *structure*—such as a subset, subsequence, partition, coloring, tree, or path—an algorithm that computes only the *value* or *cost* of the optimal structure is sufficient for full credit, unless the problem says otherwise.
  - Official solutions will provide target time bounds for full credit. Correct algorithms that are faster than the official solution will receive extra credit points; correct algorithms that are slower than the official solution will get partial credit. We rarely include these target time bounds in the actual questions, because when we do, more students submit fast but incorrect algorithms (worth 0/10 on exams) instead of correct but slow algorithms (worth 8/10 on exams).

1. Let's define a *summary* of two strings  $A$  and  $B$  to be a concatenation of substrings of the following form:

- $\blacktriangle SNA$  indicates a substring  $SNA$  of only the first string  $A$ .
- $\blacklozenge FOO$  indicates a common substring  $FOO$  of both strings.
- $\blacktriangledown BAR$  indicates a substring  $BAR$  of only the second string  $B$ .

A summary is *valid* if we can recover the original strings  $A$  and  $B$  by concatenating the appropriate substrings of the summary in order and discarding the delimiters  $\blacktriangle$ ,  $\blacklozenge$ , and  $\blacktriangledown$ . Each regular character has length 1, and each delimiter  $\blacktriangle$ ,  $\blacklozenge$ , or  $\blacktriangledown$  has some fixed non-negative length  $\Delta$ . The *length* of a summary is the sum of the lengths of its symbols.

For example, each of the following strings is a valid summary of the strings **KITTEN** and **KNITTING**:

- $\blacklozenge K \blacktriangledown N \blacklozenge I T T \blacktriangle A E \blacktriangledown I \blacklozenge N \blacktriangledown G$  has length  $9 + 7\Delta$ .
- $\blacklozenge K \blacktriangledown N \blacklozenge I T T \blacktriangle A E N \blacktriangledown I N G$  has length  $10 + 5\Delta$ .
- $\blacklozenge K \blacktriangle A I T T E N \blacktriangledown N I T T I N G$  has length  $13 + 3\Delta$ .
- $\blacktriangle A K I T T E N \blacktriangledown K N I T T I N G$  has length  $14 + 2\Delta$ .

Describe and analyze an algorithm that computes the length of the shortest summary of two given strings  $A[1..m]$  and  $B[1..n]$ . The delimiter length  $\Delta$  is also part of the input to your algorithm. For example:

- Given strings **KITTEN** and **KNITTING** and  $\Delta = 0$ , your algorithm should return 9.
- Given strings **KITTEN** and **KNITTING** and  $\Delta = 1$ , your algorithm should return 15.
- Given strings **KITTEN** and **KNITTING** and  $\Delta = 2$ , your algorithm should return 18.

2. Suppose you are given a sequence of positive integers separated by plus (+) and minus (-) signs; for example:

$$1 + 3 - 2 - 5 + 1 - 6 + 7$$

You can change the value of this expression by adding parentheses in different places. For example:

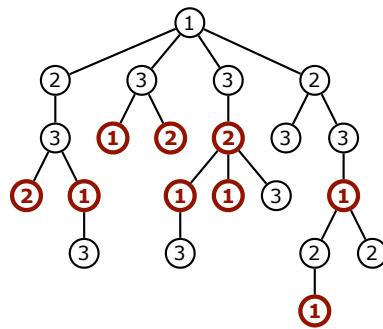
$$\begin{aligned} 1 + 3 - 2 - 5 + 1 - 6 + 7 &= -1 \\ (1 + 3 - (2 - 5)) + (1 - 6) + 7 &= 9 \\ (1 + (3 - 2)) - (5 + 1) - (6 + 7) &= -17 \end{aligned}$$

Describe and analyze an algorithm to compute the maximum possible value the expression can take by adding parentheses.

You may only use parentheses to group additions and subtractions; in particular, you are not allowed to create implicit multiplication as in  $1 + 3(-2)(-5) + 1 - 6 + 7 = 33$ .

3. The president of the Punxsutawney office of Giggle, Inc. has decided to give every employee a present to celebrate Groundhog Day! Each employee must receive one of three gifts: (1) an all-expenses-paid six-week vacation with Bill Murray,<sup>□</sup> (2) an all-the-Punxsutawney-pancakes-you-can-eat breakfast for two at [Punxy Phil's Family Restaurant](#), or (3) a burning paper bag of groundhog poop. Corporate regulations prohibit any employee from receiving exactly the same gift as their direct supervisor. Unfortunately, any employee who receives a better gift than their direct supervisor will almost certainly be fired in a fit of jealousy.

As Giggle-Punxsutawney's official gift czar, it's *your* job to decide which gift each employee receives. Describe an algorithm to distribute gifts so that the minimum number of people are fired. Yes, you can give the president groundhog poop.



A tree labeling with cost 9. The nine bold nodes have smaller labels than their parents.  
The president got a vacation with Bill Murray. This is *not* the optimal labeling for this tree.

More formally, you are given a rooted tree  $T$ , representing the company hierarchy, and you want to label each node in  $T$  with an integer 1, 2, or 3, so that every node has a different label from its parent. The **cost** of an labeling is the number of nodes that have smaller labels than their parents. Describe and analyze an algorithm to compute the minimum cost of any labeling of the given tree  $T$ .

---

<sup>□</sup>The details of scheduling  $n$  distinct six-week vacations with Bill Murray, all in a single year, are left as an [exercise for the reader](#).

## CS 473 ✦ Spring 2016

### ❖ Homework 2 ❖

Due Tuesday, February 9, 2016, at 8pm

---

1. [Insert amusing story about distributing polling stations or cell towers or Starbucks or something on a long straight road in rural Iowa. Ha ha ha, how droll.]

More formally, you are given a sorted array  $X[1..n]$  of distinct numbers and a positive integer  $k$ . A set of  $k$  intervals **covers**  $X$  if every element of  $X$  lies inside one of the  $k$  intervals. Your aim is to find  $k$  intervals  $[a_1, z_1], [a_2, z_2], \dots, [a_k, z_k]$  that cover  $X$  where the function  $\sum_{i=1}^k (z_i - a_i)^2$  is as small as possible. Intuitively, you are trying to cover the points with  $k$  intervals whose lengths are as close to equal as possible.

- (a) Describe an algorithm that finds  $k$  intervals with minimum total squared length that cover  $X$ . The running time of your algorithm should be a simple function of  $n$  and  $k$ .
- (b) Consider the two-dimensional matrix  $M[1..n, 1..n]$  defined as follows:

$$M[i, j] = \begin{cases} (X[j] - X[i])^2 & \text{if } i \leq j \\ \infty & \text{otherwise} \end{cases}$$

Prove that  $M$  satisfies the **Monge property**:  $M[i, j] + M[i', j'] \leq M[i, j'] + M[i', j]$  for all indices  $i < i'$  and  $j < j'$ .

- (c) [**Extra credit**] Describe an algorithm that finds  $k$  intervals with minimum total squared length that cover  $X$  **in  $O(nk)$  time**. [Hint: Solve part (a) first, then use part (b).]

We strongly recommend submitting your solution to part (a) separately, and only describing your changes to that solution for part (c).

2. The Doctor and River Song decide to play a game on a directed acyclic graph  $G$ , which has one source  $s$  and one sink  $t$ .<sup>1</sup>

Each player has a token on one of the vertices of  $G$ . At the start of the game, The Doctor's token is on the source vertex  $s$ , and River's token is on the sink vertex  $t$ . The players alternate turns, with The Doctor moving first. On each of his turns, the Doctor moves his token forward along a directed edge; on each of her turns, River moves her token backward along a directed edge.

If the two tokens ever meet on the same vertex, River wins the game. ("Hello, Sweetie!") If the Doctor's token reaches  $t$  or River's token reaches  $s$  before the two tokens meet, then the Doctor wins the game.

Describe and analyze an algorithm to determine who wins this game, assuming both players play perfectly. That is, if the Doctor can win *no matter how River moves*, then your algorithm should output "Doctor", and if River can win *no matter how the Doctor moves*, your algorithm should output "River". (Why are these the only two possibilities?) The input to your algorithm is the graph  $G$ .

---

<sup>1</sup>possibly short for the Untempered Schism and the Time Vortex, or the Shining World of the Seven Systems (otherwise known as Gallifrey) and Trenzalore, or Skaro and Telos, or something timey-wimey.

# CS 473 ◊ Spring 2016

## ◊ Homework 3 ◊

Due Tuesday, February 9, 2016, at 8pm

---

Unless a problem specifically states otherwise, you may assume a function `RANDOM` that takes a positive integer  $k$  as input and returns an integer chosen uniformly and independently at random from  $\{1, 2, \dots, k\}$  in  $O(1)$  time. For example, to flip a fair coin, you could call `RANDOM(2)`.

---

1. Suppose we want to write an efficient function `RANDOMPERMUTATION( $n$ )` that returns a permutation of the set  $\{1, 2, \dots, n\}$  chosen uniformly at random.

- (a) Prove that the following algorithm is *not* correct. [Hint: There is a one-line proof!]

```
RANDOMPERMUTATION( $n$ ):  
for  $i \leftarrow 1$  to  $n$   
     $\pi[i] \leftarrow i$   
for  $i \leftarrow 1$  to  $n$   
    swap  $\pi[i] \leftrightarrow \pi[\text{RANDOM}(n)]$ 
```

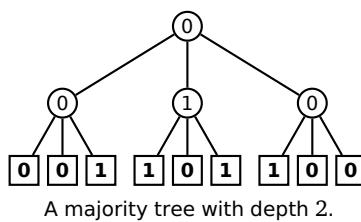
- (b) Consider the following implementation of `RANDOMPERMUTATION`.

```
RANDOMPERMUTATION( $n$ ):  
for  $i \leftarrow 1$  to  $n$   
     $\pi[i] \leftarrow \text{NULL}$   
for  $i \leftarrow 1$  to  $n$   
     $j \leftarrow \text{RANDOM}(n)$   
    while ( $\pi[j] \neq \text{NULL}$ )  
         $j \leftarrow \text{RANDOM}(n)$   
     $\pi[j] \leftarrow i$   
return  $\pi$ 
```

Prove that this algorithm is correct and analyze its expected running time.

- (c) Describe and analyze an implementation of `RANDOMPERMUTATION` that runs in expected worst-case time  $O(n)$ .

2. A **majority tree** is a complete ternary tree in which every leaf is labeled either 0 or 1. The *value* of a leaf is its label; the *value* of any internal node is the majority of the values of its three children. For example, if the tree has depth 2 and its leaves are labeled 1, 0, 0, 0, 1, 0, 1, 1, 1, the root has value 0.



It is easy to compute value of the root of a majority tree of depth  $n$  in  $O(3^n)$  time, given the sequence of  $3^n$  leaf labels as input, using a simple post-order traversal of the tree. Prove that this simple algorithm is optimal, and then describe a better algorithm. More formally:

- (a) Prove that *any* deterministic algorithm that computes the value of the root of a majority tree *must* examine every leaf. [Hint: Consider the special case  $n = 1$ . Recurse.]
  - (b) Describe and analyze a randomized algorithm that computes the value of the root in worst-case expected time  $O(c^n)$  for some explicit constant  $c < 3$ . [Hint: Consider the special case  $n = 1$ . Recurse.]
3. A **meldable priority queue** stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:
- **MAKEQUEUE**: Return a new priority queue containing the empty set.
  - **FINDMIN( $Q$ )**: Return the smallest element of  $Q$  (if any).
  - **DELETEMIN( $Q$ )**: Remove the smallest element in  $Q$  (if any).
  - **INSERT( $Q, x$ )**: Insert element  $x$  into  $Q$ , if it is not already there.
  - **DECREASEKEY( $Q, x, y$ )**: Replace an element  $x \in Q$  with a smaller key  $y$ . (If  $y > x$ , the operation fails.) The input is a pointer directly to the node in  $Q$  containing  $x$ .
  - **DELETE( $Q, x$ )**: Delete the element  $x \in Q$ . The input is a pointer directly to the node in  $Q$  containing  $x$ .
  - **MELD( $Q_1, Q_2$ )**: Return a new priority queue containing all the elements of  $Q_1$  and  $Q_2$ ; this operation destroys  $Q_1$  and  $Q_2$ .

A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm:

```
MELD( $Q_1, Q_2$ ):
    if  $Q_1$  is empty return  $Q_2$ 
    if  $Q_2$  is empty return  $Q_1$ 
    if  $key(Q_1) > key(Q_2)$ 
        swap  $Q_1 \leftrightarrow Q_2$ 
    with probability 1/2
         $left(Q_1) \leftarrow MELD(left(Q_1), Q_2)$ 
    else
         $right(Q_1) \leftarrow MELD(right(Q_1), Q_2)$ 
    return  $Q_1$ 
```

- (a) Prove that for *any* heap-ordered binary trees  $Q_1$  and  $Q_2$  (not just those constructed by the operations listed above), the expected running time of  $MELD(Q_1, Q_2)$  is  $O(\log n)$ , where  $n = |Q_1| + |Q_2|$ . [Hint: What is the expected length of a random root-to-leaf path in an  $n$ -node binary tree, where each left/right choice is made with equal probability?]
- (b) Prove that  $MELD(Q_1, Q_2)$  runs in  $O(\log n)$  time with high probability.
- (c) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and  $O(1)$  additional time. (It follows that each operation takes only  $O(\log n)$  time with high probability.)

**CS 473 ✦ Spring 2016**  
**∞ Homework 4 ∞**

Due Tuesday, March 1, 2016, at 8pm

---

Unless a problem specifically states otherwise, you may assume a function `RANDOM` that takes a positive integer  $k$  as input and returns an integer chosen uniformly and independently at random from  $\{1, 2, \dots, k\}$  in  $O(1)$  time. For example, to flip a fair coin, you could call `RANDOM(2)`.

---

1. Suppose we are given a two-dimensional array  $M[1..n, 1..n]$  in which every row and every column is sorted in increasing order and no two elements are equal.
  - (a) Describe and analyze an algorithm to solve the following problem in  $O(n)$  time: Given indices  $i, j, i', j'$  as input, compute the number of elements of  $M$  larger than  $M[i, j]$  and smaller than  $M[i', j']$ .
  - (b) Describe and analyze an algorithm to solve the following problem in  $O(n)$  time: Given indices  $i, j, i', j'$  as input, return an element of  $M$  chosen uniformly at random from the elements larger than  $M[i, j]$  and smaller than  $M[i', j']$ . Assume the requested range is always non-empty.
  - (c) Describe and analyze a randomized algorithm to compute the median element of  $M$  in  $O(n \log n)$  expected time.
2. **Tabulated hashing** uses tables of random numbers to compute hash values. Suppose  $|\mathcal{U}| = 2^w \times 2^w$  and  $m = 2^\ell$ , so the items being hashed are pairs of  $w$ -bit strings (or  $2w$ -bit strings broken in half) and hash values are  $\ell$ -bit strings.

Let  $A[0..2^w - 1]$  and  $B[0..2^w - 1]$  be arrays of independent random  $\ell$ -bit strings, and define the hash function  $h_{A,B} : \mathcal{U} \rightarrow [m]$  by setting

$$h_{A,B}(x, y) := A[x] \oplus B[y]$$

where  $\oplus$  denotes bit-wise exclusive-or. Let  $\mathcal{H}$  denote the set of all possible functions  $h_{A,B}$ . Filling the arrays  $A$  and  $B$  with independent random bits is equivalent to choosing a hash function  $h_{A,B} \in \mathcal{H}$  uniformly at random.

- (a) Prove that  $\mathcal{H}$  is 2-uniform.
- (b) Prove that  $\mathcal{H}$  is 3-uniform. [*Hint: Solve part (a) first.*]
- (c) Prove that  $\mathcal{H}$  is **not** 4-uniform.

Yes, “see part (b)” is worth full credit for part (a), but only if your solution to part (b) is correct.

**CS 473 ✦ Spring 2016**  
**❖ Homework 5 ❖**

Due Tuesday, March 1, 2016, at 8pm

---

Unless a problem specifically states otherwise, you may assume a function `RANDOM` that takes a positive integer  $k$  as input and returns an integer chosen uniformly and independently at random from  $\{1, 2, \dots, k\}$  in  $O(1)$  time. For example, to flip a fair coin, you could call `RANDOM(2)`.

---

1. *Reservoir sampling* is a method for choosing an item uniformly at random from an arbitrarily long stream of data.

```
GETONESAMPLE(stream S):
     $\ell \leftarrow 0$ 
    while  $S$  is not done
         $x \leftarrow$  next item in  $S$ 
         $\ell \leftarrow \ell + 1$ 
        if RANDOM( $\ell$ ) = 1
             $sample \leftarrow x$       (*)
```

return  $sample$

At the end of the algorithm, the variable  $\ell$  stores the length of the input stream  $S$ ; this number is *not* known to the algorithm in advance. If  $S$  is empty, the output of the algorithm is (correctly!) undefined. In the following, consider an arbitrary non-empty input stream  $S$ , and let  $n$  denote the (unknown) length of  $S$ .

- (a) Prove that the item returned by `GETONESAMPLE`( $S$ ) is chosen uniformly at random from  $S$ .
- (b) Describe and analyze an algorithm that returns a subset of  $k$  distinct items chosen uniformly at random from a data stream of length at least  $k$ . The integer  $k$  is given as part of the input to your algorithm. Prove that your algorithm is correct.

For example, if  $k = 2$  and the stream contains the sequence  $\langle \spadesuit, \heartsuit, \diamondsuit, \clubsuit \rangle$ , the algorithm should return the subset  $\{\diamondsuit, \spadesuit\}$  with probability  $1/6$ .

2. In this problem, we will derive a streaming algorithm that computes an accurate estimate  $\tilde{n}$  of the number of distinct items in a data stream  $S$ . Suppose  $S$  contains  $n$  unique items (but possibly several copies of each item); the algorithm does *not* know  $n$  in advance. Given an accuracy parameter  $0 < \varepsilon < 1$  and a confidence parameter  $0 < \delta < 1$  as part of the input, our final algorithm will guarantee that  $\Pr[|\tilde{n} - n| > \varepsilon n] < \delta$ .

As a first step, fix a positive integer  $m$  that is large enough that we don't have to worry about round-off errors in the analysis. Our first algorithm chooses a hash function  $h: \mathcal{U} \rightarrow [m]$  at random from a **2-uniform** family, computes the minimum hash value  $\hbar = \min\{h(x) \mid x \in S\}$ , and finally returns the estimate  $\tilde{n} = m/\hbar$ .

- (a) Prove that  $\Pr[\tilde{n} > (1 + \varepsilon)n] \leq 1/(1 + \varepsilon)$ . [Hint: Markov's inequality]
- (b) Prove that  $\Pr[\tilde{n} < (1 - \varepsilon)n] \leq 1 - \varepsilon$ . [Hint: Chebyshev's inequality]
- (c) We can improve this estimator by maintaining the  $k$  smallest hash values, for some integer  $k > 1$ . Let  $\tilde{n}_k = k \cdot m / \hbar_k$ , where  $\hbar_k$  is the  $k$ th smallest element of  $\{h(x) \mid x \in S\}$ .  
Estimate the smallest value of  $k$  (as a function of the accuracy parameter  $\varepsilon$ ) such that  $\Pr[|\tilde{n}_k - n| > \varepsilon n] \leq 1/4$ .
- (d) Now suppose we run  $d$  copies of the previous estimator in parallel to generate  $d$  independent estimates  $\tilde{n}_{k,1}, \tilde{n}_{k,2}, \dots, \tilde{n}_{k,d}$ , for some integer  $d > 1$ . Each copy uses its own independently chosen hash function, but they all use the same value of  $k$  that you derived in part (c). Let  $\tilde{N}$  be the *median* of these  $d$  estimates.  
Estimate the smallest value of  $d$  (as a function of the confidence parameter  $\delta$ ) such that  $\Pr[|\tilde{N} - n| > \varepsilon n] \leq \delta$ .

# CS 473 ✦ Spring 2016

## ❖ Homework 6 ❖

Due Tuesday, March 15, 2016, at 8pm

---

For problems that use maximum flows as a black box, a full-credit solution requires the following.

- A complete description of the relevant flow network, specifying the set of vertices, the set of edges (being careful about direction), the source and target vertices  $s$  and  $t$ , and the capacity of every edge. (If the flow network is part of the original input, just say that.)
- A description of the algorithm to construct this flow network from the stated input. This could be as simple as “We can construct the flow network in  $O(n^3)$  time by brute force.”
- A description of the algorithm to extract the answer to the stated problem from the maximum flow. This could be as simple as “Return TRUE if the maximum flow value is at least 42 and False otherwise.”
- A proof that your reduction is correct. This proof will almost always have two components. For example, if your algorithm returns a boolean, you should prove that its TRUE answers are correct and that its FALSE answers are correct. If your algorithm returns a number, you should prove that number is neither too large nor too small.
- The running time of the overall algorithm, expressed as a function of the original input parameters, not just the number of vertices and edges in your flow network.
- You may assume that maximum flows can be computed in  $O(VE)$  time. Do *not* regurgitate the maximum flow algorithm itself.

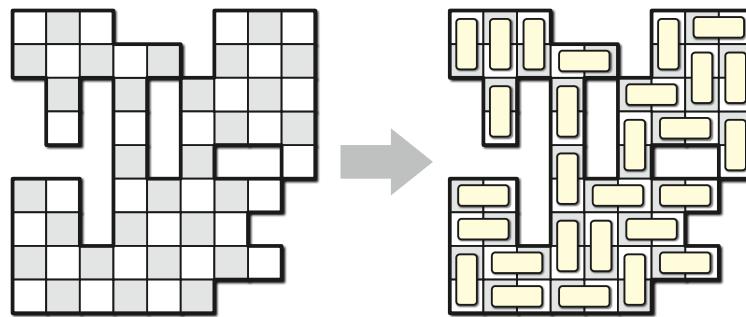
Reductions to other flow-based algorithms described in class or in the notes (for example: edge-disjoint paths, maximum bipartite matching, minimum-cost circulation) or to other standard graph problems (for example: reachability, minimum spanning tree, shortest paths) have similar requirements.

---

1. Suppose you are given a directed graph  $G = (V, E)$ , two vertices  $s$  and  $t$  in  $V$ , a capacity function  $c : E \rightarrow \mathbb{R}^+$ , and a second function  $f : E \rightarrow \mathbb{R}$ . Describe an algorithm to determine whether  $f$  is a maximum  $(s, t)$ -flow in  $G$ . Do not assume **anything** about the function  $f$ .
2. Suppose you have already computed a maximum flow  $f^*$  in a flow network  $G$  with **integer** edge capacities.
  - (a) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is increased by 1.
  - (b) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is decreased by 1.

Both algorithms should be significantly faster than recomputing the maximum flow from scratch.

3. Suppose you are given an  $n \times n$  checkerboard with some of the squares deleted. You have a large set of dominos, just the right size to cover two squares of the checkerboard. Describe and analyze an algorithm to determine whether it is possible to tile the board with dominos—each domino must cover exactly two undeleted squares, and each undeleted square must be covered by exactly one domino.



Your input is a two-dimensional array  $\text{Deleted}[1..n, 1..n]$  of bits, where  $\text{Deleted}[i, j] = \text{TRUE}$  if and only if the square in row  $i$  and column  $j$  has been deleted. Your output is a single bit; you do **not** have to compute the actual placement of dominos. For example, for the board shown above, your algorithm should return TRUE.

## CS 473 ✦ Spring 2016

## ❖ Homework 7 ❖

Due Tuesday, March 29, 2016, at 8pm

---

This is the last homework before Midterm 2.

---

- Suppose we are given a two-dimensional array  $A[1..m, 1..n]$  of non-negative real numbers. We would like to *round*  $A$  to an integer matrix, by replacing each entry  $x$  in  $A$  with either  $\lfloor x \rfloor$  or  $\lceil x \rceil$ , without changing the sum of entries in any row or column of  $A$ . For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

Describe and analyze an efficient algorithm that either rounds  $A$  in this fashion, or reports correctly that no such rounding exists.

- You're organizing the Third Annual UIUC Computer Science 72-Hour Dance Exchange, to be held all day Friday, Saturday, and Sunday in Siebel Center.<sup>□</sup> Several 30-minute sets of music will be played during the event, and a large number of DJs have applied to perform. You need to hire DJs according to the following constraints.

- Exactly  $k$  sets of music must be played each day, and thus  $3k$  sets altogether.
- Each set must be played by a single DJ in a consistent musical genre (ambient, bubblegum, dancehall, horrorcore, trip-hop, Nashville country, Chicago blues, axé, laïkó, skiffle, shape note, Nitzhonot, J-pop, K-pop, C-pop, T-pop, 8-bit, Tesla coil, . . . ).
- Each genre must be played at most once per day.
- Each DJ has given you a list of genres they are willing to play.
- No DJ can play more than five sets during the entire event.

Suppose there are  $n$  candidate DJs and  $g$  different musical genres available. Describe and analyze an efficient algorithm that either assigns a DJ and a genre to each of the  $3k$  sets, or correctly reports that no such assignment is possible.

- Describe and analyze an algorithm to determine, given an undirected<sup>□</sup> graph  $G = (V, E)$  and three vertices  $u, v, w \in V$  as input, whether  $G$  contains a simple path from  $u$  to  $w$  that passes through  $v$ .

---

<sup>□</sup>Efforts to secure overflow space in ECEB were sadly unsuccessful.

<sup>□</sup>This adjective is important; if the input graph were directed, this problem would be NP-hard.

# CS 473 ✦ Spring 2016

## ❖ Homework 8 ❖

Due Tuesday, April 12, 2016, at 8pm

You may assume the following results in your solutions:

- Maximum flows and minimum cuts can be computed in  $O(VE)$  time.
- Minimum-cost flows can be computed in  $O(E^2 \log^2 V)$  time.
- Linear programming problems with integer coefficients can be solved in polynomial time.

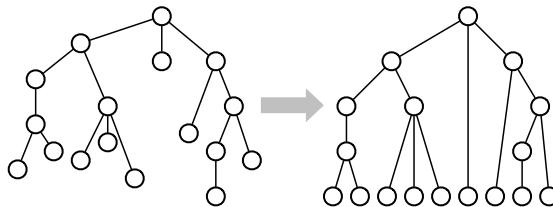
For problems that ask for a linear-programming formulation of some problem, a full credit solution requires the following components:

- A list of variables, along with a brief English description of each variable. (Omitting these English descriptions is a Deadly Sin.)
- A linear objective function (expressed either as minimization or maximization, whichever is more convenient), along with a brief English description of its meaning.
- A sequence of linear inequalities (expressed using  $\leq$ ,  $=$ , or  $\geq$ , whichever is more appropriate or convenient), along with a brief English description of each constraint.
- A proof that your linear programming formulation is correct, meaning that the optimal solution to the original problem can always be obtained from the optimal solution to the linear program. This may be very short.

It is **not** necessary to express the linear program in canonical form, or even in matrix form. Clarity is much more important than formality.

1. Suppose you are given a rooted tree  $T$ , where every edge  $e$  has two associated values: a non-negative *length*  $\ell(e)$ , and a *cost*  $\$(e)$  (which could be positive, negative, or zero). Your goal is to add a non-negative *stretch*  $s(e) \geq 0$  to the length of every edge  $e$  in  $T$ , subject to the following conditions:

- Every root-to-leaf path  $\pi$  in  $T$  has the same total stretched length  $\sum_{e \in \pi} (\ell(e) + s(e))$
- The total *weighted stretch*  $\sum_e s(e) \cdot \$\$(e)$  is as small as possible.



- (a) Describe an instance of this problem with no optimal solution.
- (b) Give a concise linear programming formulation of this problem. (For the instance described in part (a), your linear program will be unbounded.)
- (c) Suppose that for the given tree  $T$  and the given lengths and costs, the optimal solution to this problem is unique. Prove that in this optimal solution, we have  $s(e) = 0$  for every edge on some longest root-to-leaf path in  $T$ . In other words, prove that the optimally stretched tree with the same depth as the input tree. [Hint: What is a basis in your linear program? What is a feasible basis?]

Problem 1(c) originally omitted the uniqueness assumption and asked for a proof that *every* optimal solution has an unstretched root-to-leaf path, but that more general claim is false. For example, if every edge has cost zero, there are optimal solutions in which every edge has positive stretch.

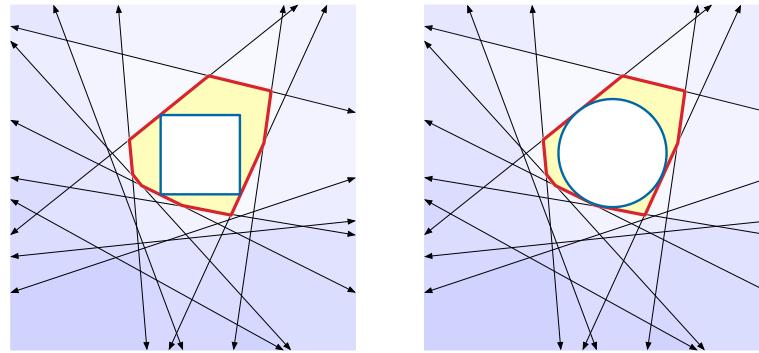
2. Describe and analyze an efficient algorithm for the following problem, first posed and solved by the German mathematician Carl Jacobi in the early 1800s.  $\square$

*Disponantur nn quantitates  $h_k^{(i)}$  quaecunque in schema Quadrati, ita ut k habeantur n series horizontales et n series verticales, quarum quaeque est n terminorum. Ex illis quantitatibus eligantur n transversales, i.e. in seriebus horizontalibus simul atque verticalibus diversis positae, quod fieri potest 1.2...n modis; ex omnibus illis modis quaerendum est is, qui summam n numerorum electorum suppedit maximam.*

For those few students who are not fluent in mid-19th century academic Latin, here is a modern English translation of Jacobi's problem. Suppose we are given an  $n \times n$  matrix  $M$ . Describe and analyze an algorithm that computes a permutation  $\sigma$  that maximizes the sum  $\sum_{i=1}^n M_{i,\sigma(i)}$ , or equivalently, permutes the columns of  $M$  so that the sum of the elements along the diagonal is as large as possible.

Please do not submit your solution in mid-19th century academic Latin.

3. Suppose we are given a sequence of  $n$  linear inequalities of the form  $a_i x + b_i y \leq c_i$ . Collectively, these  $n$  inequalities describe a convex polygon  $P$  in the plane.
  - (a) Describe a linear program whose solution describes the largest square with horizontal and vertical sides that lies entirely inside  $P$ .
  - (b) Describe a linear program whose solution describes the largest circle that lies entirely inside  $P$ .




---

$\square$ Carl Gustav Jacob Jacobi. De investigando ordine systematis aequationum differentialium vulgarium cujuscunque. *J. Reine Angew. Math.* 64(4):297–320, 1865. Posthumously published by Carl Borchardt.

# CS 473 ✦ Spring 2016

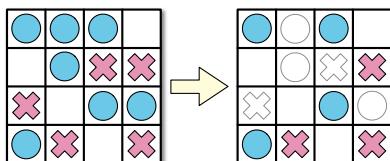
## ❖ Homework 9 ❖

Due Tuesday, April 19, 2016, at 8pm

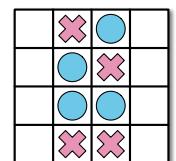
For problems that ask to prove that a given problem  $X$  is NP-hard, a full-credit solution requires the following components:

- Specify a known NP-hard problem  $Y$ , taken from the problems listed in the notes.
- Describe a polynomial-time algorithm for  $Y$ , using a black-box polynomial-time algorithm for  $X$  as a subroutine. Most NP-hardness reductions have the following form: Given an arbitrary instance of  $Y$ , describe how to transform it into an instance of  $X$ , pass this instance to a black-box algorithm for  $X$ , and finally, describe how to transform the output of the black-box subroutine to the final output. A cartoon with boxes may be helpful.
- Prove that your reduction is correct. As usual, correctness proofs for NP-hardness reductions usually have two components (“one for each f”).

1. Consider the following solitaire game. The puzzle consists of an  $n \times m$  grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) every row contains at least one stone, and (2) no column contains stones of both colors. For some initial configurations of stones, reaching this goal is impossible.



A solvable puzzle and one of its many solutions.



An unsolvable puzzle.

Prove that it is NP-hard to determine, given an initial configuration of red and blue stones, whether the puzzle can be solved.

2. Everyone's having a wonderful time at the party you're throwing, but now it's time to line up for *The Algorithm March* (アルゴリズムこうしん)! This dance was originally developed by the Japanese comedy duo Itsumo Kokokara (いつもここから) for the children's television show *PythagoraSwitch* (ピタゴラスイッチ). The Algorithm March is performed by a line of people; each person in line starts a specific sequence of movements one measure later than the person directly in front of them. Thus, the march is the dance equivalent of a musical round or canon, like "Row Row Row Your Boat". □ Proper etiquette dictates that each marcher must know the person directly in front of them in line, lest a minor mistake during lead to horrible embarrassment between strangers.

Suppose you are given a complete list of which people at your party know each other. Prove that it is NP-hard to determine the largest number of party-goers that can participate in the Algorithm March. You may assume without loss of generality that there are no ninjas at your party.

**CS 473 ✦ Spring 2016**  
**❖ Homework 10 ❖**

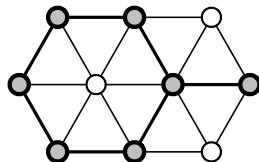
Due Tuesday, April 26, 2016, at 8pm

---

**❖ This is the last graded homework of the semester. ❖**

---

1. A *double-Hamiltonian circuit* in an undirected graph  $G$  is a closed walk that visits every vertex in  $G$  exactly twice. Prove that determining whether a given undirected graph contains a double-Hamiltonian circuit is NP-hard.
2. A subset  $S$  of vertices in an undirected graph  $G$  is called *triangle-free* if, for every triple of vertices  $u, v, w \in S$ , at least one of the three edges  $uv, uw, vw$  is *absent* from  $G$ . Prove that finding the size of the largest triangle-free subset of vertices in a given undirected graph is NP-hard.



A triangle-free subset of 7 vertices.

This is **not** the largest triangle-free subset in this graph.

3. Suppose you are given a magic black box that can determine **in polynomial time**, given an arbitrary graph  $G$ , whether  $G$  is 3-colorable. Describe and analyze a **polynomial-time** algorithm that either computes a proper 3-coloring of a given graph or correctly reports that no such coloring exists, using the magic black box as a subroutine. [Hint: The input to the magic black box is a graph. Just a graph. Vertices and edges. Nothing else.]

# CS 473 ✦ Spring 2016

## ❖ Homework 11 ❖

Solutions will be released on Tuesday, May 3, 2016.

---

This homework will not be graded.

However, material covered by this homework *may* appear on the final exam.

---

1. The *linear arrangement* problem asks, given an  $n$ -vertex directed graph as input, for an ordering  $v_1, v_2, \dots, v_n$  of the vertices that maximizes the number of forward edges: directed edges  $v_i \rightarrow v_j$  such that  $i < j$ . Describe and analyze an efficient 2-approximation algorithm for this problem. (Solving this problem exactly is NP-hard.)
2. Let  $G = (V, E)$  be an undirected graph, each of whose vertices is colored either red, green, or blue. An edge in  $G$  is *boring* if its endpoints have the same color, and *interesting* if its endpoints have different colors. The *most interesting 3-coloring* is the 3-coloring with the maximum number of interesting edges, or equivalently, with the fewest boring edges. Computing the most interesting 3-coloring is NP-hard, because the standard 3-coloring problem is a special case.
  - (a) Let  $\text{wow}(G)$  denote the number of interesting edges in the most interesting 3-coloring of  $G$ . Suppose we independently assign each vertex in  $G$  a *random* color from the set {red, green, blue}. Prove that the expected number of interesting edges is at least  $\frac{2}{3}\text{wow}(G)$ .
  - (b) Prove that with high probability, the expected number of interesting edges is at least  $\frac{1}{2}\text{wow}(G)$ . [Hint: Use Chebyshev's inequality. But wait... How do we know that we can use Chebyshev's inequality?]
  - (c) Let  $\text{zzz}(G)$  denote the number of boring edges in the most interesting 3-coloring of a graph  $G$ . Prove that it is NP-hard to approximate  $\text{zzz}(G)$  within a factor of  $10^{10^{100}}$ .
3. Suppose we want to schedule a give set of  $n$  jobs on a machine containing a row of  $p$  identical processors. Our input consists of two arrays  $\text{duration}[1..n]$  and  $\text{width}[1..n]$ . A valid schedule consists of two arrays  $\text{start}[1..n]$  and  $\text{first}[1..n]$  that satisfy the following constraints:
  - $\text{start}[j] \geq 0$  for all  $j$ .
  - The  $j$ th job runs on processors  $\text{first}[j]$  through  $\text{first}[j] + \text{width}[j] - 1$ , starting at time  $\text{start}[j]$  and ending at time  $\text{start}[j] + \text{duration}[j]$ .
  - No processor can run more than one job simultaneously.The *makespan* of a schedule is the largest finishing time:  $\max_j(\text{start}[j] + \text{duration}[j])$ . Our goal is to compute a valid schedule with the smallest possible makespan.
  - (a) Prove that this scheduling problem is NP-hard.

The *makespan* of a schedule is the largest finishing time:  $\max_j(\text{start}[j] + \text{duration}[j])$ . Our goal is to compute a valid schedule with the smallest possible makespan.

- (a) Prove that this scheduling problem is NP-hard.

- (b) Describe a polynomial-time algorithm that computes a 3-approximation of the minimum makespan of the given set of jobs. That is, if the minimum makespan is  $M$ , your algorithm should compute a schedule with makespan at most  $3M$ . You may assume that  $p$  is a power of 2. [*Hint: Assume that  $p$  is a power of 2.*]
- (c) Describe an algorithm that computes a 3-approximation of the minimum makespan of the given set of jobs *in  $O(n \log n)$  time*. Again, you may assume that  $p$  is a power of 2.

These are the standard 10-point rubrics that we will use for certain types of exam questions. When these problems appear in the homework, a score of  $x$  on this 10-point scale corresponds to a score of  $\lceil x/3 \rceil$  on the 4-point homework scale.

---

## Proof by Induction

- 2 points for stating a valid **strong** induction hypothesis.
  - The inductive hypothesis need not be stated explicitly if it is a mechanical translation of the theorem (that is, “Assume  $P(k)$  for all  $k < n$ ” when the theorem is “ $P(n)$  for all  $n$ ”) *and* it is applied correctly. However, if the proof requires a stronger induction hypothesis (“Assume  $P(k)$  and  $Q(k)$  for all  $k < n$ ”) then it must be stated explicitly.
  - By course policy, ***stating a weak inductive hypothesis triggers an automatic zero***, unless the proof is otherwise **perfect**.
  - **Ambiguous** induction hypotheses like “Assume the statement is true for all  $k < n$ .” are not valid. *What* statement? The theorem you’re trying to prove doesn’t use the variable  $k$ , so that can’t possibly be the statement you mean.
  - **Meaningless** induction hypotheses like “Assume that  $k$  is true for all  $k < n$ ” are not valid. Only propositions can be true or false;  $k$  is an integer, not a proposition.
  - **False** induction hypotheses like “Assume that  $k < n$  for all  $k$ ” are not valid. The inequality  $k < n$  does *not* hold for all  $k$ , because it does not hold when  $k = n + 5$ .
- 1 point for explicit and clearly exhaustive case analysis.
  - No penalty for overlapping or redundant cases. However, mistakes in redundant cases are still penalized.
- 2 points for the base case(s).
- 2 point for correctly applying the *stated* inductive hypothesis.
  - It is not possible to correctly apply an invalid inductive hypothesis.
  - No credit for correctly applying a different induction hypothesis than the one stated.
- 3 points for other details of the inductive case(s).

## Dynamic Programming

- **6 points for a correct recurrence**, described either using functional notation or as pseudocode for a recursive algorithm.
  - + 1 point for a clear **English** description of the function you are trying to evaluate. (Otherwise, we don't even know what you're *trying* to do.) **Automatic zero if the English description is missing.**
  - + 1 point for stating how to call your recursive function to get the final answer.
  - + 1 point for the base case(s).  $-1/2$  for one *minor* bug, like a typo or an off-by-one error.
  - + 3 points for the recursive case(s).  $-1$  for each *minor* bug, like a typo or an off-by-one error. **No credit for the rest of the problem if the recursive case(s) are incorrect.**
- **4 points for iterative details**
  - + 1 point for describing the memoization data structure; a clear picture may be sufficient.
  - + 2 points for describing a correct evaluation order; a clear picture may be sufficient. If you use nested loops, be sure to specify the nesting order.
  - + 1 point for running time
- Proofs of correctness are not required for full credit on exams, unless the problem specifically asks for one.
- Do not analyze (or optimize) space.
- For problems that ask for an algorithm that computes an optimal *structure*—such as a subset, partition, subsequence, or tree—an algorithm that computes only the *value* or *cost* of the optimal structure is sufficient for full credit, unless the problem says otherwise.
- Official solutions usually include pseudocode for the final iterative dynamic programming algorithm, **but iterative pseudocode is not required for full credit**. If your solution includes iterative pseudocode, you do not need to separately describe the recurrence, memoization structure, or evaluation order. However, you **must** give an English description of the underlying recursive function.
- Official solutions will provide target time bounds. Algorithms that are faster than this target are worth more points; slower algorithms are worth fewer points, typically by 2 or 3 points (out of 10) for each factor of  $n$ . Partial credit is **scaled** to the new maximum score. All points above 10 are recorded as extra credit.

We rarely include these target time bounds in the actual questions, because when we have included them, significantly more students turned in algorithms that meet the target time bound but didn't work (earning 0/10) instead of correct algorithms that are slower than the target time bound (earning 8/10).

## Graph Reductions

For problems solved by reducing them to a standard graph algorithm covered either in class or in a prerequisite class (for example: shortest paths, topological sort, minimum spanning trees, maximum flows, bipartite maximum matching, vertex-disjoint paths, . . .):

- **1 point for listing the vertices of the graph.** (If the original input is a graph, describing how to modify that graph is fine.)
- **1 point for listing the edges of the graph**, including whether the edges are directed or undirected. (If the original input is a graph, describing how to modify that graph is fine.)
- **1 point for describing appropriate weights** and/or lengths and/or capacities and/or costs and/or demands and/or whatever for the vertices and edges.
- **2 points for an explicit description of the problem being solved on that graph.** (For example: “We compute the maximum number of vertex-disjoint paths in  $G$  from  $v$  to  $z$ . ”)
- **3 points for other algorithmic details**, assuming the rest of the reduction is correct.
  - + 1 point for describing how to build the graph from the original input (for example: “by brute force”)
  - + 1 point for describing the algorithm you use to solve the graph problem (for example: “Orlin’s algorithm” or “as described in class”)
  - + 1 point for describing how to extract the output for the original problem from the output of the graph algorithm.
- **2 points for the running time**, expressed in terms of the original input parameters, not just  $V$  and  $E$ .
- **If the problem explicitly asks for a proof of correctness**, divide all previous points in half and add **5 points for proof of correctness**. These proofs almost always have two parts; for example, for algorithms that return TRUE or FALSE:
  - $2\frac{1}{2}$  points for proving that if your algorithm returns TRUE, then the correct answer is TRUE.
  - $2\frac{1}{2}$  points for proving that if your algorithm returns FALSE, then the correct answer is FALSE.

These proofs do not need to be as detailed as in the homeworks; we are really just looking for compelling evidence that *you* understand why your reduction is correct.

- It is still possible to get partial credit for an incorrect algorithm. For example, if you describe an algorithm that sometimes reports false positives, but you prove that all FALSE answers are correct, you would still get  $2\frac{1}{2}$  points for half of the correctness proof.

## NP-Hardness Reductions

For problems that ask “**Prove** that X is NP-hard”:

- **4 points for the polynomial-time reduction:**
  - 1 point for explicitly naming the NP-hard problem Y to reduce from. You may use any of the problems listed in the lecture notes; a list of NP-hard problems will appear on the back page of the exam.
  - 2 points for describing the polynomial-time algorithm to transform arbitrary instances of Y into inputs to the black-box algorithm for X
  - 1 point for describing the polynomial-time algorithm to transform the output of the black-box algorithm for X into the output for Y.
  - Reductions that call the black-box algorithm for X more than once are perfectly acceptable. You do *not* need to explicitly analyze the running time of your resulting algorithm for Y, but it must be polynomial in the size of the input instance of Y.
- **6 points for the proof of correctness. This is the entire point of the problem.** These proofs always have two parts; for example, if X and Y are both decision problems:
  - 3 points for proving that your reduction transforms positive instances of Y into positive instances of X.
  - 3 points for proving that your reduction transforms negative instances of Y into negative instances of X.

These proofs do not need to be as detailed as in the homeworks; however, it must be clear that you have at least considered all possible cases. We are really just looking for compelling evidence that *you* understand why your reduction is correct.

- It is still possible to get partial credit for an incorrect reduction. For example, if you describe a reduction that sometimes reports false positives, but you prove that all FALSE answers are correct, you would still get 3 points for half of the correctness proof.
- Zero points for reducing X **to** some NP-hard problem Y.
- Zero points for attempting to solve X.

## Approximation Algorithms

For problems that ask you to describe a polynomial-time approximation algorithm for some NP-hard problem X, analyze its approximation ratio, and prove that your approximation analysis is correct:

- **4 points for the actual approximation algorithm.** You do not need to analyze the running time of your algorithm (unless we explicitly ask for the running time), but it must clearly run in polynomial time. If we give you the algorithm, ignore this part and scale the rest of the rubric up to 10 points.
- **2 points for stating the correct approximation ratio.** If we give you the approximation ratio, ignore this part and scale the rest of the rubric up to 10 points.
- **4 points for proving that the stated approximation ratio is correct.** If we do not *explicitly* ask for a proof, ignore this part and scale the rest of the rubric up to 10 points.

For example, suppose we give you an algorithm and ask for its approximation ratio, but we do not explicitly ask for a proof. If the given algorithm is a 3-approximation algorithm, then you would get full credit for writing “3”.

**Write your answers in the separate answer booklet.**  
Please return this question sheet and your cheat sheet with your answers.

---

1. For any positive integer  $n$ , the  $n$ th **Fibonacci string**  $F_n$  is defined recursively as follows, where  $x \bullet y$  denotes the concatenation of strings  $x$  and  $y$ :

$$\begin{aligned} F_1 &:= \mathbf{0} \\ F_2 &:= \mathbf{1} \\ F_n &:= F_{n-1} \bullet F_{n-2} \quad \text{for all } n \geq 3 \end{aligned}$$

For example,  $F_3 = \mathbf{10}$  and  $F_4 = \mathbf{101}$ .

- (a) What is  $F_8$ ?  
(b) **Prove** that every Fibonacci string except  $F_1$  starts with  $\mathbf{1}$ .  
(c) **Prove** that no Fibonacci string contains the substring  $\mathbf{00}$ .
2. You have reached the inevitable point in the semester where it is no longer possible to finish all of your assigned work without pulling at least a few all-nighters. The problem is that pulling successive all-nighters will burn you out, so you need to pace yourself (or something).

Let's model the situation as follows. There are  $n$  days left in the semester. For simplicity, let's say you are taking one class, there are no weekends, there is an assignment due every single day until the end of the semester, and you will only work on an assignment the day before it is due. For each day  $i$ , you know two positive integers:

- $Score[i]$  is the score you will earn on the  $i$ th assignment if you do *not* pull an all-nighter the night before.
- $Bonus[i]$  is the number of additional points you could potentially earn if you *do* pull an all-nighter the night before.

However, pulling multiple all-nighters in a row has a price. If you turn in the  $i$ th assignment immediately after pulling  $k$  consecutive all-nighters, your actual score for that assignment will be  $(Score[i] + Bonus[i])/2^{k-1}$ .

Design and analyze an algorithm that computes the maximum total score you can achieve, given the arrays  $Score[1..n]$  and  $Bonus[1..n]$  as input.

3. The following algorithm finds the smallest element in an unsorted array. The subroutine `SHUFFLE` randomly permutes the input array  $A$ ; every permutation of  $A$  is equally likely.

```

RANDOMMIN( $A[1..n]$ ):
   $min \leftarrow \infty$ 
  SHUFFLE( $A$ )
  for  $i \leftarrow 1$  to  $n$ 
    if  $A[i] < min$ 
       $min \leftarrow A[i]$       ( $\star$ )
  return  $min$ 

```

In the following questions, assume all elements in the input array  $A[ ]$  are distinct.

- (a) In the worst case, how many times does `RANDOMMIN` execute line  $(\star)$ ?
- (b) For each index  $i$ , let  $X_i = 1$  if line  $(\star)$  is executed in the  $i$ th iteration of the for loop, and let  $X_i = 0$  otherwise. What is  $\Pr[X_i = 1]$ ? [Hint: First consider  $i = 1$  and  $i = n$ .]
- (c) What is the exact expected number of executions of line  $(\star)$ ?
- (d) **Prove** that line  $(\star)$  is executed  $O(\log n)$  times with high probability, assuming the variables  $X_i$  are mutually independent.
- (e) **[Extra credit]** **Prove** that the variables  $X_i$  are mutually independent.  
[Hint: Finish the rest of the exam first!]

4. Your eight-year-old cousin Elmo decides to teach his favorite new card game to his baby sister Daisy. At the beginning of the game,  $n$  cards are dealt face up in a long row. Each card is worth some number of points, which may be positive, negative, or zero. Then Elmo and Daisy take turns removing either the leftmost or rightmost card from the row, until all the cards are gone. At each turn, each player can decide which of the two cards to take. When the game ends, the player that has collected the most points wins.

Daisy isn't old enough to get this whole "strategy" thing; she's just happy to play with her big brother. When it's her turn, she takes the either leftmost card or the rightmost card, each with probability  $1/2$ .

Elmo, on the other hand, *really* wants to win. Having never taken an algorithms class, he follows the obvious greedy strategy—when it's his turn, Elmo *always* takes the card with the higher point value.

Describe and analyze an algorithm to determine Elmo's expected score, given the initial sequence of  $n$  cards as input. Assume Elmo moves first, and that no two cards have the same value.

For example, suppose the initial cards have values 1, 4, 8, 2. Elmo takes the 2, because it's larger than 1. Then Daisy takes either 1 or 8 with equal probability. If Daisy takes the 1, then Elmo takes the 8; if Daisy takes the 8, then Elmo takes the 4. Thus, Elmo's expected score is  $2 + (8 + 4)/2 = 8$ .

**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

Red text reflects corrections or clarifications given during the actual exam.

---

1. Suppose we insert  $n$  distinct items into an initially empty hash table of size  $m \gg n$ , using an **ideal random** hash function  $h$ . Recall that a collision is a set of two distinct items  $\{x, y\}$  in the table such that  $h(x) = h(y)$ .
  - (a) What is the exact expected number of collisions?
  - (b) Estimate the probability that there are no collisions. [Hint: Use Markov's inequality.]
  - (c) Estimate the **largest** value of  $n$  such that the probability of having no collisions is at least  $1 - 1/n$ . Your answer should have the form  $n = O(f(m))$  for some simple function  $f$ .
  - (d) Fix an integer  $k > 1$ . A  **$k$ -way collision** is a set of  $k$  distinct items  $\{x_1, \dots, x_k\}$  that all have the same hash value:  $h(x_1) = h(x_2) = \dots = h(x_k)$ . Estimate the **largest** value of  $n$  such that the probability of having no  $k$ -way collisions is at least  $1 - 1/n$ . Your answer should have the form  $n = O(f(m, k))$  for some simple function  $f$ . [Hint: You may want to repeat parts (a) and (b).]
2. Quentin, Alice, and the other Brakebills Physical Kids are planning an excursion through the Netherlands to Fillory. The Netherlands is a vast, deserted city composed of several plazas, each containing a single fountain that can magically transport people to a different world. Adjacent plazas are connected by gates, which have been cursed by the Beast. The gates open for only five minutes every hour, all at the same time. During those five minutes, if more than one person passes through any **single** gate, the Beast will detect their presence. □ **However, people can safely pass through different gates at the same time.** Moreover, anyone attempting to pass through more than one gate in the same five-minute period will turn into a niffin. □

You are given a map of the Netherlands, which is a graph  $G$  with a vertex for each fountain and an edge for each gate, with the fountains to Earth and Fillory clearly marked; you are also given a positive integer  $h$ . Describe and analyze an algorithm to compute the maximum number of people that can walk from the Earth fountain to the Fillory fountain in  $h$  hours, without anyone alerting the Beast or turning into a niffin.

---

□This is very bad.  
□This is very bad.

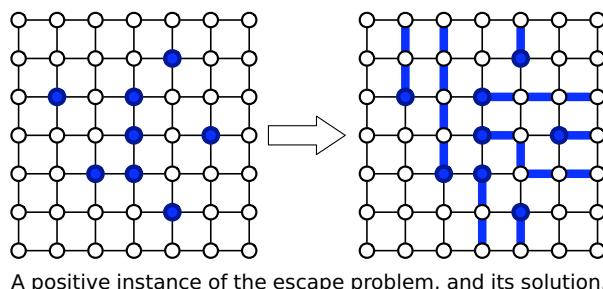
3. Recall that a **Bloom filter** is an array  $B[1..m]$  of bits, together with a collection of  $k$  independent ideal random hash functions  $h_1, h_2, \dots, h_k$ . To insert an item  $x$  into a Bloom filter, we set  $B[h_i(x)] \leftarrow 1$  for every index  $i$ . To test whether an item  $x$  belongs to a set represented by a Bloom filter, we check whether  $B[h_i(x)] = 1$  for every index  $i$ . This algorithm always returns TRUE if  $x$  is in the set, but may return either TRUE or FALSE when  $x$  is not in the set. Thus, there may be false positives, but no false negatives.

If there are  $n$  distinct items stored in the Bloom filter, then the probability of a false positive is  $(1 - p)^k$ , where  $p \approx e^{-kn/m}$  is the probability that  $B[j] = 0$  for any particular index  $j$ . In particular, if we set  $k = (m/n) \ln 2$ , then  $p = 1/2$ , and the probability of a false positive is  $(1/2)^{(m/n)\ln 2} \approx (0.61850)^{m/n}$ .

After months spent lovingly crafting a Bloom filter of size  $m$  for a set  $S$  of  $n$  items, using exactly  $k = (m/n) \ln 2$  hash functions (so  $p = 1/2$ ), your boss tells you that you must reduce the size of your Bloom filter from  $m$  bits down to  $m/2$  bits. Unfortunately, you no longer have the original set  $S$ , and your company's product ships tomorrow; you have to do something quick and dirty. Fortunately, your boss has a couple of ideas.

- (a) First your boss suggests simply discarding half of the Bloom filter, keeping only the subarray  $B[1..m/2]$ . Describe an algorithm to check whether a given item  $x$  is an element of the original set  $S$ , using only this smaller Bloom filter. As usual, if  $x \in S$ , your algorithm **must** return TRUE.
  - (b) What is the probability that your algorithm returns TRUE when  $x \notin S$ ?
  - (c) Next your boss suggests merging the two halves of your old Bloom filter, defining a new array  $B'[1..m/2]$  by setting  $B'[i] \leftarrow B[i] \vee B[i + m/2]$  for all  $i$ . Describe an algorithm to check whether a given item  $x$  is an element of the original set  $S$ , using only this smaller Bloom filter  $B'$ . As usual, if  $x \in S$ , your algorithm **must** return TRUE.
  - (d) What is the probability that your algorithm returns TRUE when  $x \notin S$ ?
4. An  $n \times n$  grid is an undirected graph with  $n^2$  vertices organized into  $n$  rows and  $n$  columns. We denote the vertex in the  $i$ th row and the  $j$ th column by  $(i, j)$ . Every vertex  $(i, j)$  has exactly four neighbors  $(i - 1, j)$ ,  $(i + 1, j)$ ,  $(i, j - 1)$ , and  $(i, j + 1)$ , except the *boundary* vertices, for which  $i = 1$ ,  $i = n$ ,  $j = 1$ , or  $j = n$ .

Let  $(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$  be distinct vertices, called *terminals*, in the  $n \times n$  grid. The **escape problem** is to determine whether there are  $m$  vertex-disjoint paths in the grid that connect these terminals to any  $m$  distinct boundary vertices. Describe and analyze an efficient algorithm to solve the escape problem.



**Write your answers in the separate answer booklet.**

Please return this question handout and your cheat sheets with your answers.

---

1. Let  $G = (V, E)$  be an arbitrary undirected graph. A **triple-Hamiltonian circuit** in  $G$  is a closed walk in  $G$  that visits every vertex of  $G$  exactly *three* times. *Prove* that it is NP-hard to determine whether a given undirected graph has a triple-Hamiltonian circuit. [Hint: Modify your reduction for double-Hamiltonian circuits from Homework 10.]
2. Marie-Joseph Paul Yves Roch Gilbert du Motier, Marquis de Lafayette, colonial America's favorite fighting Frenchman, needs to choose a subset of his ragtag volunteer army of  $m$  soldiers to complete a set of  $n$  important tasks, like "go to France for more funds" or "come back with more guns". Each task requires a specific set of skills, such as "knows what to do in a trench" or "ingenuine and fluent in French". For each task, exactly  $k$  soldiers are qualified to complete that task.

Unfortunately, Lafayette's soldiers are extremely lazy. For each task, if Lafayette chooses more than one soldier qualified for that task, each of them will assume that someone else will take on that task, and so the task will never be completed. A task will be completed if and only if exactly one of the chosen soldiers has the necessary skills for that task.

So Lafayette needs to choose a subset  $S$  of soldiers that maximizes the number of tasks for which *exactly one* soldier in  $S$  is qualified. Not surprisingly, Lafayette's problem is NP-hard.

- (a) Suppose Lafayette chooses each soldier independently with probability  $p$ . What is the *exact* expected number of tasks that will be completed, in terms of  $p$  and  $k$ ?  
(b) What value of  $p$  maximizes this expected value?  
(c) Describe a randomized polynomial-time  $O(1)$ -approximation algorithm for Lafayette's problem. What is the expected approximation ratio for your algorithm?
3. Suppose we are given a set of  $n$  rectangular boxes, each specified by their height, width, and depth in centimeters. All three dimensions of each box lie strictly between 10cm and 20cm, and all  $3n$  dimensions are distinct. As you might expect, one box can be nested inside another if the first box can be rotated so that it is smaller in every dimension than the second box. Boxes can be nested recursively, but two boxes cannot be nested side-by-side inside a third box. A box is *visible* if it is not nested inside another box.

Describe and analyze an algorithm to nest the boxes, so that the number of visible boxes is as small as possible.

4. Hercules Mulligan, a tailor spyin' on the British government, has determined a set of routes and towns that the British army plans to use to move their troops from Charleston, South Carolina to Yorktown, Virginia. (He took their measurements, information, and then he smuggled it.) The American revolutionary army wants to set up ambush points in some of these towns, so that every unit of the British army will face at least one ambush before reaching Yorktown. On the other hand, General Washington wants to leave as many troops available as possible to help defend Yorktown when the British army inevitably arrives.

Describe an efficient algorithm that computes the smallest number of towns where the revolutionary army should set up ambush points. The input to your algorithm is Mulligan's graph of towns (vertices) and routes (edges), with Charleston and Yorktown clearly marked.

5. Consider the following randomized algorithm to approximate the smallest vertex cover in an undirected graph  $G = (V, E)$ . For each vertex  $v \in V$ , define the *priority* of  $v$  to be a real number between 0 and 1, chosen independently and uniformly at random. Finally, let  $S$  be the subset of vertices with higher priority than at least one of their neighbors:

$$S := \left\{ v \in V \mid \text{priority}(v) > \min_{uv \in E} \text{priority}(u) \right\}$$

- (a) What is the probability that the set  $S$  is a vertex cover of  $G$ ? *Prove* your answer is correct. (Your proof should be *short*.)
  - (b) Suppose the input graph  $G$  is a cycle of length  $n$ . What is the *exact* expected size of  $S$ ?
  - (c) Suppose the input graph  $G$  is a *star*: a tree with one vertex of degree  $n - 1$  and  $n - 1$  vertices of degree 1. What is the *exact* probability that  $S$  is the *smallest* vertex cover of  $G$ ?
  - (d) Again, suppose  $G$  is a star. Suppose we run the randomized algorithm  $N$  times, generating a sequence of subsets  $S_1, S_2, \dots, S_N$ . How large must  $N$  be to guarantee with high probability that some  $S_i$  is the minimum vertex cover of  $G$ ?
6. After the Revolutionary War, Alexander Hamilton's biggest rival as a lawyer was Aaron Burr. (Sir!) In fact, the two worked next door to each other. Unlike Hamilton, Burr cannot work non-stop; every case he tries exhausts him. The bigger the case, the longer he must rest before he is well enough to take the next case. (Of course, he is willing to wait for it.) If a case arrives while Burr is resting, Hamilton snatches it up instead.

Burr has been asked to consider a sequence of  $n$  upcoming cases. He quickly computes two arrays  $\text{profit}[1..n]$  and  $\text{skip}[1..n]$ , where for each index  $i$ ,

- $\text{profit}[i]$  is the amount of money Burr would make by taking the  $i$ th case, and
- $\text{skip}[i]$  is the number of consecutive cases Burr must skip if he accepts the  $i$ th case. That is, if Burr accepts the  $i$ th case, he cannot accept cases  $i + 1$  through  $i + \text{skip}[i]$ .

Design and analyze an algorithm that determines the maximum total profit Burr can secure from these  $n$  cases, using his two arrays as input.

## CS/ECE 374 ♦ Fall 2016

## ❖ Homework o ❖

Due Tuesday, August 30, 2016 at 8pm

---

- **Each student must submit individual solutions for this homework.** For all future homeworks, groups of up to three students can submit joint solutions.
  - **Submit your solutions electronically on the course Gradescope site as PDF files.** Submit a separate PDF file for each numbered problem. If you plan to typeset your solutions, please use the L<sup>A</sup>T<sub>E</sub>X solution template on the course web site. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app (or an actual scanner, not just a phone camera).
  - You are *not* required to sign up on Gradescope (or Piazza) with your real name and your illinois.edu email address; you may use any email address and alias of your choice. However, to give you credit for the homework, we need to know who Gradescope thinks you are. **Please fill out the web form linked from the course web page.**
- 

## ❖ Some important course policies ❖

- **You may use any source at your disposal**—paper, electronic, or human—but you *must* cite *every* source that you use, and you *must* write everything yourself in your own words. See the academic integrity policies on the course web site for more details.
- The answer “*I don’t know*” (and *nothing else*) is worth 25% partial credit on any required problem or subproblem, on any homework or exam. We will accept synonyms like “No idea” or “WTF” or “~\(\bullet\_-\)/~”, but you must write *something*.
- **Avoid the Three Deadly Sins!** Any homework or exam solution that breaks any of the following rules will be given an **automatic zero**, unless the solution is otherwise perfect. Yes, we really mean it. We’re not trying to be scary or petty (Honest!), but we do want to break a few common bad habits that seriously impede mastery of the course material.
  - Always give complete solutions, not just examples.
  - Always declare all your variables, in English. In particular, always describe the specific problem your algorithm is supposed to solve.
  - Never use weak induction.

---

**See the course web site for more information.**

If you have any questions about these policies,  
please don’t hesitate to ask in class, in office hours, or on Piazza.

---

1. The famous Czech professor Jiřína Z. Džunglová has a favorite 23-node binary tree, in which each node is labeled with a unique letter of the alphabet. Preorder and inorder traversals of the tree visit the nodes in the following order:

- Preorder: **Y G E P V U B N X I Z L O F J A H R C D S M T**
  - Inorder: **P E U V B G X N I Y F O J L R H D C S A M Z T**
- (a) List the nodes in Professor Džunglová's tree in post-order.  
 (b) Draw Professor Džunglová's tree.

2. The *complement*  $w^c$  of a string  $w \in \{0, 1\}^*$  is obtained from  $w$  by replacing every **0** in  $w$  with a **1** and vice versa; for example, **111011000100**<sup>c</sup> = **000100111011**. The complement function is formally defined as follows:

$$w^c := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ \mathbf{1} \cdot x^c & \text{if } w = \mathbf{0}x \\ \mathbf{0} \cdot x^c & \text{if } w = \mathbf{1}x \end{cases}$$

- (a) Prove by induction that  $|w| = |w^c|$  for every string  $w$ .  
 (b) Prove by induction that  $(x \bullet y)^c = x^c \bullet y^c$  for all strings  $x$  and  $y$ .

Your proofs must be formal and self-contained, and they must invoke the *formal* definitions of length  $|w|$ , concatenation  $x \bullet y$ , and complement  $w^c$ . Do not appeal to intuition!

3. Recursively define a set  $L$  of strings over the alphabet  $\{0, 1\}$  as follows:

- The empty string  $\varepsilon$  is in  $L$ .
- For all strings  $x$  and  $y$  in  $L$ , the string **0** $x**1** $y$  is also in  $L$ .$
- For all strings  $x$  and  $y$  in  $L$ , the string **1** $x**0** $y$  is also in  $L$ .$
- These are the only strings in  $L$ .

Let  $\#(a, w)$  denote the number of times symbol  $a$  appears in string  $w$ ; for example,

$$\#(\mathbf{0}, \mathbf{01000110111001}) = \#(\mathbf{1}, \mathbf{01000110111001}) = 7.$$

- (a) Prove that the string **01000110111001** is in  $L$ .  
 (b) Prove by induction that every string in  $L$  has exactly the same number of **0**s and **1**s.  
 (You may assume without proof that  $\#(a, xy) = \#(a, x) + \#(a, y)$  for any symbol  $a$  and any strings  $x$  and  $y$ .)  
 (c) Prove by induction that  $L$  contains every string with the same number of **0**s and **1**s.

Each homework assignment will include at least one solved problem, similar to the problems assigned in that homework, together with the grading rubric we would apply if this problem appeared on a homework or exam. These model solutions illustrate our recommendations for structure, presentation, and level of detail in your homework solutions. Of course, the actual content of your solutions won't match the model solutions, because your problems are different!

## Solved Problems

4. Recall that the *reversal*  $w^R$  of a string  $w$  is defined recursively as follows:

$$w^R := \begin{cases} \epsilon & \text{if } w = \epsilon \\ x^R \bullet a & \text{if } w = a \cdot x \end{cases}$$

A *palindrome* is any string that is equal to its reversal, like **AMANAPLANACANALPANAMA**, **RACECAR**, **POOP**, **I**, and the empty string.

- (a) Give a recursive definition of a palindrome over the alphabet  $\Sigma$ .
- (b) Prove  $w = w^R$  for every palindrome  $w$  (according to your recursive definition).
- (c) Prove that every string  $w$  such that  $w = w^R$  is a palindrome (according to your recursive definition).

In parts (b) and (c), you may assume without proof that  $(x \cdot y)^R = y^R \bullet x^R$  and  $(x^R)^R = x$  for all strings  $x$  and  $y$ .

### Solution:

- (a) A string  $w \in \Sigma^*$  is a palindrome if and only if either
- $w = \epsilon$ , or
  - $w = a$  for some symbol  $a \in \Sigma$ , or
  - $w = axa$  for some symbol  $a \in \Sigma$  and some *palindrome*  $x \in \Sigma^*$ .

**Rubric:** 2 points =  $\frac{1}{2}$  for each base case + 1 for the recursive case. No credit for the rest of the problem unless this is correct.

- (b) Let  $w$  be an arbitrary palindrome.

Assume that  $x = x^R$  for every palindrome  $x$  such that  $|x| < |w|$ .

There are three cases to consider (mirroring the three cases in the definition):

- If  $w = \epsilon$ , then  $w^R = \epsilon$  by definition, so  $w = w^R$ .
- If  $w = a$  for some symbol  $a \in \Sigma$ , then  $w^R = a$  by definition, so  $w = w^R$ .
- Suppose  $w = axa$  for some symbol  $a \in \Sigma$  and some palindrome  $x \in P$ . Then

$$\begin{aligned} w^R &= (a \cdot x \bullet a)^R \\ &= (x \bullet a)^R \bullet a && \text{by definition of reversal} \\ &= a^R \bullet x^R \bullet a && \text{You said we could assume this.} \\ &= a \bullet x^R \bullet a && \text{by definition of reversal} \\ &= a \bullet x \bullet a && \text{by the inductive hypothesis} \\ &= w && \text{by assumption} \end{aligned}$$

In all three cases, we conclude that  $w = w^R$ .

**Rubric:** 4 points: standard induction rubric (scaled)

- (c) Let  $w$  be an arbitrary string such that  $w = w^R$ .

Assume that every string  $x$  such that  $|x| < |w|$  and  $x = x^R$  is a palindrome.

There are three cases to consider (mirroring the definition of “palindrome”):

- If  $w = \varepsilon$ , then  $w$  is a palindrome by definition.
- If  $w = a$  for some symbol  $a \in \Sigma$ , then  $w$  is a palindrome by definition.
- Otherwise, we have  $w = ax$  for some symbol  $a$  and some non-empty string  $x$ .  
The definition of reversal implies that  $w^R = (ax)^R = x^Ra$ .  
Because  $x$  is non-empty, its reversal  $x^R$  is also non-empty.  
Thus,  $x^R = by$  for some symbol  $b$  and some string  $y$ .  
It follows that  $w^R = bya$ , and therefore  $w = (w^R)^R = (bya)^R = ay^Rb$ .

*[At this point, we need to prove that  $a = b$  and that  $y$  is a palindrome.]*

Our assumption that  $w = w^R$  implies that  $bya = ay^Rb$ .

The recursive definition of string equality immediately implies  $a = b$ .

Because  $a = b$ , we have  $w = ay^Ra$  and  $w^R = aya$ .

The recursive definition of string equality implies  $y^Ra = ya$ .

It immediately follows that  $(y^Ra)^R = (ya)^R$ .

Known properties of reversal imply  $(y^Ra)^R = a(y^R)^R = ay$  and  $(ya)^R = ay^R$ .

It follows that  $ay^R = ay$ , and therefore  $y = y^R$ .

The inductive hypothesis now implies that  $y$  is a palindrome.

We conclude that  $w$  is a palindrome by definition.

In all three cases, we conclude that  $w$  is a palindrome.

**Rubric:** 4 points: standard induction rubric (scaled).

- No penalty for jumping from  $aya = ay^Ra$  directly to  $y = y^R$ .



**Rubric (induction):** For problems worth 10 points:

- + 1 for explicitly considering an *arbitrary* object
- + 2 for a valid **strong** induction hypothesis
  - **Deadly Sin!** Automatic zero for stating a weak induction hypothesis, unless the rest of the proof is *perfect*.
- + 2 for explicit exhaustive case analysis
  - No credit here if the case analysis omits an infinite number of objects. (For example: all odd-length palindromes.)
  - -1 if the case analysis omits a finite number of objects. (For example: the empty string.)
  - -1 for making the reader infer the case conditions. Spell them out!
  - No penalty if cases overlap (for example:)
- + 1 for cases that do not invoke the inductive hypothesis (“base cases”)
  - No credit here if one or more “base cases” are missing.
- + 2 for correctly applying the *stated* inductive hypothesis
  - No credit here for applying a *different* inductive hypothesis, even if that different inductive hypothesis would be valid.
- + 2 for other details in cases that invoke the inductive hypothesis (“inductive cases”)
  - No credit here if one or more “inductive cases” are missing.

## ❧ Homework 1 ❧

Due Tuesday, September 6, 2016 at 8pm

**Starting with this homework, groups of up to three people can submit joint solutions.** Each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the Gradescope names and email addresses of each group member. In addition, whoever submits the homework must tell Gradescope who their other group members are.

1. For each of the following languages over the alphabet  $\{0, 1\}$ , give a regular expression that describes that language, and briefly argue why your expression is correct.
  - (a) All strings that end with the suffix  $01010101$ .
  - (b) All strings except  $111$ .
  - (c) All strings that contain the substring  $010$ .
  - (d) All strings that contain the subsequence  $010$ .
  - (e) All strings that do not contain the substring  $010$ .
  - (f) All strings that do not contain the subsequence  $010$ .
  
  
  
  
  
  
2. This problem considers two special classes of regular expressions.
  - A regular expression  $R$  is **plus-free** if and only if it never uses the  $+$  operator.
  - A regular expression  $R$  is **top-plus** if and only if either
    - $R$  is plus-free, or
    - $R = S + T$ , where  $S$  and  $T$  are top-plus.

For example,  $1((0^*10)^*1)^*0$  is plus-free and (therefore) top-plus;  $01^*0 + 10^*1 + \epsilon$  is top-plus but not plus-free, and  $0(0 + 1)^*(1 + \epsilon)$  is neither top-plus nor plus-free.

Recall that two regular expressions  $R$  and  $S$  are **equivalent** if they describe exactly the same language:  $L(R) = L(S)$ .

- (a) Prove that for any top-plus regular expressions  $R$  and  $S$ , there is a top-plus regular expression that is equivalent to  $RS$ . [Hint: Use the fact that  $(A + B)(C + D)$  and  $AC + AD + BC + BD$  are equivalent, for all regular expressions  $A, B, C$ , and  $D$ .]
- (b) Prove that for any top-plus regular expression  $R$ , there is a **plus-free** regular expression  $S$  such that  $R^*$  and  $S^*$  are equivalent. [Hint: Use the fact that  $(A+B)^*$  is equivalent to  $(A^*B^*)^*$ , for all regular expressions  $A$  and  $B$ .]
- (c) Prove that for any regular expression, there is an equivalent top-plus regular expression.

3. Let  $L$  be the set of all strings in  $\{\text{0}, \text{1}\}^*$  that contain exactly two occurrences of the substring  $\text{001}$ .

- (a) Describe a DFA that over the alphabet  $\Sigma = \{\text{0}, \text{1}\}$  that accepts the language  $L$ . Argue that your machine accepts every string in  $L$  and nothing else, by explaining what each state in your DFA means.

You may either draw the DFA or describe it formally, but the states  $Q$ , the start state  $s$ , the accepting states  $A$ , and the transition function  $\delta$  must be clearly specified.

- (b) Give a regular expression for  $L$ , and briefly argue that why expression is correct.

## Solved problem

4. **C comments** are the set of strings over alphabet  $\Sigma = \{\star, /, A, \diamond, \downarrow\}$  that form a proper comment in the C program language and its descendants, like C++ and Java. Here  $\downarrow$  represents the newline character,  $\diamond$  represents any other whitespace character (like the space and tab characters), and  $A$  represents any non-whitespace character other than  $\star$  or  $/$ . □ There are two types of C comments:

- Line comments: Strings of the form  $/* \dots \downarrow$ .
- Block comments: Strings of the form  $/* \dots */$ .

Following the C99 standard, we explicitly disallow **nesting** comments of the same type. A line comment starts with  $/*$  and ends at the first  $\downarrow$  after the opening  $/*$ . A block comment starts with  $/*$  and ends at the the first  $*/$  completely after the opening  $/*$ ; in particular, every block comment has at least two  $*$ s. For example, each of the following strings is a valid C comment:

- $/* */$
- $/* \diamond /* \diamond \downarrow$
- $/* /* \diamond \star \diamond \downarrow */$
- $/* \diamond /* \downarrow \diamond */$

On the other hand, *none* of the following strings is a valid C comments:

- $/* /$
- $/* \diamond /* \downarrow \diamond \downarrow$
- $/* \diamond /* \star \diamond /* \diamond */$

- (a) Describe a DFA that accepts the set of all C comments.
- (b) Describe a DFA that accepts the set of all strings composed entirely of blanks ( $\diamond$ ), newlines ( $\downarrow$ ), and C comments.

**You must explain in English how your DFAs work.** Drawings or formal descriptions without English explanations will receive no credit, even if they are correct.

---

□The actual C commenting syntax is considerably more complex than described here, because of character and string literals.

- The opening  $/*$  or  $/*$  of a comment must not be inside a string literal ("...") or a (multi-)character literal ('...').
- The opening double-quote of a string literal must not be inside a character literal ('") or a comment.
- The closing double-quote of a string literal must not be escaped (\")
- The opening single-quote of a character literal must not be inside a string literal ("... ' ...") or a comment.
- The closing single-quote of a character literal must not be escaped (\')
- A backslash escapes the next symbol if and only if it is not itself escaped (\\\) or inside a comment.

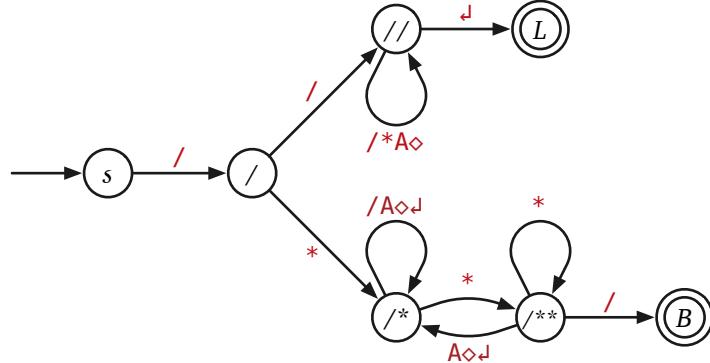
For example, the string  $/* \\ */ /* /* /* /* */$  is a valid string literal (representing the 5-character string  $/* \star /$ , which is itself a valid block comment!) followed immediately by a valid block comment. **For this homework question, just pretend that the characters ', ", and \ don't exist.**

Commenting in C++ is even more complicated, thanks to the addition of raw string literals. Don't ask.

Some C and C++ compilers do support nested block comments, in violation of the language specification. A few other languages, like OCaml, explicitly allow nesting block comments.

**Solution:**

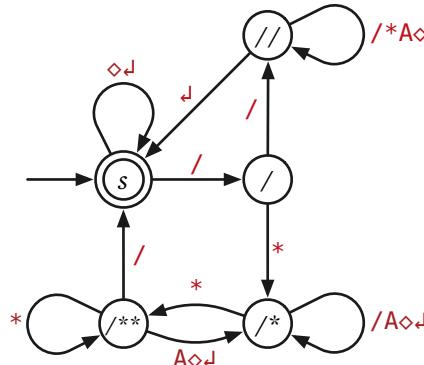
- (a) The following eight-state DFA recognizes the language of C comments. All missing transitions lead to a hidden reject state.



The states are labeled mnemonically as follows:

- $s$  — We have not read anything.
- $/$  — We just read the initial  $/$ .
- $//$  — We are reading a line comment.
- $L$  — We have read a complete line comment.
- $/*$  — We are reading a block comment, and we did not just read a  $*$  after the opening  $/*$ .
- $/**$  — We are reading a block comment, and we just read a  $*$  after the opening  $/*$ .
- $B$  — We have read a complete block comment.

- (b) By merging the accepting states of the previous DFA with the start state and adding white-space transitions at the start state, we obtain the following six-state DFA. Again, all missing transitions lead to a hidden reject state.



The states are labeled mnemonically as follows:

- $s$  — We are between comments.
- $/$  — We just read the initial  $/$  of a comment.
- $//$  — We are reading a line comment.

- `/*` — We are reading a block comment, and we did not just read a `*` after the opening `/*`.
- `/**` — We are reading a block comment, and we just read a `*` after the opening `/*`. ■

**Rubric:** 10 points = 5 for each part, using the standard DFA design rubric (scaled)

**Rubric (DFA design):** For problems worth 10 points:

- 2 points for an unambiguous description of a DFA, including the states set  $Q$ , the start state  $s$ , the accepting states  $A$ , and the transition function  $\delta$ .
  - **For drawings:** Use an arrow from nowhere to indicate  $s$ , and doubled circles to indicate accepting states  $A$ . If  $A = \emptyset$ , say so explicitly. If your drawing omits a reject state, say so explicitly. **Draw neatly!** If we can't read your solution, we can't give you credit for it.,
  - **For text descriptions:** You can describe the transition function either using a 2d array, using mathematical notation, or using an algorithm.
  - **For product constructions:** You must give a complete description of the states and transition functions of the DFAs you are combining (as either drawings or text), together with the accepting states of the product DFA.
- **Homework only:** 4 points for briefly and correctly explaining the purpose of each state *in English*. This is how you justify that your DFA is correct.
  - For product constructions, explaining the states in the factor DFAs is enough.
  - **Deadly Sin:** (“Declare your variables.”) No credit for the problem if the English description is missing, even if the DFA is correct.
- 4 points for correctness. (8 points on exams, with all penalties doubled)
  - −1 for a single mistake: a single misdirected transition, a single missing or extra accept state, rejecting exactly one string that should be accepted, or accepting exactly one string that should be accepted.
  - −2 for incorrectly accepting/rejecting more than one but a finite number of strings.
  - −4 for incorrectly accepting/rejecting an infinite number of strings.
- DFA drawings with too many states may be penalized. DFA drawings with significantly too many states may get no credit at all.
- Half credit for describing an NFA when the problem asks for a DFA.

**❖ Homework 2 ❖**

Due Tuesday, September 13, 2016 at 8pm

1. A **Moore machine** is a variant of a finite-state automaton that produces output; Moore machines are sometimes called finite-state *transducers*. For purposes of this problem, a Moore machine formally consists of six components:

- A finite set  $\Sigma$  called the input alphabet
- A finite set  $\Gamma$  called the output alphabet
- A finite set  $Q$  whose elements are called states
- A start state  $s \in Q$
- A transition function  $\delta: Q \times \Sigma \rightarrow Q$
- An output function  $\omega: Q \rightarrow \Gamma$

More intuitively, a Moore machine is a graph with a special start vertex, where every node (state) has one outgoing edge labeled with each symbol from the input alphabet, and each node (state) is additionally labeled with a symbol from the output alphabet.

The Moore machine reads an input string  $w \in \Sigma^*$  one symbol at a time. For each symbol, the machine changes its state according to the transition function  $\delta$ , and then outputs the symbol  $\omega(q)$ , where  $q$  is the new state. Formally, we recursively define a *transducer* function  $\omega^*: Q \times \Sigma^* \rightarrow \Gamma^*$  as follows:

$$\omega^*(q, w) = \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ \omega(\delta(q, a)) \cdot \omega^*(\delta(q, a), x) & \text{if } w = ax \end{cases}$$

Given input string  $w \in \Sigma^*$ , the machine outputs the string  $\omega^*(w, s) \in \Gamma^*$ . The **output language**  $L^\circ(M)$  of a Moore machine  $M$  is the set of all strings that the machine can output:

$$L^\circ(M) := \{\omega^*(s, w) \mid w \in \Sigma^*\}$$

- (a) Let  $M$  be an arbitrary Moore machine. Prove that  $L^\circ(M)$  is a regular language.  
 (b) Let  $M$  be an arbitrary Moore machine whose input alphabet  $\Sigma$  and output alphabet  $\Gamma$  are identical. Prove that the language

$$L^=(M) = \{w \in \Sigma^* \mid w = \omega^*(s, w)\}$$

is regular.  $L^=(M)$  consists of all strings  $w$  such that  $M$  outputs  $w$  when given input  $w$ ; these are also called *fixed points* for the transducer function  $\omega^*$ .

*[Hint: These problems are easier than they look!]*

2. Prove that the following languages are *not* regular.

- (a)  $\{w \in (\textcolor{red}{0} + \textcolor{red}{1})^* \mid |\#(\textcolor{red}{0}, w) - \#(\textcolor{red}{1}, w)| < 5\}$
- (b) Strings in  $(\textcolor{red}{0} + \textcolor{red}{1})^*$  in which the substrings  $\textcolor{red}{00}$  and  $\textcolor{red}{11}$  appear the same number of times.
- (c)  $\{\textcolor{red}{0}^m \textcolor{red}{1} \textcolor{red}{0}^n \mid n/m \text{ is an integer}\}$

3. Let  $L$  be an arbitrary regular language.

- (a) Prove that the language  $palin(L) := \{w \mid ww^R \in L\}$  is also regular.
- (b) Prove that the language  $drome(L) := \{w \mid w^R w \in L\}$  is also regular.

## Solved problem

4. Let  $L$  be an arbitrary regular language. Prove that the language  $\text{half}(L) := \{w \mid ww \in L\}$  is also regular.

**Solution:** Let  $M = (\Sigma, Q, s, A, \delta)$  be an arbitrary DFA that accepts  $L$ . We define a new NFA  $M' = (\Sigma, Q', s', A', \delta')$  with  $\epsilon$ -transitions that accepts  $\text{half}(L)$ , as follows:

$$\begin{aligned} Q' &= (Q \times Q \times Q) \cup \{s'\} \\ s' &\text{ is an explicit state in } Q' \\ A' &= \{(h, h, q) \mid h \in Q \text{ and } q \in A\} \\ \delta'(s', \epsilon) &= \{(s, h, h) \mid h \in Q\} \\ \delta'((p, h, q), a) &= \{(\delta(p, a), h, \delta(q, a))\} \end{aligned}$$

$M'$  reads its input string  $w$  and simulates  $M$  reading the input string  $ww$ . Specifically,  $M'$  simultaneously simulates two copies of  $M$ , one reading the left half of  $ww$  starting at the usual start state  $s$ , and the other reading the right half of  $ww$  starting at some intermediate state  $h$ .

- The new start state  $s'$  non-deterministically guesses the “halfway” state  $h = \delta^*(s, w)$  without reading any input; this is the only non-determinism in  $M'$ .
- State  $(p, h, q)$  means the following:
  - The left copy of  $M$  (which started at state  $s$ ) is now in state  $p$ .
  - The initial guess for the halfway state is  $h$ .
  - The right copy of  $M$  (which started at state  $h$ ) is now in state  $q$ .
- $M'$  accepts if and only if the left copy of  $M$  ends at state  $h$  (so the initial non-deterministic guess  $h = \delta^*(s, w)$  was correct) and the right copy of  $M$  ends in an accepting state. ■

**Rubric:** 5 points =

- + 1 for a formal, complete, and unambiguous description of a DFA or NFA
  - No points for the rest of the problem if this is missing.
- + 3 for a correct NFA
  - -1 for a single mistake in the description (for example a typo)
- + 1 for a brief English justification. We explicitly do *not* want a formal proof of correctness, but we do want one or two sentences explaining how the NFA works.

## ∞ Homework 3 ∞

Due Tuesday, September 20, 2016 at 8pm

**Groups of up to three people can submit joint solutions.** Each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the Gradescope names and email addresses of each group member. In addition, whoever submits the homework must tell Gradescope who their other group members are.

1. For each of the following regular expressions, describe or draw two finite-state machines:
  - An NFA that accepts the same language, obtained using Thompson's recursive algorithm
  - An equivalent DFA, obtained using the incremental subset construction. For each state in your DFA, identify the corresponding subset of states in your NFA. Your DFA should have no unreachable states.

(a)  $(\text{00} + \text{11})^*(\text{0} + \text{1} + \varepsilon)$

(b)  $\text{1}^* + (\text{01})^* + (\text{001})^*$
  
2. Give context-free grammars for the following languages, and clearly explain how they work and the role of each nonterminal. Grammars can be very difficult to understand; if the grader does not understand how your construction is intended to generate the language, then you will receive no credit.
  - (a) In any string, a *block* (also called a *run*) is a maximal non-empty substring of identical symbols. For example, the string  $0111000011001$  has six blocks: three blocks of 0s of lengths 1, 4, and 2, and three blocks of 1s of lengths 3, 2, and 1.  
Let  $L$  be the set of all strings in  $\{\text{0}, \text{1}\}^*$  that contain two blocks of 0s of equal length. For example,  $L$  contains the strings  $01101111$  and  $01001011100010$  but does not contain the strings  $000110011011$  and  $000000001111$ .
 

(b)  $L = \{w \in \{\text{0}, \text{1}\}^* \mid w \text{ is not a palindrome}\}$ .
  
  3. Let  $L = \{\text{0}^i \text{1}^j \text{2}^k \mid k = i + j\}$ .
    - (a) Show that  $L$  is context-free by describing a grammar for  $L$ .
    - (b) Prove that your grammar  $G$  is correct. As usual, you need to prove both  $L \subseteq L(G)$  and  $L(G) \subseteq L$ .

## Solved problem

4. Let  $L$  be the set of all strings over  $\{0, 1\}^*$  with exactly twice as many 0s as 1s.

- (a) Describe a CFG for the language  $L$ .

[Hint: For any string  $u$  define  $\Delta(u) = \#(0, u) - 2\#(1, u)$ . Introduce intermediate variables that derive strings with  $\Delta(u) = 1$  and  $\Delta(u) = -1$  and use them to define a non-terminal that generates  $L$ .]

**Solution:**  $S \rightarrow \epsilon \mid SS \mid 00S1 \mid 0S1S0 \mid 1S00$  ■

- (b) Prove that your grammar  $G$  is correct. As usual, you need to prove both  $L \subseteq L(G)$  and  $L(G) \subseteq L$ .

[Hint: Let  $u_{\leq i}$  denote the prefix of  $u$  of length  $i$ . If  $\Delta(u) = 1$ , what can you say about the smallest  $i$  for which  $\Delta(u_{\leq i}) = 1$ ? How does  $u$  split up at that position? If  $\Delta(u) = -1$ , what can you say about the smallest  $i$  such that  $\Delta(u_{\leq i}) = -1$ ?]

**Solution:** (Hopefully you recognized this as a more advanced version of HWo problem 3.) We separately prove  $L \subseteq L(G)$  and  $L(G) \subseteq L$  as follows:

**Claim 1.**  $L(G) \subseteq L$ , that is, every string in  $L(G)$  has exactly twice as many 0s as 1s.

**Proof:** As suggested by the hint, for any string  $u$ , let  $\Delta(u) = \#(0, u) - 2\#(1, u)$ . We need to prove that  $\Delta(w) = 0$  for every string  $w \in L(G)$ .

Let  $w$  be an arbitrary string in  $L(G)$ , and consider an arbitrary derivation of  $w$  of length  $k$ . Assume that  $\Delta(x) = 0$  for every string  $x \in L(G)$  that can be derived with fewer than  $k$  productions. □ There are five cases to consider, depending on the first production in the derivation of  $w$ .

- If  $w = \epsilon$ , then  $\#(0, w) = \#(1, w) = 0$  by definition, so  $\Delta(w) = 0$ .
- Suppose the derivation begins  $S \rightsquigarrow SS \rightsquigarrow^* w$ . Then  $w = xy$  for some strings  $x, y \in L(G)$ , each of which can be derived with fewer than  $k$  productions. The inductive hypothesis implies  $\Delta(x) = \Delta(y) = 0$ . It immediately follows that  $\Delta(w) = 0$ . □
- Suppose the derivation begins  $S \rightsquigarrow 00S1 \rightsquigarrow^* w$ . Then  $w = 00x1$  for some string  $x \in L(G)$ . The inductive hypothesis implies  $\Delta(x) = 0$ . It immediately follows that  $\Delta(w) = 0$ .
- Suppose the derivation begins  $S \rightsquigarrow 1S00 \rightsquigarrow^* w$ . Then  $w = 1x00$  for some string  $x \in L(G)$ . The inductive hypothesis implies  $\Delta(x) = 0$ . It immediately follows that  $\Delta(w) = 0$ .
- Suppose the derivation begins  $S \rightsquigarrow 0S1S1 \rightsquigarrow^* w$ . Then  $w = 0x1y0$  for some strings  $x, y \in L(G)$ . The inductive hypothesis implies  $\Delta(x) = \Delta(y) = 0$ . It immediately follows that  $\Delta(w) = 0$ .

In all cases, we conclude that  $\Delta(w) = 0$ , as required. □

□Alternatively: Consider the shortest derivation of  $w$ , and assume  $\Delta(x) = 0$  for every string  $x \in L(G)$  such that  $|x| < |w|$ .

□Alternatively: Suppose the shortest derivation of  $w$  begins  $S \rightsquigarrow SS \rightsquigarrow^* w$ . Then  $w = xy$  for some strings  $x, y \in L(G)$ . Neither  $x$  or  $y$  can be empty, because otherwise we could shorten the derivation of  $w$ . Thus,  $x$  and  $y$  are both shorter than  $w$ , so the induction hypothesis implies . . . We need some way to deal with the decompositions  $w = \epsilon \bullet w$  and  $w = w \bullet \epsilon$ , which are both consistent with the production  $S \rightarrow SS$ , without falling into an infinite loop.

**Claim 2.**  $L \subseteq L(G)$ ; that is,  $G$  generates every binary string with exactly twice as many **0s** as **1s**.

**Proof:** As suggested by the hint, for any string  $u$ , let  $\Delta(u) = \#(\text{0}, u) - 2\#(\text{1}, u)$ . For any string  $u$  and any integer  $0 \leq i \leq |u|$ , let  $u_i$  denote the  $i$ th symbol in  $u$ , and let  $u_{\leq i}$  denote the prefix of  $u$  of length  $i$ .

Let  $w$  be an arbitrary binary string with twice as many **0s** as **1s**. Assume that  $G$  generates every binary string  $x$  that is shorter than  $w$  and has twice as many **0s** as **1s**. There are two cases to consider:

- If  $w = \epsilon$ , then  $\epsilon \in L(G)$  because of the production  $S \rightarrow \epsilon$ .
- Suppose  $w$  is non-empty. To simplify notation, let  $\Delta_i = \Delta(w_{\leq i})$  for every index  $i$ , and observe that  $\Delta_0 = \Delta_{|w|} = 0$ . There are several subcases to consider:
  - Suppose  $\Delta_i = 0$  for some index  $0 < i < |w|$ . Then we can write  $w = xy$ , where  $x$  and  $y$  are non-empty strings with  $\Delta(x) = \Delta(y) = 0$ . The induction hypothesis implies that  $x, y \in L(G)$ , and thus the production rule  $S \rightarrow SS$  implies that  $w \in L(G)$ .
  - Suppose  $\Delta_i > 0$  for all  $0 < i < |w|$ . Then  $w$  must begin with **00**, since otherwise  $\Delta_1 = -2$  or  $\Delta_2 = -1$ , and the last symbol in  $w$  must be **1**, since otherwise  $\Delta_{|w|-1} = -1$ . Thus, we can write  $w = \text{00}x\text{1}$  for some binary string  $x$ . We easily observe that  $\Delta(x) = 0$ , so the induction hypothesis implies  $x \in L(G)$ , and thus the production rule  $S \rightarrow \text{00}S\text{1}$  implies  $w \in L(G)$ .
  - Suppose  $\Delta_i < 0$  for all  $0 < i < |w|$ . A symmetric argument to the previous case implies  $w = \text{1}x\text{00}$  for some binary string  $x$  with  $\Delta(x) = 0$ . The induction hypothesis implies  $x \in L(G)$ , and thus the production rule  $S \rightarrow \text{1}S\text{00}$  implies  $w \in L(G)$ .
  - Finally, suppose none of the previous cases applies:  $\Delta_i < 0$  and  $\Delta_j > 0$  for some indices  $i$  and  $j$ , but  $\Delta_i \neq 0$  for all  $0 < i < |w|$ .

Let  $i$  be the smallest index such that  $\Delta_i < 0$ . Because  $\Delta_j$  either increases by 1 or decreases by 2 when we increment  $j$ , for all indices  $0 < j < |w|$ , we must have  $\Delta_j > 0$  if  $j < i$  and  $\Delta_j < 0$  if  $j \geq i$ .

In other words, there is a *unique* index  $i$  such that  $\Delta_{i-1} > 0$  and  $\Delta_i < 0$ . In particular, we have  $\Delta_1 > 0$  and  $\Delta_{|w|-1} < 0$ . Thus, we can write  $w = \text{0}x\text{1}y\text{0}$  for some binary strings  $x$  and  $y$ , where  $|\text{0}x\text{1}| = i$ .

We easily observe that  $\Delta(x) = \Delta(y) = 0$ , so the inductive hypothesis implies  $x, y \in L(G)$ , and thus the production rule  $S \rightarrow \text{0}S\text{1}S\text{0}$  implies  $w \in L(G)$ .

In all cases, we conclude that  $G$  generates  $w$ . □

Together, Claim 1 and Claim 2 imply  $L = L(G)$ . ■

**Rubric:** 10 points:

- part (a) = 4 points. As usual, this is not the only correct grammar.
- part (b) = 6 points = 3 points for  $\subseteq$  + 3 points for  $\supseteq$ , each using the standard induction template (scaled).

CS/ECE 374 ♦ Fall 2016

## ❀ Homework 4 ❀

Due Tuesday, October 4, 2016 at 8pm

1. Consider the following restricted variant of the Tower of Hanoi puzzle. The pegs are numbered 0, 1, and 2, and your task is to move a stack of  $n$  disks from peg 1 to peg 2. However, you are forbidden to move any disk *directly* between peg 1 and peg 2; *every* move must involve peg 0.

Describe an algorithm to solve this version of the puzzle in as few moves as possible.  
Exactly how many moves does your algorithm make?

2. Consider the following cruel and unusual sorting algorithm.

```

CRUEL(A[1..n]):
  if n > 1
    CRUEL(A[1..n/2])
    CRUEL(A[n/2+1..n])
    UNUSUAL(A[1..n])
  
```

Notice that the comparisons performed by the algorithm do not depend at all on the values in the input array; such a sorting algorithm is called *oblivious*. Assume for this problem that the input size  $n$  is always a power of 2.

- (a) Prove by induction that CRUEL correctly sorts any input array. [Hint: Consider an array that contains  $n/4$  1s,  $n/4$  2s,  $n/4$  3s, and  $n/4$  4s. Why is this special case enough? What does UNUSUAL actually do?]
  - (b) Prove that CRUEL would *not* correctly sort if we removed the for-loop from UNUSUAL.
  - (c) Prove that CRUEL would *not* correctly sort if we swapped the last two lines of UNUSUAL.
  - (d) What is the running time of UNUSUAL? Justify your answer.
  - (e) What is the running time of CRUEL? Justify your answer.

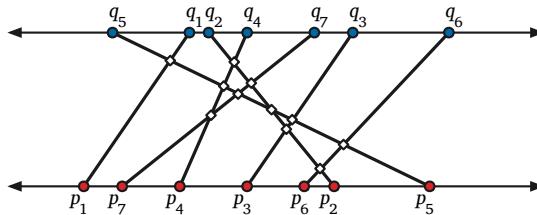
3. You are a visitor at a political convention (or perhaps a faculty meeting) with  $n$  delegates. Each delegate is a member of exactly one political party. It is impossible to tell which political party any delegate belongs to. In particular, you will be summarily ejected from the convention if you ask. However, you *can* determine whether any pair of delegates belong to the *same* party or not simply by introducing them to each other. Members of the same party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.
- (a) Suppose more than half of the delegates belong to the same political party. Describe and analyze an efficient algorithm that identifies every member of this majority party.
  - (b) Now suppose precisely  $p$  political parties are present and one party has a plurality: more delegates belong to that party than to any other party. Please present a procedure to pick out the people from the plurality party as parsimoniously as possible.  $\square$  Do not assume that  $p = O(1)$ .

---

$\square$ Describe and analyze an efficient algorithm that identifies every member of the plurality party.

## Solved Problem

4. Suppose we are given two sets of  $n$  points, one set  $\{p_1, p_2, \dots, p_n\}$  on the line  $y = 0$  and the other set  $\{q_1, q_2, \dots, q_n\}$  on the line  $y = 1$ . Consider the  $n$  line segments connecting each point  $p_i$  to the corresponding point  $q_i$ . Describe and analyze a divide-and-conquer algorithm to determine how many pairs of these line segments intersect, in  $O(n \log n)$  time. See the example below.



Seven segments with endpoints on parallel lines, with 11 intersecting pairs.

Your input consists of two arrays  $P[1..n]$  and  $Q[1..n]$  of  $x$ -coordinates; you may assume that all  $2n$  of these numbers are distinct. No proof of correctness is necessary, but you should justify the running time.

**Solution:** We begin by sorting the array  $P[1..n]$  and permuting the array  $Q[1..n]$  to maintain correspondence between endpoints, in  $O(n \log n)$  time. Then for any indices  $i < j$ , segments  $i$  and  $j$  intersect if and only if  $Q[i] > Q[j]$ . Thus, our goal is to compute the number of pairs of indices  $i < j$  such that  $Q[i] > Q[j]$ . Such a pair is called an *inversion*.

We count the number of inversions in  $Q$  using the following extension of mergesort; as a side effect, this algorithm also sorts  $Q$ . If  $n < 100$ , we use brute force in  $O(1)$  time. Otherwise:

- Recursively count inversions in (and sort)  $Q[1..\lfloor n/2 \rfloor]$ .
- Recursively count inversions in (and sort)  $Q[\lfloor n/2 \rfloor + 1..n]$ .
- Count inversions  $Q[i] > Q[j]$  where  $i \leq \lfloor n/2 \rfloor$  and  $j > \lfloor n/2 \rfloor$  as follows:
  - Color the elements in the Left half  $Q[1..\lfloor n/2 \rfloor]$  **Blue**.
  - Color the elements in the Right half  $Q[\lfloor n/2 \rfloor + 1..n]$  **Red**.
  - Merge  $Q[1..\lfloor n/2 \rfloor]$  and  $Q[\lfloor n/2 \rfloor + 1..n]$ , maintaining their colors.
  - For each **blue** element  $Q[i]$ , count the number of smaller **red** elements  $Q[j]$ .

The last substep can be performed in  $O(n)$  time using a simple for-loop:

```
COUNTREDBLUE(A[1..n]):
  count ← 0
  total ← 0
  for i ← 1 to n
    if A[i] is red
      count ← count + 1
    else
      total ← total + count
  return total
```

In fact, we can execute the third merge-and-count step directly by modifying the MERGE algorithm, without any need for “colors”. Here changes to the standard MERGE algorithm are indicated in red.

```
MERGEANDCOUNT( $A[1..n], m$ ):
   $i \leftarrow 1; j \leftarrow m + 1; count \leftarrow 0; total \leftarrow 0$ 
  for  $k \leftarrow 1$  to  $n$ 
    if  $j > n$ 
       $B[k] \leftarrow A[i]; i \leftarrow i + 1; total \leftarrow total + count$ 
    else if  $i > m$ 
       $B[k] \leftarrow A[j]; j \leftarrow j + 1; count \leftarrow count + 1$ 
    else if  $A[i] < A[j]$ 
       $B[k] \leftarrow A[i]; i \leftarrow i + 1; total \leftarrow total + count$ 
    else
       $B[k] \leftarrow A[j]; j \leftarrow j + 1; count \leftarrow count + 1$ 
  for  $k \leftarrow 1$  to  $n$ 
     $A[k] \leftarrow B[k]$ 
  return  $total$ 
```

We can further optimize this algorithm by observing that  $count$  is always equal to  $j - m - 1$ . (Proof: Initially,  $j = m + 1$  and  $count = 0$ , and we always increment  $j$  and  $count$  together.)

```
MERGEANDCOUNT2( $A[1..n], m$ ):
   $i \leftarrow 1; j \leftarrow m + 1; total \leftarrow 0$ 
  for  $k \leftarrow 1$  to  $n$ 
    if  $j > n$ 
       $B[k] \leftarrow A[i]; i \leftarrow i + 1; total \leftarrow total + j - m - 1$ 
    else if  $i > m$ 
       $B[k] \leftarrow A[j]; j \leftarrow j + 1$ 
    else if  $A[i] < A[j]$ 
       $B[k] \leftarrow A[i]; i \leftarrow i + 1; total \leftarrow total + j - m - 1$ 
    else
       $B[k] \leftarrow A[j]; j \leftarrow j + 1$ 
  for  $k \leftarrow 1$  to  $n$ 
     $A[k] \leftarrow B[k]$ 
  return  $total$ 
```

The modified MERGE algorithm still runs in  $O(n)$  time, so the running time of the resulting modified mergesort still obeys the recurrence  $T(n) = 2T(n/2) + O(n)$ . We conclude that the overall running time is  $O(n \log n)$ , as required. ■

**Rubric:** 10 points = 2 for base case + 3 for divide (split and recurse) + 3 for conquer (merge and count) + 2 for time analysis. Max 3 points for a correct  $O(n^2)$ -time algorithm. This is neither the only way to correctly describe this algorithm nor the only correct  $O(n \log n)$ -time algorithm. No proof of correctness is required.

## ∞ Homework 5 ∞

Due Tuesday, October 11, 2016 at 8pm

1. For each of the following problems, the input consists of two arrays  $X[1..k]$  and  $Y[1..n]$  where  $k \leq n$ .
  - (a) Describe and analyze an algorithm to determine whether  $X$  occurs as two *disjoint* subsequences of  $Y$ , where “disjoint” means the two subsequences have no indices in common. For example, the string **PPAP** appears as two disjoint subsequences in the string **PENPINEAPPLEAPPLEPEN**, but the string **PEOPLE** does not.
  - (b) Describe and analyze an algorithm to compute the number of occurrences of  $X$  as a subsequence of  $Y$ . For example, the string **PPAP** appears exactly 23 times as a subsequence of the string **PENPINEAPPLEAPPLEPEN**. If all characters in  $X$  and  $Y$  are equal, your algorithm should return  $\binom{n}{k}$ . For purposes of analysis, assume that each arithmetic operation takes  $O(1)$  time.
2. You are driving a bus along a long straight highway, full of rowdy, hyper, thirsty students and an endless supply of soda. Each minute that each student is on your bus, that student drinks one ounce of soda. Your goal is to drive all students home, so that the total volume of soda consumed by the students is as small as possible.

Your bus begins at an exit (probably not at either end) with all students on board and moves at a constant speed of 37.4 miles per hour. Each student needs to be dropped off at a highway exit. You may reverse directions as often as you like; for example, you are allowed to drive forward to the next exit, let some students off, then turn around and drive back to the previous exit, drop more students off, then turn around again and drive to further exits. (Assume that at each exit, you can stop the bus, drop off students, and if necessary turn around, all instantaneously.)

Describe an efficient algorithm to take the students home so that they drink as little soda as possible. Your algorithm will be given the following input:

- A sorted array  $L[1..n]$ , where  $L[i]$  is the *Location* of the  $i$ th exit, measured in miles from the first exit; in particular,  $L[1] = 0$ .
- An array  $N[1..n]$ , where  $N[i]$  is the *Number* of students you need to drop off at the  $i$ th exit
- An integer *start* equal to the index of the starting exit.

Your algorithm should return the total volume of soda consumed by the students when you drive the optimal route.□

□Non-US students are welcome to assume kilometers and liters instead of miles and ounces. Late 18th-century French students are welcome to use decimal minutes.

3. *Vankin's Mile* is an American solitaire game played on an  $n \times n$  square grid. The player starts by placing a token on any square of the grid. Then on each turn, the player moves the token either one square to the right or one square down. The game ends when player moves the token off the edge of the board. Each square of the grid has a numerical value, which could be positive, negative, or zero. The player starts with a score of zero; whenever the token lands on a square, the player adds its value to his score. The object of the game is to score as many points as possible.

For example, given the grid below, the player can score  $8 - 6 + 7 - 3 + 4 = 10$  points by placing the initial token on the 8 in the second row, and then moving down, down, right, down, down. (This is *not* the best possible score for these values.)

-1	7	-8	10	-5
-4	-9	8	-6	0
5	-2	-6	-6	7
-7	4	7	-3	-3
7	1	-6	4	-9

- (a) Describe and analyze an efficient algorithm to compute the maximum possible score for a game of Vankin's Mile, given the  $n \times n$  array of values as input.
- (b) In the Canadian version of this game, appropriately called *Vankin's Kilometer*, the player can move the token either one square down, one square right, or one square left in each turn. However, to prevent infinite scores, the token cannot land on the same square more than once. Describe and analyze an efficient algorithm to compute the maximum possible score for a game of Vankin's Kilometer, given the  $n \times n$  array of values as input.  $\square$

---

$\square$ If we also allowed upward movement, the resulting game (*Vankin's Fathom?*) would be NP-hard.

## Solved Problem

4. A *shuffle* of two strings  $X$  and  $Y$  is formed by interspersing the characters into a new string, keeping the characters of  $X$  and  $Y$  in the same order. For example, the string **BANANAANANAS** is a shuffle of the strings **BANANA** and **ANANAS** in several different ways.

**BANANAANANAS**      **BANANAANANAS**      **BANANAANANAS**

Similarly, the strings **PRODGYRNAMAMMIINC** and **DYPRONGARMAMMAGIC** are both shuffles of **DYNAMIC** and **PROGRAMMING**:

**PRODGYRNAMAMMIINC**      **DYPRONGARMAMMAGIC**

Given three strings  $A[1..m]$ ,  $B[1..n]$ , and  $C[1..m+n]$ , describe and analyze an algorithm to determine whether  $C$  is a shuffle of  $A$  and  $B$ .

**Solution:** We define a boolean function  $Shuf(i, j)$ , which is TRUE if and only if the prefix  $C[1..i+j]$  is a shuffle of the prefixes  $A[1..i]$  and  $B[1..j]$ . This function satisfies the following recurrence:

$$Shuf(i, j) = \begin{cases} \text{TRUE} & \text{if } i = j = 0 \\ Shuf(0, j-1) \wedge (B[j] = C[j]) & \text{if } i = 0 \text{ and } j > 0 \\ Shuf(i-1, 0) \wedge (A[i] = C[i]) & \text{if } i > 0 \text{ and } j = 0 \\ (Shuf(i-1, j) \wedge (A[i] = C[i+j])) \\ \vee (Shuf(i, j-1) \wedge (B[j] = C[i+j])) & \text{if } i > 0 \text{ and } j > 0 \end{cases}$$

We need to compute  $Shuf(m, n)$ .

We can memoize all function values into a two-dimensional array  $Shuf[0..m][0..n]$ . Each array entry  $Shuf[i, j]$  depends only on the entries immediately below and immediately to the right:  $Shuf[i-1, j]$  and  $Shuf[i, j-1]$ . Thus, we can fill the array in standard row-major order. The original recurrence gives us the following pseudocode:

```

SHUFFLE?(A[1..m], B[1..n], C[1..m+n]):
    Shuf[0, 0] ← TRUE
    for j ← 1 to n
        Shuf[0, j] ← Shuf[0, j-1] ∧ (B[j] = C[j])
    for i ← 1 to n
        Shuf[i, 0] ← Shuf[i-1, 0] ∧ (A[i] = B[i])
        for j ← 1 to n
            Shuf[i, j] ← FALSE
            if A[i] = C[i+j]
                Shuf[i, j] ← Shuf[i, j] ∨ Shuf[i-1, j]
            if B[i] = C[i+j]
                Shuf[i, j] ← Shuf[i, j] ∨ Shuf[i, j-1]
    return Shuf[m, n]

```

The algorithm runs in  $O(mn)$  time. ■

**Rubric:** Max 10 points: Standard dynamic programming rubric. No proofs required.  
Max 7 points for a slower polynomial-time algorithm; scale partial credit accordingly.

**Standard dynamic programming rubric.** For problems worth 10 points:

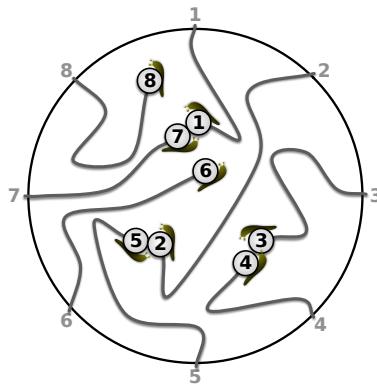
- 6 points for a correct recurrence, described either using mathematical notation or as pseudocode for a recursive algorithm.
  - + 1 point for a clear **English** description of the function you are trying to evaluate. (Otherwise, we don't even know what you're *trying* to do.)  
**Automatic zero if the English description is missing.**
  - + 1 point for stating how to call your function to get the final answer.
  - + 1 point for base case(s).  $-\frac{1}{2}$  for one *minor* bug, like a typo or an off-by-one error.
  - + 3 points for recursive case(s).  $-1$  for each *minor* bug, like a typo or an off-by-one error. **No credit for the rest of the problem if the recursive case(s) are incorrect.**
- 4 points for details of the dynamic programming algorithm
  - + 1 point for describing the memoization data structure
  - + 2 points for describing a correct evaluation order; a clear picture is usually sufficient. If you use nested loops, be sure to specify the nesting order.
  - + 1 point for time analysis
- It is *not* necessary to state a space bound.
- For problems that ask for an algorithm that computes an optimal *structure*—such as a subset, partition, subsequence, or tree—an algorithm that computes only the *value* or *cost* of the optimal structure is sufficient for full credit, unless the problem says otherwise.
- Official solutions usually include pseudocode for the final iterative dynamic programming algorithm, **but iterative pseudocode is not required for full credit.** If your solution includes iterative pseudocode, you do not need to separately describe the recurrence, memoization structure, or evaluation order. (But you still need to describe the underlying recursive function in English.)
- Official solutions will provide target time bounds. Algorithms that are faster than this target are worth more points; slower algorithms are worth fewer points, typically by 2 or 3 points (out of 10) for each factor of  $n$ . Partial credit is scaled to the new maximum score, and all points above 10 are recorded as extra credit.

We rarely include these target time bounds in the actual questions, because when we have included them, significantly more students turned in algorithms that meet the target time bound but didn't work (earning 0/10) instead of correct algorithms that are slower than the target time bound (earning 8/10).

## ∞ Homework 6 ∞

Due Tuesday, October 16, 2016 at 8pm

- Every year, as part of its annual meeting, the Antarctic Snail Lovers of Upper Glacierville hold a Round Table Mating Race. Several high-quality breeding snails are placed at the edge of a round table. The snails are numbered in order around the table from 1 to  $n$ . During the race, each snail wanders around the table, leaving a trail of slime behind it. The snails have been specially trained never to fall off the edge of the table or to cross a slime trail, even their own. If two snails meet, they are declared a breeding pair, removed from the table, and whisked away to a romantic hole in the ground to make little baby snails. Note that some snails may never find a mate, even if the race goes on forever.



The end of a typical Antarctic SLUG race. Snails 6 and 8 never find mates.

The organizers must pay  $M[3, 4] + M[2, 5] + M[1, 7]$ .

For every pair of snails, the Antarctic SLUG race organizers have posted a monetary reward, to be paid to the owners if that pair of snails meets during the Mating Race. Specifically, there is a two-dimensional array  $M[1..n, 1..n]$  posted on the wall behind the Round Table, where  $M[i, j] = M[j, i]$  is the reward to be paid if snails  $i$  and  $j$  meet. Rewards may be positive, negative, or zero.

Describe and analyze an algorithm to compute the maximum total reward that the organizers could be forced to pay, given the array  $M$  as input.

2. You and your eight-year-old nephew Elmo decide to play a simple card game. At the beginning of the game, the cards are dealt face up in a long row. Each card is worth a different number of points. After all the cards are dealt, you and Elmo take turns removing either the leftmost or rightmost card from the row, until all the cards are gone. At each turn, you can decide which of the two cards to take. The winner of the game is the player that has collected the most points when the game ends.

Having never taken an algorithms class, Elmo follows the obvious greedy strategy—when it's his turn, Elmo *always* takes the card with the higher point value. Your task is to find a strategy that will beat Elmo whenever possible. (It might seem mean to beat up on a little kid like this, but Elmo absolutely *hates* it when grown-ups let him win.)

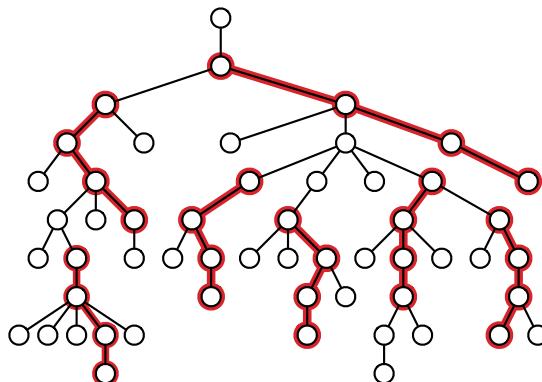
- (a) Prove that you should not also use the greedy strategy. That is, show that there is a game that you can win, but only if you do *not* follow the same greedy strategy as Elmo.
- (b) Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that you can collect playing against Elmo.
- (c) Five years later, Elmo has become a *significantly* stronger player. Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that you can collect playing against a *perfect* opponent. [Hint: What is a *perfect* opponent?]

3. One day, Alex got tired of climbing in a gym and decided to take a very large group of climber friends outside to climb. The climbing area where they went, had a huge wide boulder, not very tall, with various marked hand and foot holds. Alex quickly determined an “allowed” set of moves that her group of friends can perform to get from one hold to another.

The overall system of holds can be described by a rooted tree  $T$  with  $n$  vertices, where each vertex corresponds to a hold and each edge corresponds to an allowed move between holds. The climbing paths converge as they go up the boulder, leading to a unique hold at the summit, represented by the root of  $T$ .  $\square$

Alex and her friends (who are all excellent climbers) decided to play a game, where as many climbers as possible are simultaneously on the boulder and each climber needs to perform a sequence of *exactly*  $k$  moves. Each climber can choose an arbitrary hold to start from, and all moves must move away from the ground. Thus, each climber traces out a path of  $k$  edges in the tree  $T$ , all directed toward the root. However, no two climbers are allowed to touch the same hold; the paths followed by different climbers cannot intersect at all.

Describe and analyze an efficient algorithm to compute the maximum number of climbers that can play this game. More formally, you are given a rooted tree  $T$  and an integer  $k$ , and you want to find the largest possible number of disjoint paths in  $T$ , where each path has length  $k$ . For full credit, do **not** assume that  $T$  is a binary tree. For example, given the tree  $T$  below and  $k = 3$  as input, your algorithm should return the integer 8.



Seven disjoint paths of length  $k=3$  in a rooted tree.  
This is *not* the largest such set of paths in this tree.

---

$\square$ Q: Why do computer science professors think trees have their roots at the top?  
A: Because they've never been outside!

## Solved Problems

4. A string  $w$  of parentheses ( and ) and brackets [ and ] is *balanced* if it is generated by the following context-free grammar:

$$S \rightarrow \varepsilon \mid (S) \mid [S] \mid SS$$

For example, the string  $w = (\text{([()])}(\text{)})\text{[()]}(\text{)}$  is balanced, because  $w = xy$ , where

$$x = (\text{([()])} \text{[()}) \quad \text{and} \quad y = \text{[()}} \text{)} \text{[()}$$

Describe and analyze an algorithm to compute the length of a longest balanced subsequence of a given string of parentheses and brackets. Your input is an array  $A[1..n]$ , where  $A[i] \in \{\text{(,)}, \text{[,]}\}$  for every index  $i$ .

**Solution:** Suppose  $A[1..n]$  is the input string. For all indices  $i$  and  $j$ , we write  $A[i] \sim A[j]$  to indicate that  $A[i]$  and  $A[j]$  are matching delimiters: Either  $A[i] = ($  and  $A[j] = )$  or  $A[i] = [$  and  $A[j] = ]$ .

For all indices  $i$  and  $j$ , let  $LBS(i, j)$  denote the length of the longest balanced subsequence of the substring  $A[i..j]$ . We need to compute  $LBS(1, n)$ . This function obeys the following recurrence:

$$LBS(i, j) = \begin{cases} 0 & \text{if } i \geq j \\ \max \left\{ \max_{k=1}^{j-1} (LBS(i, k) + LBS(k+1, j)) \right\} & \text{if } A[i] \sim A[j] \\ \max_{k=1}^{j-1} (LBS(i, k) + LBS(k+1, j)) & \text{otherwise} \end{cases}$$

We can memoize this function into a two-dimensional array  $LBS[1..n, 1..n]$ . Since every entry  $LBS[i, j]$  depends only on entries in later rows or earlier columns (or both), we can evaluate this array row-by-row from bottom up in the outer loop, scanning each row from left to right in the inner loop. The resulting algorithm runs in  $O(n^3)$  time.

```

LONGESTBALANCEDSUBSEQUENCE( $A[1..n]$ ):
  for  $i \leftarrow n$  down to 1
     $LBS[i, i] \leftarrow 0$ 
    for  $j \leftarrow i + 1$  to  $n$ 
      if  $A[i] \sim A[j]$ 
         $LBS[i, j] \leftarrow LBS[i + 1, j - 1] + 2$ 
      else
         $LBS[i, j] \leftarrow 0$ 
      for  $k \leftarrow i$  to  $j - 1$ 
         $LBS[i, j] \leftarrow \max \{LBS[i, j], LBS[i, k] + LBS[k + 1, j]\}$ 
  return  $LBS[1, n]$ 
```

**Rubric:** 10 points, standard dynamic programming rubric

5. Oh, no! You've just been appointed as the new organizer of Giggle, Inc.'s annual mandatory holiday party! The employees at Giggle are organized into a strict hierarchy, that is, a tree with the company president at the root. The all-knowing oracles in Human Resources have assigned a real number to each employee measuring how "fun" the employee is. In order to keep things social, there is one restriction on the guest list: An employee cannot attend the party if their immediate supervisor is also present. On the other hand, the president of the company *must* attend the party, even though she has a negative fun rating; it's her company, after all.

Describe an algorithm that makes a guest list for the party that maximizes the sum of the "fun" ratings of the guests. The input to your algorithm is a rooted tree  $T$  describing the company hierarchy, where each node  $v$  has a field  $v.fun$  storing the "fun" rating of the corresponding employee.

**Solution (two functions):** We define two functions over the nodes of  $T$ .

- $\text{MaxFunYes}(v)$  is the maximum total "fun" of a legal party among the descendants of  $v$ , where  $v$  is definitely invited.
- $\text{MaxFunNo}(v)$  is the maximum total "fun" of a legal party among the descendants of  $v$ , where  $v$  is definitely not invited.

We need to compute  $\text{MaxFunYes}(\text{root})$ . These two functions obey the following mutual recurrences:

$$\begin{aligned}\text{MaxFunYes}(v) &= v.fun + \sum_{\text{children } w \text{ of } v} \text{MaxFunNo}(w) \\ \text{MaxFunNo}(v) &= \sum_{\text{children } w \text{ of } v} \max\{\text{MaxFunYes}(w), \text{MaxFunNo}(w)\}\end{aligned}$$

(These recurrences do not require separate base cases, because  $\sum \emptyset = 0$ .) We can memoize these functions by adding two additional fields  $v.yes$  and  $v.no$  to each node  $v$  in the tree. The values at each node depend only on the values at its children, so we can compute all  $2n$  values using a postorder traversal of  $T$ .

```
BESTPARTY( $T$ ):
    COMPUTEMAXFUN( $T.root$ )
    return  $T.root.yes$ 
```

<pre><u>COMPUTEMAXFUN(<math>v</math>):</u> <math>v.yes \leftarrow v.fun</math> <math>v.no \leftarrow 0</math> for all children <math>w</math> of <math>v</math>     COMPUTEMAXFUN(<math>w</math>)     <math>v.yes \leftarrow v.yes + w.no</math>     <math>v.no \leftarrow v.no + \max\{w.yes, w.no\}</math></pre>
--

(Yes, this is still dynamic programming; we're only traversing the tree recursively because that's the most natural way to traverse trees!) The algorithm spends  $O(1)$  time at each node, and therefore runs in  $O(n)$  time altogether. ■

---

□ A naïve recursive implementation would run in  $O(\phi^n)$  time in the worst case, where  $\phi = (1 + \sqrt{5})/2 \approx 1.618$  is the golden ratio. The worst-case tree is a path—every non-leaf node has exactly one child.

**Solution (one function):** For each node  $v$  in the input tree  $T$ , let  $\text{MaxFun}(v)$  denote the maximum total “fun” of a legal party among the descendants of  $v$ , where  $v$  may or may not be invited.

The president of the company must be invited, so none of the president’s “children” in  $T$  can be invited. Thus, the value we need to compute is

$$\text{root.fun} + \sum_{\text{grandchildren } w \text{ of root}} \text{MaxFun}(w).$$

The function  $\text{MaxFun}$  obeys the following recurrence:

$$\text{MaxFun}(v) = \max \left\{ \begin{array}{l} v.\text{fun} + \sum_{\text{grandchildren } x \text{ of } v} \text{MaxFun}(x) \\ \sum_{\text{children } w \text{ of } v} \text{MaxFun}(w) \end{array} \right\}$$

(This recurrence does not require a separate base case, because  $\sum \emptyset = 0$ .) We can memoize this function by adding an additional field  $v.\text{maxFun}$  to each node  $v$  in the tree. The value at each node depends only on the values at its children and grandchildren, so we can compute all values using a postorder traversal of  $T$ .

```
BESTPARTY( $T$ ):
    COMPUTEMAXFUN( $T.\text{root}$ )
     $\text{party} \leftarrow T.\text{root.fun}$ 
    for all children  $w$  of  $T.\text{root}$ 
        for all children  $x$  of  $w$ 
             $\text{party} \leftarrow \text{party} + x.\text{maxFun}$ 
    return  $\text{party}$ 
```

```
COMPUTEMAXFUN( $v$ ):
     $\text{yes} \leftarrow v.\text{fun}$ 
     $\text{no} \leftarrow 0$ 
    for all children  $w$  of  $v$ 
        COMPUTEMAXFUN( $w$ )
         $\text{no} \leftarrow \text{no} + w.\text{maxFun}$ 
        for all children  $x$  of  $w$ 
             $\text{yes} \leftarrow \text{yes} + x.\text{maxFun}$ 
     $v.\text{maxFun} \leftarrow \max\{\text{yes}, \text{no}\}$ 
```

(Yes, this is still dynamic programming; we’re only traversing the tree recursively because that’s the most natural way to traverse trees!  $\square$ )

The algorithm spends  $O(1)$  time at each node (because each node has exactly one parent and one grandparent) and therefore runs in  $O(n)$  time altogether.  $\blacksquare$

**Rubric:** 10 points: standard dynamic programming rubric. These are not the only correct solutions.

---

$\square$ Like the previous solution, a direct recursive implementation would run in  $O(\phi^n)$  time in the worst case, where  $\phi = (1 + \sqrt{5})/2 \approx 1.618$  is the golden ratio.

## ∞ Homework 7 ∞

Due Tuesday, October 25, 2016 at 8pm

If you use a greedy algorithm, you **must** prove that it is correct, or you will get zero points *even if your algorithm is correct*.

1. You've been hired to store a sequence of  $n$  books on shelves in a library. The order of the books is fixed by the cataloging system and cannot be changed; each shelf must store a contiguous interval of the given sequence of books. You are given two arrays  $H[1..n]$  and  $T[1..n]$ , where  $H[i]$  and  $T[i]$  are respectively the height and thickness of the  $i$ th book in the sequence. All shelves in this library have the same length  $L$ ; the total thickness of all books on any single shelf cannot exceed  $L$ .
  - (a) Suppose all the books have the same height  $h$  (that is,  $H[i] = h$  for all  $i$ ) and the shelves have height larger than  $h$ , so each book fits on every shelf. Describe and analyze a greedy algorithm to store the books in as few shelves as possible. [Hint: The algorithm is obvious, but why is it correct?]
  - (b) That was a nice warmup, but now here's the real problem. In fact the books have different heights, but you can adjust the height of each shelf to match the tallest book on that shelf. (In particular, you can change the height of any empty shelf to zero.) Now your task is to store the books so that the sum of the heights of the shelves is as small as possible. Show that your greedy algorithm from part (a) does *not* always give the best solution to this problem.
  - (c) Describe and analyze an algorithm to find the best assignment of books to shelves as described in part (b).
2. Consider a directed graph  $G$ , where each edge is colored either red, white, or blue. A walk<sup>□</sup> in  $G$  is called a *French flag walk* if its sequence of edge colors is red, white, blue, red, white, blue, and so on. More formally, a walk  $v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_k$  is a French flag walk if, for every integer  $i$ , the edge  $v_i \rightarrow v_{i+1}$  is red if  $i \bmod 3 = 0$ , white if  $i \bmod 3 = 1$ , and blue if  $i \bmod 3 = 2$ .

Describe an efficient algorithm to find all vertices in a given edge-colored directed graph  $G$  that can be reached from a given vertex  $v$  through a French flag walk.

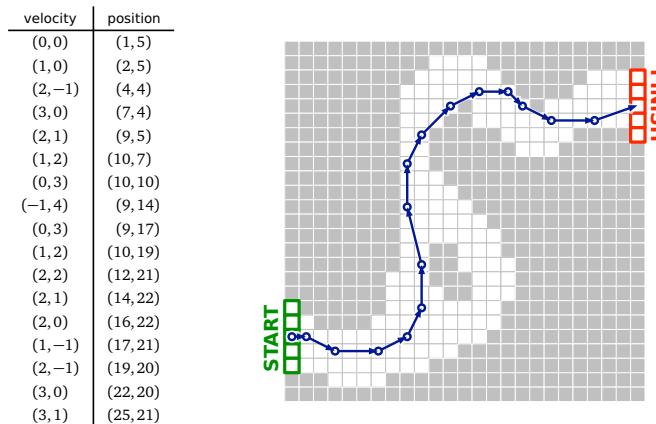
<sup>□</sup>Recall that a **walk** in a directed graph  $G$  is a sequence of vertices  $v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_k$ , such that  $v_{i-1} \rightarrow v_i$  is an edge in  $G$  for every index  $i$ . A **path** is a walk in which no vertex appears more than once.

3. **Racetrack** (also known as *Graph Racers* and *Vector Rally*) is a two-player paper-and-pencil racing game that Jeff played on the bus in 5th grade.<sup>□</sup> The game is played with a track drawn on a sheet of graph paper. The players alternately choose a sequence of grid points that represent the motion of a car around the track, subject to certain constraints explained below.

Each car has a *position* and a *velocity*, both with integer *x*- and *y*-coordinates. A subset of grid squares is marked as the *starting area*, and another subset is marked as the *finishing area*. The initial position of each car is chosen by the player somewhere in the starting area; the initial velocity of each car is always  $(0, 0)$ . At each step, the player optionally increments or decrements either or both coordinates of the car's velocity; in other words, each component of the velocity can change by at most 1 in a single step. The car's new position is then determined by adding the new velocity to the car's previous position. The new position must be inside the track; otherwise, the car crashes and that player loses the race.<sup>□</sup> The race ends when the first car reaches a position inside the finishing area.

Suppose the racetrack is represented by an  $n \times n$  array of bits, where each 0 bit represents a grid point inside the track, each 1 bit represents a grid point outside the track, the “starting area” is the first column, and the “finishing area” is the last column.

Describe and analyze an algorithm to find the minimum number of steps required to move a car from the starting line to the finish line of a given racetrack. [Hint: Build a graph. No, not that graph, a different one. What are the vertices? What are the edges? What problem is this?]



A 16-step Racetrack run, on a  $25 \times 25$  track. This is *not* the shortest run on this track.

<sup>□</sup>The actual game is a bit more complicated than the version described here. See <http://harmmade.com/vectorracer/> for an excellent online version.

<sup>□</sup>However, it is not necessary for the line between the old position and the new position to lie entirely within the track. Sometimes Speed Racer has to push the A button.

## Solved Problem

4. Professor McClane takes you out to a lake and hands you three empty jars. Each jar holds a positive integer number of gallons; the capacities of the three jars may or may not be different. The professor then demands that you put exactly  $k$  gallons of water into one of the jars (which one doesn't matter), for some integer  $k$ , using only the following operations:
- Fill a jar with water from the lake until the jar is full.
  - Empty a jar of water by pouring water into the lake.
  - Pour water from one jar to another, until either the first jar is empty or the second jar is full, whichever happens first.

For example, suppose your jars hold 6, 10, and 15 gallons. Then you can put 13 gallons of water into the third jar in six steps:

- Fill the third jar from the lake.
- Fill the first jar from the third jar. (Now the third jar holds 9 gallons.)
- Empty the first jar into the lake.
- Fill the second jar from the lake.
- Fill the first jar from the second jar. (Now the second jar holds 4 gallons.)
- Empty the second jar into the third jar.

Describe and analyze an efficient algorithm that either finds the smallest number of operations that leave exactly  $k$  gallons in any jar, or reports correctly that obtaining exactly  $k$  gallons of water is impossible. Your input consists of the capacities of the three jars and the positive integer  $k$ . For example, given the four numbers 6, 10, 15 and 13 as input, your algorithm should return the number 6 (for the sequence of operations listed above).

**Solution:** Let  $A, B, C$  denote the capacities of the three jars. We reduce the problem to breadth-first search in the following directed graph:

- $V = \{(a, b, c) \mid 0 \leq a \leq p \text{ and } 0 \leq b \leq B \text{ and } 0 \leq c \leq C\}$ . Each vertex corresponds to a possible **configuration** of water in the three jars. There are  $(A+1)(B+1)(C+1) = O(ABC)$  vertices altogether.
- The graph has a directed edge  $(a, b, c) \rightarrow (a', b', c')$  whenever it is possible to move from the first configuration to the second in one step. Specifically, there is an edge from  $(a, b, c)$  to each of the following vertices (except those already equal to  $(a, b, c)$ ):
  - $(0, b, c)$  and  $(a, 0, c)$  and  $(a, b, 0)$  — dumping a jar into the lake
  - $(A, b, c)$  and  $(a, B, c)$  and  $(a, b, C)$  — filling a jar from the lake
  - $\begin{cases} (0, a+b, c) & \text{if } a+b \leq B \\ (a+b-B, B, c) & \text{if } a+b \geq B \end{cases}$  — pouring from the first jar into the second
  - $\begin{cases} (0, b, a+c) & \text{if } a+c \leq C \\ (a+c-C, b, C) & \text{if } a+c \geq C \end{cases}$  — pouring from the first jar into the third
  - $\begin{cases} (a+b, 0, c) & \text{if } a+b \leq A \\ (A, a+b-A, c) & \text{if } a+b \geq A \end{cases}$  — pouring from the second jar into the first

- $\begin{cases} (a, 0, b+c) & \text{if } b+c \leq C \\ (a, b+c-C, C) & \text{if } b+c \geq C \end{cases}$  — pouring from the second jar into the third
- $\begin{cases} (a+c, b, 0) & \text{if } a+c \leq A \\ (A, b, a+c-A) & \text{if } a+c \geq A \end{cases}$  — pouring from the third jar into the first
- $\begin{cases} (a, b+c, 0) & \text{if } b+c \leq B \\ (a, B, b+c-B) & \text{if } b+c \geq B \end{cases}$  — pouring from the third jar into the second

Since each vertex has at most 12 outgoing edges, there are at most  $12(A+1) \times (B+1)(C+1) = O(ABC)$  edges altogether.

To solve the jars problem, we need to find the **shortest path** in  $G$  from the start vertex  $(0, 0, 0)$  to any target vertex of the form  $(k, \cdot, \cdot)$  or  $(\cdot, k, \cdot)$  or  $(\cdot, \cdot, k)$ . We can compute this shortest path by calling **breadth-first search** starting at  $(0, 0, 0)$ , and then examining every target vertex by brute force. If BFS does not visit any target vertex, we report that no legal sequence of moves exists. Otherwise, we find the target vertex closest to  $(0, 0, 0)$  and trace its parent pointers back to  $(0, 0, 0)$  to determine the shortest sequence of moves. The resulting algorithm runs in  $O(V + E) = O(ABC)$  time.

We can make this algorithm faster by observing that every move either leaves at least one jar empty or leaves at least one jar full. Thus, we only need vertices  $(a, b, c)$  where either  $a = 0$  or  $b = 0$  or  $c = 0$  or  $a = A$  or  $b = B$  or  $c = C$ ; no other vertices are reachable from  $(0, 0, 0)$ . The number of non-redundant vertices and edges is  $O(AB + BC + AC)$ . Thus, if we only construct and search the relevant portion of  $G$ , the algorithm runs in  $O(AB + BC + AC)$  time. ■

**Rubric (for graph reduction problems):** 10 points:

- 2 for correct vertices
- 2 for correct edges
  - $\frac{1}{2}$  for forgetting “directed”
- 2 for stating the correct problem (shortest paths)
  - “Breadth-first search” is not a problem; it’s an algorithm.
- 2 points for correctly applying the correct algorithm (breadth-first search)
  - 1 for using Dijkstra instead of BFS
- 2 points for time analysis in terms of the input parameters.
- Max 8 points for  $O(ABC)$  time; scale partial credit

 Homework 8 

Due Tuesday, November 1, 2016 at 8pm

---

This is the last homework before Midterm 2.

---

1. After a grueling algorithms midterm, you decide to take the bus home. Since you planned ahead, you have a schedule that lists the times and locations of every stop of every bus in Champaign-Urbana. Champaign-Urbana is currently suffering from a plague of zombies, so even though the bus stops have fences that *supposedly* keep the zombies out, you'd still like to spend as little time waiting at bus stops as possible. Unfortunately, there isn't a single bus that visits both your exam building and your home; you must transfer between buses at least once.

Describe and analyze an algorithm to determine a sequence of bus rides from Siebel to your home, that minimizes the total time you spend waiting at bus stops. You can assume that there are  $b$  different bus lines, and each bus stops  $n$  times per day. Assume that the buses run exactly on schedule, that you have an accurate watch, and that walking between bus stops is too dangerous to even contemplate.

2. Kris is a professional rock climber (friends with Alex and the rest of the climbing crew from HW6) who is competing in the U.S. climbing nationals. The competition requires Kris to use as many holds on the climbing wall as possible, using only transitions that have been explicitly allowed by the route-setter.

The climbing wall has  $n$  holds. Kris is given a list of  $m$  pairs  $(x, y)$  of holds, each indicating that moving directly from hold  $x$  to hold  $y$  is allowed; however, moving directly from  $y$  to  $x$  is not allowed unless the list also includes the pair  $(y, x)$ . Kris needs to figure out a sequence of allowed transitions that uses as many holds as possible, since each new hold increases his score by one point. The rules allow Kris to choose the first and last hold in his climbing route. The rules also allow him to use each hold as many times as he likes; however, only the first use of each hold increases Kris's score.

- (a) Define the natural graph representing the input. Describe and analyze an algorithm to solve Kris's climbing problem if you are guaranteed that the input graph is a dag.
- (b) Describe and analyze an algorithm to solve Kris's climbing problem with no restrictions on the input graph.

Both of your algorithms should output the maximum possible score that Kris can earn.

3. Many years later, in a land far, far away, after winning all the U.S. national competitions for 10 years in a row, Kris retired from competitive climbing and became a route setter for competitions. However, as the years passed, the rules changed. Climbers are now required to climb along the *shortest* sequence of legal moves from one specific node to another, where the distance between two holds is specified by the route setter. In addition to the usual set of  $n$  holds and  $m$  valid moves between them (as in the previous problem), climbers are now told their start hold  $s$ , their finish hold  $t$ , and the distance from  $x$  to  $y$  for every allowed move  $(x, y)$ .

Rather than make up this year's new route completely from scratch, Kris decides to make one small change to last year's input. The previous route setter suggested a list of  $k$  new allowed moves and their distances. Kris needs to choose the single edge from this list of suggestions that decreases the distance from  $s$  to  $t$  as much as possible.

Describe and analyze an algorithm to solve Kris's problem. Your input consists of the following information:

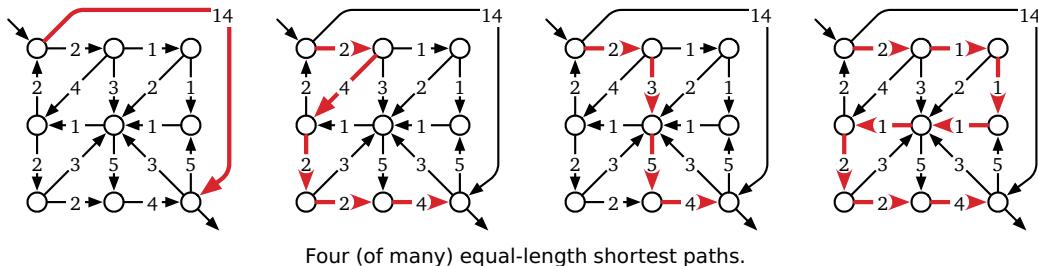
- A directed graph  $G = (V, E)$ .
- Two vertices  $s, t \in V$ .
- A set of  $k$  new edges  $E'$ , such that  $E \cap E' = \emptyset$
- A length  $\ell(e) \geq 0$  for every edge  $e \in E \cup E'$ .

Your algorithm should return the edge  $e \in E'$  whose addition to the graph yields the smallest shortestpath distance from  $s$  to  $t$ .

For full credit, your algorithm should run in  $O(m \log n + k)$  time, but as always, a slower correct algorithm is worth more than a faster incorrect algorithm.

## Solved Problem

4. Although we typically speak of “the” shortest path between two nodes, a single graph could contain several minimum-length paths with the same endpoints.



Describe and analyze an algorithm to determine the *number* of shortest paths from a source vertex  $s$  to a target vertex  $t$  in an arbitrary directed graph  $G$  with weighted edges. You may assume that all edge weights are positive and that all necessary arithmetic operations can be performed in  $O(1)$  time.

[Hint: Compute shortest path distances from  $s$  to every other vertex. Throw away all edges that cannot be part of a shortest path from  $s$  to another vertex. What's left?]

**Solution:** We start by computing shortest-path distances  $dist(v)$  from  $s$  to  $v$ , for every vertex  $v$ , using Dijkstra's algorithm. Call an edge  $u \rightarrow v$  **tight** if  $dist(u) + w(u \rightarrow v) = dist(v)$ . Every edge in a shortest path from  $s$  to  $t$  must be tight. Conversely, every path from  $s$  to  $t$  that uses only tight edges has total length  $dist(t)$  and is therefore a shortest path!

Let  $H$  be the subgraph of all tight edges in  $G$ . We can easily construct  $H$  in  $O(V + E)$  time. Because all edge weights are positive,  $H$  is a directed acyclic graph. It remains only to count the number of paths from  $s$  to  $t$  in  $H$ .

For any vertex  $v$ , let  $PathsToT(v)$  denote the number of paths in  $H$  from  $v$  to  $t$ ; we need to compute  $PathsToT(s)$ . This function satisfies the following simple recurrence:

$$PathsToT(v) = \begin{cases} 1 & \text{if } v = t \\ \sum_{v \rightarrow w} PathsToT(w) & \text{otherwise} \end{cases}$$

In particular, if  $v$  is a sink but  $v \neq t$  (and thus there are no paths from  $v$  to  $t$ ), this recurrence correctly gives us  $PathsToT(v) = \sum \emptyset = 0$ .

We can memoize this function into the graph itself, storing each value  $PathsToT(v)$  at the corresponding vertex  $v$ . Since each subproblem depends only on its successors in  $H$ , we can compute  $PathsToT(v)$  for all vertices  $v$  by considering the vertices in reverse topological order, or equivalently, by performing a depth-first search of  $H$  starting at  $s$ . The resulting algorithm runs in  $O(V + E)$  time.

The overall running time of the algorithm is dominated by Dijkstra's algorithm in the preprocessing phase, which runs in  $O(E \log V)$  time. ■

**Rubric:** 10 points = 5 points for reduction to counting paths in a dag + 5 points for the path-counting algorithm (standard dynamic programming rubric)

## ∞ Homework 9 ∞

Due Tuesday, November 15, 2016 at 8pm

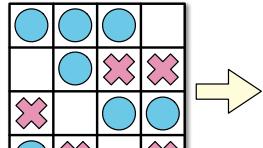
---

1. Consider the following problem, called BoxDEPTH: Given a set of  $n$  axis-aligned rectangles in the plane, how big is the largest subset of these rectangles that contain a common point?
    - (a) Describe a polynomial-time reduction from BoxDEPTH to MAXCLIQUE, and prove that your reduction is correct.
    - (b) Describe and analyze a polynomial-time algorithm for BoxDEPTH. [Hint: Don't try to optimize the running time;  $O(n^3)$  is good enough.]
    - (c) Why don't these two results imply that P=NP?
  
  2. This problem asks you to describe polynomial-time reductions between two closely related problems:
    - SUBSETSUM: Given a set  $S$  of positive integers and a target integer  $T$ , is there a subset of  $S$  whose sum is  $T$ ?
    - PARTITION: Given a set  $S$  of positive integers, is there a way to partition  $S$  into two subsets  $S_1$  and  $S_2$  that have the same sum?
    - (a) Describe a polynomial-time reduction from SUBSETSUM to PARTITION.
    - (b) Describe a polynomial-time reduction from PARTITION to SUBSETSUM.

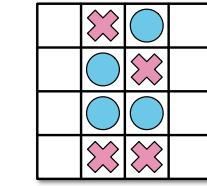
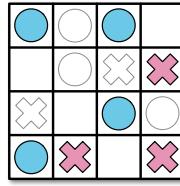
Don't forget to prove that your reductions are correct.
  
  3. Suppose you are given a graph  $G = (V, E)$  where  $V$  represents a collection of people and an edge between two people indicates that they are friends. You wish to partition  $V$  into at most  $k$  non-overlapping groups  $V_1, V_2, \dots, V_k$  such that each group is very cohesive. One way to model cohesiveness is to insist that each pair of people in the same group should be friends; in other words, they should form a clique.
- Prove that the following problem is NP-hard: Given an undirected graph  $G$  and an integer  $k$ , decide whether the vertices of  $G$  can be partitioned into  $k$  cliques.

## Solved Problem

4. Consider the following solitaire game. The puzzle consists of an  $n \times m$  grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) every row contains at least one stone, and (2) no column contains stones of both colors. For some initial configurations of stones, reaching this goal is impossible.



A solvable puzzle and one of its many solutions.



An unsolvable puzzle.

Prove that it is NP-hard to determine, given an initial configuration of red and blue stones, whether this puzzle can be solved.

**Solution:** We show that this puzzle is NP-hard by describing a reduction from 3SAT.

Let  $\Phi$  be a 3CNF boolean formula with  $m$  variables and  $n$  clauses. We transform this formula into a puzzle configuration in polynomial time as follows. The size of the board is  $n \times m$ . The stones are placed as follows, for all indices  $i$  and  $j$ :

- If the variable  $x_j$  appears in the  $i$ th clause of  $\Phi$ , we place a blue stone at  $(i, j)$ .
- If the negated variable  $\bar{x}_j$  appears in the  $i$ th clause of  $\Phi$ , we place a red stone at  $(i, j)$ .
- Otherwise, we leave cell  $(i, j)$  blank.

*We claim that this puzzle has a solution if and only if  $\Phi$  is satisfiable.* This claim immediately implies that solving the puzzle is NP-hard. We prove our claim as follows:

$\implies$  First, suppose  $\Phi$  is satisfiable; consider an arbitrary satisfying assignment. For each index  $j$ , remove stones from column  $j$  according to the value assigned to  $x_j$ :

- If  $x_j = \text{TRUE}$ , remove all red stones from column  $j$ .
- If  $x_j = \text{FALSE}$ , remove all blue stones from column  $j$ .

In other words, remove precisely the stones that correspond to FALSE literals. Because every variable appears in at least one clause, each column now contains stones of only one color (if any). On the other hand, each clause of  $\Phi$  must contain at least one TRUE literal, and thus each row still contains at least one stone. We conclude that the puzzle is satisfiable.

$\Leftarrow$  On the other hand, suppose the puzzle is solvable; consider an arbitrary solution. For each index  $j$ , assign a value to  $x_j$  depending on the colors of stones left in column  $j$ :

- If column  $j$  contains blue stones, set  $x_j = \text{TRUE}$ .
- If column  $j$  contains red stones, set  $x_j = \text{FALSE}$ .
- If column  $j$  is empty, set  $x_j$  arbitrarily.

In other words, assign values to the variables so that the literals corresponding to the remaining stones are all TRUE. Each row still has at least one stone, so each clause of  $\Phi$  contains at least one TRUE literal, so this assignment makes  $\Phi = \text{TRUE}$ . We conclude that  $\Phi$  is satisfiable.

This reduction clearly requires only polynomial time. ■

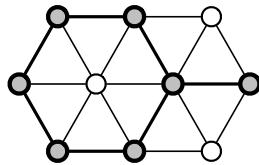
**Rubric (for all polynomial-time reductions):** 10 points =

- + 3 points for the reduction itself
  - For an NP-hardness proof, the reduction must be from a known NP-hard problem. You can use any of the NP-hard problems listed in the lecture notes (except the one you are trying to prove NP-hard, of course).
- + 3 points for the “if” proof of correctness
- + 3 points for the “only if” proof of correctness
- + 1 point for writing “polynomial time”
  - An incorrect polynomial-time reduction that still satisfies half of the correctness proof is worth at most 4/10.
  - A reduction in the wrong direction is worth 0/10.

## ❖ Homework 10 ❖

Due Tuesday, November 29, 2016 at 8pm

1. A subset  $S$  of vertices in an undirected graph  $G$  is called *almost independent* if at most 374 edges in  $G$  have both endpoints in  $S$ . Prove that finding the size of the largest almost-independent set of vertices in a given undirected graph is NP-hard.
  
2. A subset  $S$  of vertices in an undirected graph  $G$  is called *triangle-free* if, for every triple of vertices  $u, v, w \in S$ , at least one of the three edges  $uv, uw, vw$  is *absent* from  $G$ . Prove that finding the size of the largest triangle-free subset of vertices in a given undirected graph is NP-hard.



A triangle-free subset of 7 vertices.  
This is **not** the largest triangle-free subset in this graph.

3. Charon needs to ferry  $n$  recently deceased people across the river Acheron into Hades. Certain pairs of these people are sworn enemies, who cannot be together on either side of the river unless Charon is also present. (If two enemies are left alone, one will steal the obol from the other's mouth, leaving them to wander the banks of the Acheron as a ghost for all eternity. Let's just say this is a Very Bad Thing.) The ferry can hold at most  $k$  passengers at a time, including Charon, and only Charon can pilot the ferry.

Prove that it is NP-hard to decide whether Charon can ferry all  $n$  people across the Acheron unharmed. □ The input for Charon's problem consists of the integers  $k$  and  $n$  and an  $n$ -vertex graph  $G$  describing the pairs of enemies. The output is either TRUE or FALSE.

□Aside from being, you know, dead.

Problem 3 is a generalization of the following extremely well-known puzzle, whose first known appearance is in the treatise *Propositiones ad Acuedos Juvenes* [Problems to Sharpen the Young] by the 8th-century English scholar Alcuin of York.□

#### XVIII. PROPOSITIO DE HOMINE ET CAPRA ET LVPO.

Homo quidam debebat ultra fluum transferre lupum, capram, et fasciculum cauli. Et non potuit aliam nauem inuenire, nisi quae duos tantum ex ipsis ferre ualebat. Praeceptum itaque ei fuerat, ut omnia haec ultra illaesha omnino transferret. Dicat, qui potest, quomodo eis illaesis transire potuit?

**Solutio.** Simili namque tenore ducerem prius capram et dimitterem foris lupum et caulum. Tum deinde uenirem, lupumque transferrem: lupoque foris misso capram nauis receptam ultra reducerem; capramque foris missam caulum transueherem ultra; atque iterum remigassem, capramque assumptam ultra duxissem. Sicque faciendo facta erit remigatio salubris, absque uoragine lacerationis.

In case your classical Latin is rusty, here is an English translation:

#### XVIII. THE PROBLEM OF THE MAN, THE GOAT, AND THE WOLF.

A man needed to transfer a wolf, a goat, and a bundle of cabbage across a river. However, he found that his boat could only bear the weight of two [objects at a time, including the man]. And he had to get everything across unharmed. Tell me if you can: How they were able to cross unharmed?

**Solution.** In a similar fashion [as an earlier problem], I would first take the goat across and leave the wolf and cabbage on the opposite bank. Then I would take the wolf across; leaving the wolf on shore, I would retrieve the goat and bring it back again. Then I would leave the goat and take the cabbage across. And then I would row across again and get the goat. In this way the crossing would go well, without any threat of slaughter.

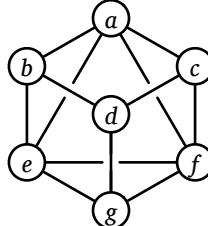
Please do not write your solution to problem 3 in classical Latin.

---

□At least, we *think* that's who wrote it; the evidence for his authorship is rather circumstantial, although we do know from his correspondence with Charlemagne that he sent the emperor some “simple arithmetical problems for fun”. Most scholars believe that even if Alcuin is the actual author of the *Propositiones*, he didn’t come up with the problems himself, but just collected his problems from other sources. Some things never change.

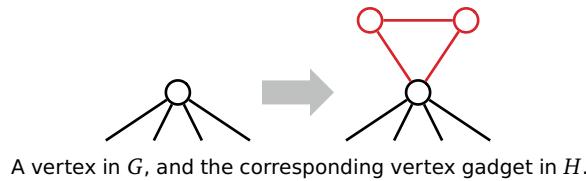
## Solved Problem

4. A *double-Hamiltonian tour* in an undirected graph  $G$  is a closed walk that visits every vertex in  $G$  exactly twice. Prove that it is NP-hard to decide whether a given graph  $G$  has a double-Hamiltonian tour.



This graph contains the double-Hamiltonian tour  $a \rightarrow b \rightarrow d \rightarrow g \rightarrow e \rightarrow b \rightarrow d \rightarrow c \rightarrow f \rightarrow a \rightarrow c \rightarrow f \rightarrow g \rightarrow e \rightarrow a$ .

**Solution:** We prove the problem is NP-hard with a reduction from the standard Hamiltonian cycle problem. Let  $G$  be an arbitrary undirected graph. We construct a new graph  $H$  by attaching a small gadget to every vertex of  $G$ . Specifically, for each vertex  $v$ , we add two vertices  $v^\flat$  and  $v^\sharp$ , along with three edges  $vv^\flat$ ,  $vv^\sharp$ , and  $v^\flat v^\sharp$ .



I claim that  $G$  has a Hamiltonian cycle if and only if  $H$  has a double-Hamiltonian tour.

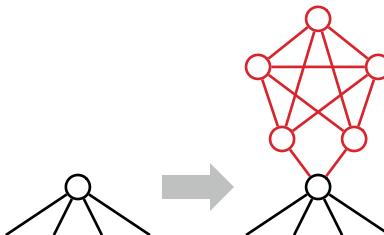
$\implies$  Suppose  $G$  has a Hamiltonian cycle  $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n \rightarrow v_1$ . We can construct a double-Hamiltonian tour of  $H$  by replacing each vertex  $v_i$  with the following walk:

$$\dots \rightarrow v_i \rightarrow v_i^\flat \rightarrow v_i^\sharp \rightarrow v_i^\flat \rightarrow v_i^\sharp \rightarrow v_i \rightarrow \dots$$

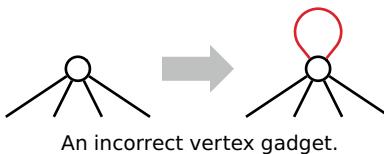
$\Leftarrow$  Conversely, suppose  $H$  has a double-Hamiltonian tour  $D$ . Consider any vertex  $v$  in the original graph  $G$ ; the tour  $D$  must visit  $v$  exactly twice. Those two visits split  $D$  into two closed walks, each of which visits  $v$  exactly once. Any walk from  $v^\flat$  or  $v^\sharp$  to any other vertex in  $H$  must pass through  $v$ . Thus, one of the two closed walks visits only the vertices  $v$ ,  $v^\flat$ , and  $v^\sharp$ . Thus, if we simply remove the vertices in  $H \setminus G$  from  $D$ , we obtain a closed walk in  $G$  that visits every vertex in  $G$  once.

Given any graph  $G$ , we can clearly construct the corresponding graph  $H$  in polynomial time.

With more effort, we can construct a graph  $H$  that contains a double-Hamiltonian tour *that traverses each edge of  $H$  at most once* if and only if  $G$  contains a Hamiltonian cycle. For each vertex  $v$  in  $G$  we attach a more complex gadget containing five vertices and eleven edges, as shown on the next page. ■

A vertex in  $G$ , and the corresponding modified vertex gadget in  $H$ .

**Common incorrect solution (self-loops):** We attempt to prove the problem is NP-hard with a reduction from the Hamiltonian cycle problem. Let  $G$  be an arbitrary undirected graph. We construct a new graph  $H$  by attaching a self-loop every vertex of  $G$ . Given any graph  $G$ , we can clearly construct the corresponding graph  $H$  in polynomial time.

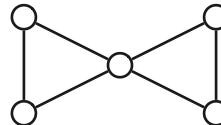


An incorrect vertex gadget.

Suppose  $G$  has a Hamiltonian cycle  $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n \rightarrow v_1$ . We can construct a double-Hamiltonian tour of  $H$  by alternating between edges of the Hamiltonian cycle and self-loops:

$$v_1 \rightarrow v_1 \rightarrow v_2 \rightarrow v_2 \rightarrow v_3 \rightarrow \dots \rightarrow v_n \rightarrow v_n \rightarrow v_1.$$

On the other hand, if  $H$  has a double-Hamiltonian tour, we *cannot* conclude that  $G$  has a Hamiltonian cycle, because we cannot guarantee that a double-Hamiltonian tour in  $H$  uses *any* self-loops. The graph  $G$  shown below is a counterexample; it has a double-Hamiltonian tour (even before adding self-loops) but no Hamiltonian cycle.



This graph has a double-Hamiltonian tour.



**Rubric (for all polynomial-time reductions):** 10 points =

- + 3 points for the reduction itself
  - For an NP-hardness proof, the reduction must be from a known NP-hard problem. You can use any of the NP-hard problems listed in the lecture notes (except the one you are trying to prove NP-hard, of course).
- + 3 points for the “if” proof of correctness
- + 3 points for the “only if” proof of correctness
- + 1 point for writing “polynomial time”
  - An incorrect polynomial-time reduction that still satisfies half of the correctness proof is worth at most 4/10.
  - A reduction in the wrong direction is worth 0/10.

**CS/ECE 374 ♦ Fall 2016**

**❖ Homework 11 ❖**

“Due” Tuesday, December, 2016

---

This homework is only for practice; it will not be graded. However, **similar questions may appear on the final exam**, so we still strongly recommend treating this as a regular homework. Solutions will be released next Tuesday as usual.

---

1. Recall that  $w^R$  denotes the reversal of string  $w$ ; for example,  $\text{TURING}^R = \text{GNIRUT}$ . Prove that the following language is undecidable.

$$\text{REVACCEPT} := \{\langle M \rangle \mid M \text{ accepts } \langle M \rangle^R\}$$

Note that Rice’s theorem does *not* apply to this language.

2. Let  $M$  be a Turing machine, let  $w$  be an arbitrary input string, and let  $s$  be an integer. We say that  $M$  *accepts w in space s* if, given  $w$  as input,  $M$  accesses only the first  $s$  (or fewer) cells on its tape and eventually accepts.

- (a) Sketch a Turing machine/algorithm that correctly decides the following language:

$$\{\langle M, w \rangle \mid M \text{ accepts } w \text{ in space } |w|^2\}$$

- (b) Prove that the following language is undecidable:

$$\{\langle M \rangle \mid M \text{ accepts at least one string } w \text{ in space } |w|^2\}$$

3. Consider the language  $\text{SOMETIMESHALT} = \{\langle M \rangle \mid M \text{ halts on at least one input string}\}$ . Note that  $\langle M \rangle \in \text{SOMETIMESHALT}$  does not imply that  $M$  *accepts* any strings; it is enough that  $M$  *halts* on (and possibly rejects) some string.

- (a) Prove that  $\text{SOMETIMESHALT}$  is undecidable.

- (b) Sketch a Turing machine/algorithm that *accepts*  $\text{SOMETIMESHALT}$ .

## Solved Problem

4. For each of the following languages, either prove that the language is decidable, or prove that the language is undecidable.

$$(a) L_0 = \{ \langle M \rangle \mid \text{given any input string, } M \text{ eventually leaves its } \texttt{start} \text{ state}\}$$

**Solution:** We can determine whether a given Turing machine  $M$  always leaves its start state by careful analysis of its transition function  $\delta$ . As a technical point, I will assume that crashing on the first transition does *not* count as leaving the **start** state.

- If  $\delta(\texttt{start}, a) = (\cdot, \cdot, -1)$  for any input symbol  $a \in \Sigma$ , then  $M$  crashes on input  $a$  without leaving the **start** state.
- If  $\delta(\texttt{start}, \square) = (\cdot, \cdot, -1)$ , then  $M$  crashes on the empty input without leaving the **start** state.
- Otherwise,  $M$  moves to the right until it leaves the **start** state. There are two subcases to consider:
  - If  $\delta(\texttt{start}, \square) = (\texttt{start}, \cdot, +1)$ , then  $M$  loops forever on the empty input without leaving the **start** state.
  - Otherwise, for any input string,  $M$  must eventually leave the **start** state, either when reading some input symbol or when reading the first blank.

It is straightforward (but tedious) to perform this case analysis with a Turing machine that receives the encoding  $\langle M \rangle$  as input. We conclude that  $L_0$  is **decidable**. ■

$$(b) L_1 = \{ \langle M \rangle \mid M \text{ decides } L_0 \}$$

**Solution:**

- By part (a), there is a Turing machine that decides  $L_0$ .
- Let  $M_{\text{reject}}$  be a Turing machine that immediately **rejects** its input, by defining  $\delta(\texttt{start}, a) = \text{reject}$  for all  $a \in \Sigma \cup \{\square\}$ . Then  $M_{\text{reject}}$  decides the language  $\emptyset \neq L_0$ . ■

Thus, Rice's Decision Theorem implies that  $L_1$  is **undecidable**.

$$(c) L_2 = \{ \langle M \rangle \mid M \text{ decides } L_1 \}$$

**Solution:** By part (b), no Turing machine decides  $L_1$ , which implies that  $L_2 = \emptyset$ . Thus,  $M_{\text{reject}}$  correctly decides  $L_2$ . We conclude that  $L_2$  is **decidable**. ■

$$(d) L_3 = \{ \langle M \rangle \mid M \text{ decides } L_2 \}$$

**Solution:** Because  $L_2 = \emptyset$ , we have

$$L_3 = \{ \langle M \rangle \mid M \text{ decides } \emptyset \} = \{ \langle M \rangle \mid \text{REJECT}(M) = \Sigma^* \}$$

- We have already seen a Turing machine  $M_{\text{reject}}$  such that  $\text{REJECT}(M_{\text{reject}}) = \Sigma^*$ .
- Let  $M_{\text{accept}}$  be a Turing machine that immediately **accepts** its input, by defining  $\delta(\texttt{start}, a) = \text{accept}$  for all  $a \in \Sigma \cup \{\square\}$ . Then  $\text{REJECT}(M_{\text{accept}}) = \emptyset \neq \Sigma^*$ . ■

Thus, Rice's Rejection Theorem implies that  $L_3$  is **undecidable**.

$$(e) L_4 = \{\langle M \rangle \mid M \text{ decides } L_3\}$$

**Solution:** By part (b), no Turing machine decides  $L_3$ , which implies that  $L_4 = \emptyset$ . Thus,  $M_{\text{reject}}$  correctly decides  $L_4$ . We conclude that  $L_4$  is **decidable**.

At this point, we have fallen into a loop. For any  $k > 4$ , define

$$L_k = \{\langle M \rangle \mid M \text{ decides } L_{k-1}\}.$$

Then  $L_k$  is decidable (because  $L_k = \emptyset$ ) if and only if  $k$  is even. ■

**Rubric:** 10 points: 4 for part (a) + 1½ for each other part.

**Rubric (for all undecidability proofs, out of 10 points):**

**Diagonalization:**

- + 4 for correct wrapper Turing machine
- + 6 for self-contradiction proof (= 3 for  $\Leftarrow$  + 3 for  $\Rightarrow$ )

**Reduction:**

- + 4 for correct reduction
- + 3 for “if” proof
- + 3 for “only if” proof

**Rice’s Theorem:**

- + 4 for positive Turing machine
- + 4 for negative Turing machine
- + 2 for other details (including using the correct variant of Rice’s Theorem)

The following problems ask you to prove some “obvious” claims about recursively-defined string functions. In each case, we want a self-contained, step-by-step induction proof that builds on formal definitions and prior results, *not* on intuition. In particular, your proofs must refer to the formal recursive definitions of string length and string concatenation:

$$|w| := \begin{cases} 0 & \text{if } w = \epsilon \\ 1 + |x| & \text{if } w = ax \text{ for some symbol } a \text{ and some string } x \end{cases}$$

$$w \bullet z := \begin{cases} z & \text{if } w = \epsilon \\ a \cdot (x \bullet z) & \text{if } w = ax \text{ for some symbol } a \text{ and some string } x \end{cases}$$

You may freely use the following results, which are proved in the lecture notes:

**Lemma 1:**  $w \bullet \epsilon = w$  for all strings  $w$ .

**Lemma 2:**  $|w \bullet x| = |w| + |x|$  for all strings  $w$  and  $x$ .

**Lemma 3:**  $(w \bullet x) \bullet y = w \bullet (x \bullet y)$  for all strings  $w$ ,  $x$ , and  $y$ .

The **reversal**  $w^R$  of a string  $w$  is defined recursively as follows:

$$w^R := \begin{cases} \epsilon & \text{if } w = \epsilon \\ x^R \bullet a & \text{if } w = ax \text{ for some symbol } a \text{ and some string } x \end{cases}$$

For example,  $\text{STRESSED}^R = \text{DESSERTS}$  and  $\text{WTF374}^R = \text{473FTW}$ .

1. Prove that  $|w| = |w^R|$  for every string  $w$ .
2. Prove that  $(w \bullet z)^R = z^R \bullet w^R$  for all strings  $w$  and  $z$ .
3. Prove that  $(w^R)^R = w$  for every string  $w$ .

[Hint: You need #2 to prove #3, but you may find it easier to solve #3 first.]

**To think about later:** Let  $\#(a, w)$  denote the number of times symbol  $a$  appears in string  $w$ . For example,  $\#(\text{X}, \text{WTF374}) = 0$  and  $\#(\text{0}, \text{000010101010010100}) = 12$ .

4. Give a formal recursive definition of  $\#(a, w)$ .
5. Prove that  $\#(a, w \bullet z) = \#(a, w) + \#(a, z)$  for all symbols  $a$  and all strings  $w$  and  $z$ .
6. Prove that  $\#(a, w^R) = \#(a, w)$  for all symbols  $a$  and all strings  $w$ .

Give regular expressions for each of the following languages over the alphabet  $\{0, 1\}$ .

1. All strings containing the substring **000**.
2. All strings *not* containing the substring **000**.
3. All strings in which every run of **0**s has length at least 3.
4. All strings in which every substring **000** appears after every **1**.
5. All strings containing at least three **0**s.
6. Every string except **000**. [*Hint: Don't try to be clever.*]

**Work on these later:**

7. All strings  $w$  such that *in every prefix of  $w$* , the number of **0**s and **1**s differ by at most 1.
- \*8. All strings containing at least two **0**s and at least one **1**.
- \*9. All strings  $w$  such that *in every prefix of  $w$* , the number of **0**s and **1**s differ by at most 2.
- ★10. All strings in which the substring **000** appears an even number of times.  
(For example, **0001000** and **0000** are in this language, but **00000** is not.)

Describe deterministic finite-state automata that accept each of the following languages over the alphabet  $\Sigma = \{0, 1\}$ . Describe briefly what each state in your DFAs *means*.

Either drawings or formal descriptions are acceptable, as long as the states  $Q$ , the start state  $s$ , the accept states  $A$ , and the transition function  $\delta$  are all be clear. Try to keep the number of states small.

1. All strings containing the substring **000**.
2. All strings *not* containing the substring **000**.
3. All strings in which every run of **0**s has length at least 3.
4. All strings in which no substring **000** appears before a **1**.
5. All strings containing at least three **0**s.
6. Every string except **000**. [*Hint: Don't try to be clever.*]

**Work on these later:**

7. All strings  $w$  such that *in every prefix of  $w$* , the number of **0**s and **1**s differ by at most 1.
  8. All strings containing at least two **0**s and at least one **1**.
  9. All strings  $w$  such that *in every prefix of  $w$* , the number of **0**s and **1**s differ by at most 2.
- \*10. All strings in which the substring **000** appears an even number of times.  
(For example, **0001000** and **0000** are in this language, but **00000** is not.)

Describe deterministic finite-state automata that accept each of the following languages over the alphabet  $\Sigma = \{\texttt{0}, \texttt{1}\}$ . You may find it easier to describe these DFAs formally than to draw pictures.

Either drawings or formal descriptions are acceptable, as long as the states  $Q$ , the start state  $s$ , the accept states  $A$ , and the transition function  $\delta$  are all be clear. Try to keep the number of states small.

1. All strings in which the number of **0**s is even and the number of **1**s is *not* divisible by 3.
2. All strings that are **both** the binary representation of an integer divisible by 3 **and** the ternary (base-3) representation of an integer divisible by 4.

For example, the string **1100** is an element of this language, because it represents  $2^3 + 2^2 = 12$  in binary and  $3^3 + 3^2 = 36$  in ternary.

#### Work on these later:

3. All strings  $w$  such that  $\binom{|w|}{2} \bmod 6 = 4$ . [Hint: Maintain both  $\binom{|w|}{2} \bmod 6$  and  $|w| \bmod 6$ .]
- \*4. All strings  $w$  such that  $F_{\#(\texttt{10}, w)} \bmod 10 = 4$ , where  $\#(\texttt{10}, w)$  denotes the number of times **10** appears as a substring of  $w$ , and  $F_n$  is the  $n$ th Fibonacci number:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

Prove that each of the following languages is *not* regular.

1.  $\{\text{0}^{2^n} \mid n \geq 0\}$
2.  $\{\text{0}^{2n}\text{1}^n \mid n \geq 0\}$
3.  $\{\text{0}^m\text{1}^n \mid m \neq 2n\}$
4. Strings over  $\{\text{0}, \text{1}\}$  where the number of **0**s is exactly twice the number of **1**s.
5. Strings of properly nested parentheses **( )**, brackets **[ ]**, and braces **{ }**. For example, the string **( [ ] { } )** is in this language, but the string **( [ ] )** is not, because the left and right delimiters don't match.
6. Strings of the form  $w_1\#w_2\#\cdots\#w_n$  for some  $n \geq 2$ , where each substring  $w_i$  is a string in  $\{\text{0}, \text{1}\}^*$ , and some pair of substrings  $w_i$  and  $w_j$  are equal.

**Work on these later:**

7.  $\{\text{0}^{n^2} \mid n \geq 0\}$
8.  $\{w \in (\text{0} + \text{1})^* \mid w \text{ is the binary representation of a perfect square}\}$

Let  $L$  be an arbitrary regular language.

1. Prove that the language  $\text{insert1}(L) := \{x\mathbf{1}y \mid xy \in L\}$  is regular.

Intuitively,  $\text{insert1}(L)$  is the set of all strings that can be obtained from strings in  $L$  by inserting exactly one  $\mathbf{1}$ . For example, if  $L = \{\epsilon, \mathbf{0OK!}\}$ , then  $\text{insert1}(L) = \{\mathbf{1}, \mathbf{10OK!}, \mathbf{01OK!}, \mathbf{001K!}, \mathbf{0OK1!}, \mathbf{0OK!1}\}$ .

2. Prove that the language  $\text{delete1}(L) := \{xy \mid x\mathbf{1}y \in L\}$  is regular.

Intuitively,  $\text{delete1}(L)$  is the set of all strings that can be obtained from strings in  $L$  by deleting exactly one  $\mathbf{1}$ . For example, if  $L = \{\mathbf{101101}, \mathbf{00}, \epsilon\}$ , then  $\text{delete1}(L) = \{\mathbf{01101}, \mathbf{10101}, \mathbf{10110}\}$ .

---

**Work on these later:** (In fact, these might be easier than problems 1 and 2.)

3. Consider the following recursively defined function on strings:

$$\text{stutter}(w) := \begin{cases} \epsilon & \text{if } w = \epsilon \\ aa \bullet \text{stutter}(x) & \text{if } w = ax \text{ for some symbol } a \text{ and some string } x \end{cases}$$

Intuitively,  $\text{stutter}(w)$  doubles every symbol in  $w$ . For example:

- $\text{stutter}(\mathbf{PRESTO}) = \mathbf{PPRREESSTT00}$
- $\text{stutter}(\mathbf{HOCUS\diamondPOCUS}) = \mathbf{HHOOCCUUSS\diamond\diamond PP00CCUUSS}$

Let  $L$  be an arbitrary regular language.

- (a) Prove that the language  $\text{stutter}^{-1}(L) := \{w \mid \text{stutter}(w) \in L\}$  is regular.
- (b) Prove that the language  $\text{stutter}(L) := \{\text{stutter}(w) \mid w \in L\}$  is regular.

4. Consider the following recursively defined function on strings:

$$\text{evens}(w) := \begin{cases} \epsilon & \text{if } w = \epsilon \\ \epsilon & \text{if } w = a \text{ for some symbol } a \\ b \cdot \text{evens}(x) & \text{if } w = abx \text{ for some symbols } a \text{ and } b \text{ and some string } x \end{cases}$$

Intuitively,  $\text{evens}(w)$  skips over every other symbol in  $w$ . For example:

- $\text{evens}(\mathbf{EXPPELLIARMUS}) = \mathbf{XELAMS}$
- $\text{evens}(\mathbf{AVADA\diamondKEDAVRA}) = \mathbf{VD\diamondEAR.}$

Once again, let  $L$  be an arbitrary regular language.

- (a) Prove that the language  $\text{evens}^{-1}(L) := \{w \mid \text{evens}(w) \in L\}$  is regular.
- (b) Prove that the language  $\text{evens}(L) := \{\text{evens}(w) \mid w \in L\}$  is regular.

Alex showed the following context-free grammars in class on Tuesday; in each example, the grammar itself is on the left; the explanation for each non-terminal is on the right.

- Properly nested strings of parentheses.

$$S \rightarrow \epsilon \mid S(S) \quad \text{properly nested parentheses}$$

Here is a different grammar for the same language:

$$S \rightarrow \epsilon \mid (S) \mid SS \quad \text{properly nested parentheses}$$

- $\{\mathbf{0}^m \mathbf{1}^n \mid m \neq n\}$ . This is the set of all binary strings composed of some number of **0**s followed by a different number of **1**s.

$S \rightarrow A \mid B$	$\{\mathbf{0}^m \mathbf{1}^n \mid m \neq n\}$
$A \rightarrow \mathbf{0}A \mid \mathbf{0}C$	$\{\mathbf{0}^m \mathbf{1}^n \mid m > n\}$
$B \rightarrow B\mathbf{1} \mid C\mathbf{1}$	$\{\mathbf{0}^m \mathbf{1}^n \mid m < n\}$
$C \rightarrow \epsilon \mid \mathbf{0}C\mathbf{1}$	$\{\mathbf{0}^m \mathbf{1}^n \mid m = n\}$

Give context-free grammars for each of the following languages. For each grammar, describe *in English* the language for each non-terminal, and in the examples above. As usual, we won't get to all of these in section.

1.  $\{\mathbf{0}^{2n} \mathbf{1}^n \mid n \geq 0\}$

2.  $\{\mathbf{0}^m \mathbf{1}^n \mid m \neq 2n\}$

[Hint: If  $m \neq 2n$ , then either  $m < 2n$  or  $m > 2n$ . Extend the previous grammar, but pay attention to parity. This language contains the string **01**.]

3.  $\{\mathbf{0}, \mathbf{1}\}^* \setminus \{\mathbf{0}^{2n} \mathbf{1}^n \mid n \geq 0\}$

[Hint: Extend the previous grammar. What's missing?]

**Work on these later:**

4.  $\{w \in \{\mathbf{0}, \mathbf{1}\}^* \mid \#(\mathbf{0}, w) = 2 \cdot \#(\mathbf{1}, w)\}$  — Binary strings where the number of **0**s is exactly twice the number of **1**s.

5.  $\{\mathbf{0}, \mathbf{1}\}^* \setminus \{ww \mid w \in \{\mathbf{0}, \mathbf{1}\}^*\}$ .

[Anti-hint: The language  $\{ww \mid w \in \{\mathbf{0}, \mathbf{1}\}^*\}$  is **not** context-free. Thus, the complement of a context-free language is not necessarily context-free!]

For each of the following languages over the alphabet  $\Sigma = \{\texttt{0}, \texttt{1}\}$ , either prove the language is regular (by giving an equivalent regular expression, DFA, or NFA) or prove that the language is not regular (using a fooling set argument). Exactly half of these languages are regular.

1.  $\{\texttt{0}^n\texttt{1}\texttt{0}^n \mid n \geq 0\}$
2.  $\{\texttt{0}^n\texttt{1}\texttt{0}^n w \mid n \geq 0 \text{ and } w \in \Sigma^*\}$
3.  $\{w\texttt{0}^n\texttt{1}\texttt{0}^n x \mid w \in \Sigma^* \text{ and } n \geq 0 \text{ and } x \in \Sigma^*\}$
4. Strings in which the number of **0**s and the number of **1**s differ by at most 2.
5. Strings such that *in every prefix*, the number of **0**s and the number of **1**s differ by at most 2.
6. Strings such that *in every substring*, the number of **0**s and the number of **1**s differ by at most 2.

Design Turing machines  $M = (Q, \Sigma, \Gamma, \delta, \text{start}, \text{accept}, \text{reject})$  for each of the following tasks, either by listing the states  $Q$ , the tape alphabet  $\Gamma$ , and the transition function  $\delta$  (in a table), or by drawing the corresponding labeled graph.

Each of these machines uses the input alphabet  $\Sigma = \{\mathbf{1}, \#\}$ ; the tape alphabet  $\Gamma$  can be any superset of  $\{\mathbf{1}, \#, \square, \triangleright\}$  where  $\square$  is the blank symbol and  $\triangleright$  is a special symbol marking the left end of the tape. Each machine should **reject** any input not in the form specified below.

---

1. On input  $\mathbf{1}^n$ , for any non-negative integer  $n$ , write  $\mathbf{1}^n \# \mathbf{1}^n$  on the tape and **accept**.
2. On input  $\#^n \mathbf{1}^m$ , for any non-negative integers  $m$  and  $n$ , write  $\mathbf{1}^m$  on the tape and **accept**.  
In other words, delete all the  $\#$ s and shift the  $\mathbf{1}$ s to the start of the tape.
3. On input  $\# \mathbf{1}^n$ , for any non-negative integer  $n$ , write  $\# \mathbf{1}^{2n}$  on the tape and **accept**. [Hint: Modify the Turing machine from problem 1.]
4. On input  $\mathbf{1}^n$ , for any non-negative integer  $n$ , write  $\mathbf{1}^{2^n}$  on the tape and **accept**. [Hint: Use the three previous Turing machines as subroutines.]

Here are several problems that are easy to solve in  $O(n)$  time, essentially by brute force. Your task is to design algorithms for these problems that are significantly faster.

1. Suppose we are given an array  $A[1..n]$  of  $n$  distinct integers, which could be positive, negative, or zero, sorted in increasing order so that  $A[1] < A[2] < \dots < A[n]$ .
  - (a) Describe a fast algorithm that either computes an index  $i$  such that  $A[i] = i$  or correctly reports that no such index exists.
  - (b) Suppose we know in advance that  $A[1] > 0$ . Describe an even faster algorithm that either computes an index  $i$  such that  $A[i] = i$  or correctly reports that no such index exists. [Hint: This is **really easy**.]
2. Suppose we are given an array  $A[1..n]$  such that  $A[1] \geq A[2]$  and  $A[n-1] \leq A[n]$ . We say that an element  $A[x]$  is a **local minimum** if both  $A[x-1] \geq A[x]$  and  $A[x] \leq A[x+1]$ . For example, there are exactly six local minima in the following array:

9	7	7	2	1	3	7	5	4	7	3	3	4	8	6	9
	▲		▲				▲			▲					▲

Describe and analyze a fast algorithm that returns the index of one local minimum. For example, given the array above, your algorithm could return the integer 9, because  $A[9]$  is a local minimum. [Hint: With the given boundary conditions, any array **must** contain at least one local minimum. Why?]

3. Suppose you are given two sorted arrays  $A[1..n]$  and  $B[1..n]$  containing distinct integers. Describe a fast algorithm to find the median (meaning the  $n$ th smallest element) of the union  $A \cup B$ . For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \quad B[1..8] = [2, 4, 5, 8, 17, 19, 21, 23]$$

your algorithm should return the integer 9. [Hint: What can you learn by comparing one element of  $A$  with one element of  $B$ ?]

**To think about later:**

4. Now suppose you are given two sorted arrays  $A[1..m]$  and  $B[1..n]$  and an integer  $k$ . Describe a fast algorithm to find the  $k$ th smallest element in the union  $A \cup B$ . For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \quad B[1..5] = [2, 5, 7, 17, 19] \quad k = 6$$

your algorithm should return the integer 7.

In lecture, Alex described an algorithm of Karatsuba that multiplies two  $n$ -digit integers using  $O(n^{\lg 3})$  single-digit additions, subtractions, and multiplications. In this lab we'll look at some extensions and applications of this algorithm.

1. Describe an algorithm to compute the product of an  $n$ -digit number and an  $m$ -digit number, where  $m < n$ , in  $O(m^{\lg 3-1}n)$  time.
2. Describe an algorithm to compute the decimal representation of  $2^n$  in  $O(n^{\lg 3})$  time. (The standard algorithm that computes one digit at a time requires  $\Theta(n^2)$  time.)
3. Describe a divide-and-conquer algorithm to compute the decimal representation of an arbitrary  $n$ -bit binary number in  $O(n^{\lg 3})$  time. [Hint: Let  $x = a \cdot 2^{n/2} + b$ . Watch out for an extra log factor in the running time.]

**Think about later:**

4. Suppose we can multiply two  $n$ -digit numbers in  $O(M(n))$  time. Describe an algorithm to compute the decimal representation of an arbitrary  $n$ -bit binary number in  $O(M(n)\log n)$  time.

A **subsequence** of a sequence (for example, an array, linked list, or string), obtained by removing zero or more elements and keeping the rest in the same sequence order. A subsequence is called a **substring** if its elements are contiguous in the original sequence. For example:

- **SUBSEQUENCE**, **UBSEQU**, and the empty string  $\epsilon$  are all substrings (and therefore subsequences) of the string **SUBSEQUENCE**;
- **SBSQNC**, **SQUEE**, and **EEE** are all subsequences of **SUBSEQUENCE** but not substrings;
- **QUEUE**, **EQUUS**, and **DIMAGGIO** are not subsequences (and therefore not substrings) of **SUBSEQUENCE**.

Describe recursive backtracking algorithms for the following problems. *Don't worry about running times.*

- Given an array  $A[1..n]$  of integers, compute the length of a **longest increasing subsequence**. A sequence  $B[1..\ell]$  is *increasing* if  $B[i] > B[i - 1]$  for every index  $i \geq 2$ .

For example, given the array

$$\langle 3, \underline{1}, \underline{4}, 1, \underline{5}, 9, 2, \underline{6}, 5, 3, 5, \underline{8}, \underline{9}, 7, 9, 3, 2, 3, 8, 4, 6, 2, 7 \rangle$$

your algorithm should return the integer 6, because  $\langle 1, 4, 5, 6, 8, 9 \rangle$  is a longest increasing subsequence (one of many).

- Given an array  $A[1..n]$  of integers, compute the length of a **longest decreasing subsequence**. A sequence  $B[1..\ell]$  is *decreasing* if  $B[i] < B[i - 1]$  for every index  $i \geq 2$ .

For example, given the array

$$\langle 3, 1, 4, 1, 5, \underline{9}, 2, \underline{6}, 5, 3, \underline{5}, 8, 9, 7, 9, 3, 2, 3, 8, \underline{4}, 6, \underline{2}, 7 \rangle$$

your algorithm should return the integer 5, because  $\langle 9, 6, 5, 4, 2 \rangle$  is a longest decreasing subsequence (one of many).

- Given an array  $A[1..n]$  of integers, compute the length of a **longest alternating subsequence**. A sequence  $B[1..\ell]$  is *alternating* if  $B[i] < B[i - 1]$  for every even index  $i \geq 2$ , and  $B[i] > B[i - 1]$  for every odd index  $i \geq 3$ .

For example, given the array

$$\langle \underline{3}, \underline{1}, \underline{4}, \underline{1}, \underline{5}, 9, \underline{2}, \underline{6}, \underline{5}, 3, 5, \underline{8}, 9, \underline{7}, \underline{9}, \underline{3}, 2, 3, \underline{8}, \underline{4}, \underline{6}, \underline{2}, \underline{7} \rangle$$

your algorithm should return the integer 17, because  $\langle 3, 1, 4, 1, 5, 2, 6, 5, 8, 7, 9, 3, 8, 4, 6, 2, 7 \rangle$  is a longest alternating subsequence (one of many).

**To think about later:**

4. Given an array  $A[1..n]$  of integers, compute the length of a longest **convex** subsequence of  $A$ . A sequence  $B[1..\ell]$  is *convex* if  $B[i] - B[i-1] > B[i-1] - B[i-2]$  for every index  $i \geq 3$ .

For example, given the array

$$\langle \underline{3}, \underline{1}, 4, \underline{1}, 5, 9, \underline{2}, 6, 5, 3, \underline{5}, 8, \underline{9}, 7, 9, 3, 2, 3, 8, 4, 6, 2, 7 \rangle$$

your algorithm should return the integer 6, because  $\langle 3, 1, 1, 2, 5, 9 \rangle$  is a longest convex subsequence (one of many).

5. Given an array  $A[1..n]$ , compute the length of a longest **palindrome** subsequence of  $A$ . Recall that a sequence  $B[1..\ell]$  is a *palindrome* if  $B[i] = B[\ell - i + 1]$  for every index  $i$ .

For example, given the array

$$\langle 3, 1, \underline{4}, 1, 5, \underline{9}, 2, 6, \underline{5}, \underline{3}, \underline{5}, 8, 9, 7, \underline{9}, 3, 2, 3, 8, \underline{4}, 6, 2, 7 \rangle$$

your algorithm should return the integer 7, because  $\langle 4, 9, 5, 3, 5, 9, 4 \rangle$  is a longest palindrome subsequence (one of many).

A *subsequence* of a sequence (for example, an array, a linked list, or a string), obtained by removing zero or more elements and keeping the rest in the same sequence order. A subsequence is called a *substring* if its elements are contiguous in the original sequence. For example:

- **SUBSEQUENCE, UBSEQU**, and the empty string  $\epsilon$  are all substrings of the string **SUBSEQUENCE**;
  - **SBSQNC, UEQUE**, and **EEE** are all subsequences of **SUBSEQUENCE** but not substrings;
  - **QUEUE, SSS**, and **FOOBAR** are not subsequences of **SUBSEQUENCE**.
- 

Describe and analyze *dynamic programming* algorithms for the following problems. For the first three, use the backtracking algorithms you developed on Wednesday.

1. Given an array  $A[1..n]$  of integers, compute the length of a longest *increasing* subsequence of  $A$ . A sequence  $B[1..\ell]$  is *increasing* if  $B[i] > B[i-1]$  for every index  $i \geq 2$ .
2. Given an array  $A[1..n]$  of integers, compute the length of a longest *decreasing* subsequence of  $A$ . A sequence  $B[1..\ell]$  is *decreasing* if  $B[i] < B[i-1]$  for every index  $i \geq 2$ .
3. Given an array  $A[1..n]$  of integers, compute the length of a longest *alternating* subsequence of  $A$ . A sequence  $B[1..\ell]$  is *alternating* if  $B[i] < B[i-1]$  for every even index  $i \geq 2$ , and  $B[i] > B[i-1]$  for every odd index  $i \geq 3$ .
4. Given an array  $A[1..n]$  of integers, compute the length of a longest *convex* subsequence of  $A$ . A sequence  $B[1..\ell]$  is *convex* if  $B[i] - B[i-1] > B[i-1] - B[i-2]$  for every index  $i \geq 3$ .
5. Given an array  $A[1..n]$ , compute the length of a longest *palindrome* subsequence of  $A$ . Recall that a sequence  $B[1..\ell]$  is a *palindrome* if  $B[i] = B[\ell-i+1]$  for every index  $i$ .

## Basic steps in developing a dynamic programming algorithm

1. **Formulate the problem recursively.** This is the hard part. There are two distinct but equally important things to include in your formulation.
  - (a) **Specification.** First, give a clear and precise English description of the problem you are claiming to solve. Not *how* to solve the problem, but *what* the problem actually is. Omitting this step in homeworks or exams is an automatic zero.
  - (b) **Solution.** Second, give a clear recursive formula or algorithm for the whole problem in terms of the answers to smaller instances of *exactly* the same problem. It generally helps to think in terms of a recursive definition of your inputs and outputs. If you discover that you need a solution to a *similar* problem, or a slightly *related* problem, you're attacking the wrong problem; go back to step 1.
2. **Build solutions to your recurrence from the bottom up.** Write an algorithm that starts with the base cases of your recurrence and works its way up to the final solution, by considering intermediate subproblems in the correct order. This stage can be broken down into several smaller, relatively mechanical steps:
  - (a) **Identify the subproblems.** What are all the different ways can your recursive algorithm call itself, starting with some initial input?
  - (b) **Analyze running time.** Add up the running times of all possible subproblems, *ignoring the recursive calls*.
  - (c) **Choose a memoization data structure.** For most problems, each recursive subproblem can be identified by a few integers, so you can use a multidimensional array. But some problems need a more complicated data structure.
  - (d) **Identify dependencies.** Except for the base cases, every recursive subproblem depends on other subproblems—which ones? Draw a picture of your data structure, pick a generic element, and draw arrows from each of the other elements it depends on. Then formalize your picture.
  - (e) **Find a good evaluation order.** Order the subproblems so that each subproblem comes *after* the subproblems it depends on. Typically, you should consider the base cases first, then the subproblems that depends only on base cases, and so on. *Be careful!*
  - (f) **Write down the algorithm.** You know what order to consider the subproblems, and you know how to solve each subproblem. So do that! If your data structure is an array, this usually means writing a few nested for-loops around your original recurrence.

Lenny Rutenbar, the founding dean of the new Maximilian Q. Levchin College of Computer Science, has commissioned a series of snow ramps on the south slope of the Orchard Downs sledding hill<sup>□</sup> and challenged Bill Kudeki, head of the Department of Electrical and Computer Engineering, to a sledding contest. Bill and Lenny will both sled down the hill, each trying to maximize their air time. The winner gets to expand their department/college into both Siebel Center and the new ECE Building; the loser has to move their entire department/college under the Boneyard bridge next to Everitt Lab.

Whenever Lenny or Bill reaches a ramp *while on the ground*, they can either use that ramp to jump through the air, possibly flying over one or more ramps, or sled past that ramp and stay on the ground. Obviously, if someone flies over a ramp, they cannot use that ramp to extend their jump.

1. Suppose you are given a pair of arrays  $Ramp[1..n]$  and  $Length[1..n]$ , where  $Ramp[i]$  is the distance from the top of the hill to the  $i$ th ramp, and  $Length[i]$  is the distance that any sledder who takes the  $i$ th ramp will travel through the air.

Describe and analyze an algorithm to determine the maximum total distance that Lenny or Bill can spend in the air.

2. Uh-oh. The university lawyers heard about Lenny and Bill's little bet and immediately objected. To protect the university from either lawsuits or sky-rocketing insurance rates, they impose an upper bound on the number of jumps that either sledder can take.

Describe and analyze an algorithm to determine the maximum total distance that Lenny or Bill can spend in the air *with at most  $k$  jumps*, given the original arrays  $Ramp[1..n]$  and  $Length[1..n]$  and the integer  $k$  as input.

3. **To think about later:** When the lawyers realized that imposing their restriction didn't immediately shut down the contest, they added a new restriction: No ramp can be used more than once! Disgusted by the legal interference, Lenny and Bill give up on their bet and decide to cooperate to put on a good show for the spectators.

Describe and analyze an algorithm to determine the maximum total distance that Lenny and Bill can spend in the air, each taking at most  $k$  jumps (so at most  $2k$  jumps total), and with each ramp used at most once.

---

<sup>□</sup>The north slope is faster, but too short for an interesting contest.

1. A **basic arithmetic expression** is composed of characters from the set  $\{1, +, \times\}$  and parentheses. Almost every integer can be represented by more than one basic arithmetic expression. For example, all of the following basic arithmetic expression represent the integer 14:

$$\begin{aligned} & 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 \\ & ((1 + 1) \times (1 + 1 + 1 + 1 + 1)) + ((1 + 1) \times (1 + 1)) \\ & (1 + 1) \times (1 + 1 + 1 + 1 + 1 + 1 + 1) \\ & (1 + 1) \times (((1 + 1 + 1) \times (1 + 1)) + 1) \end{aligned}$$

Describe and analyze an algorithm to compute, given an integer  $n$  as input, the minimum number of 1's in a basic arithmetic expression whose value is equal to  $n$ . The number of parentheses doesn't matter, just the number of 1's. For example, when  $n = 14$ , your algorithm should return 8, for the final expression above. The running time of your algorithm should be bounded by a small polynomial function of  $n$ .

2. **To think about later:** Suppose you are given a sequence of integers separated by  $+$  and  $-$  signs; for example:

$$1 + 3 - 2 - 5 + 1 - 6 + 7$$

You can change the value of this expression by adding parentheses in different places. For example:

$$\begin{aligned} & 1 + 3 - 2 - 5 + 1 - 6 + 7 = -1 \\ & (1 + 3 - (2 - 5)) + (1 - 6) + 7 = 9 \\ & (1 + (3 - 2)) - (5 + 1) - (6 + 7) = -17 \end{aligned}$$

Describe and analyze an algorithm to compute, given a list of integers separated by  $+$  and  $-$  signs, the maximum possible value the expression can take by adding parentheses. Parentheses must be used only to group additions and subtractions; in particular, do not use them to create implicit multiplication as in  $1 + 3(-2)(-5) + 1 - 6 + 7 = 33$ .

Recall the class scheduling problem described in lecture on Tuesday. We are given two arrays  $S[1..n]$  and  $F[1..n]$ , where  $S[i] < F[i]$  for each  $i$ , representing the start and finish times of  $n$  classes. Your goal is to find the largest number of classes you can take without ever taking two classes simultaneously. We showed in class that the following greedy algorithm constructs an optimal schedule:

Choose the course that *ends first*, discard all conflicting classes, and recurse.

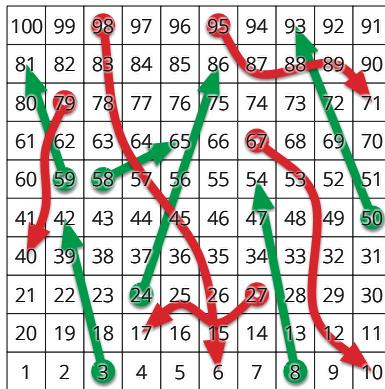
But this is not the only greedy strategy we could have tried. For each of the following alternative greedy algorithms, either prove that the algorithm always constructs an optimal schedule, or describe a small input example for which the algorithm does not produce an optimal schedule. Assume that all algorithms break ties arbitrarily (that is, in a manner that is completely out of your control). ***Exactly three of these greedy strategies actually work.***

1. Choose the course  $x$  that *ends last*, discard classes that conflict with  $x$ , and recurse.
2. Choose the course  $x$  that *starts first*, discard all classes that conflict with  $x$ , and recurse.
3. Choose the course  $x$  that *starts last*, discard all classes that conflict with  $x$ , and recurse.
4. Choose the course  $x$  with *shortest duration*, discard all classes that conflict with  $x$ , and recurse.
5. Choose a course  $x$  that *conflicts with the fewest other courses*, discard all classes that conflict with  $x$ , and recurse.
6. If no classes conflict, choose them all. Otherwise, discard the course with *longest duration* and recurse.
7. If no classes conflict, choose them all. Otherwise, discard a course that *conflicts with the most other courses* and recurse.
8. Let  $x$  be the class with the *earliest start time*, and let  $y$  be the class with the *second earliest start time*.
  - If  $x$  and  $y$  are disjoint, choose  $x$  and recurse on everything but  $x$ .
  - If  $x$  completely contains  $y$ , discard  $x$  and recurse.
  - Otherwise, discard  $y$  and recurse.
9. If any course  $x$  completely contains another course, discard  $x$  and recurse. Otherwise, choose the course  $y$  that *ends last*, discard all classes that conflict with  $y$ , and recurse.

For each of the problems below, transform the input into a graph and apply a standard graph algorithm that you've seen in class. Whenever you use a standard graph algorithm, you **must** provide the following information. (I recommend actually using a bulleted list.)

- What are the vertices?
- What are the edges? Are they directed or undirected?
- If the vertices and/or edges have associated values, what are they?
- What problem do you need to solve on this graph?
- What standard algorithm are you using to solve that problem?
- What is the running time of your entire algorithm, *including* the time to build the graph, as a function of the original input parameters?

1. **Snakes and Ladders** is a classic board game, originating in India no later than the 16th century. The board consists of an  $n \times n$  grid of squares, numbered consecutively from 1 to  $n^2$ , starting in the bottom left corner and proceeding row by row from bottom to top, with rows alternating to the left and right. Certain pairs of squares, always in different rows, are connected by either “snakes” (leading down) or “ladders” (leading up). Each square can be an endpoint of at most one snake or ladder.



A typical Snakes and Ladders board.

Upward straight arrows are ladders; downward wavy arrows are snakes.

You start with a token in cell 1, in the bottom left corner. In each move, you advance your token up to  $k$  positions, for some fixed constant  $k$  (typically 6). If the token ends the move at the *top* end of a snake, you **must** slide the token down to the bottom of that snake. If the token ends the move at the *bottom* end of a ladder, you **may** move the token up to the top of that ladder.

Describe and analyze an algorithm to compute the smallest number of moves required for the token to reach the last square of the grid.

2. Let  $G$  be a connected undirected graph. Suppose we start with two coins on two arbitrarily chosen vertices of  $G$ . At every step, each coin *must* move to an adjacent vertex. Describe and analyze an algorithm to compute the minimum number of steps to reach a configuration where both coins are on the same vertex, or to report correctly that no such configuration is reachable. The input to your algorithm consists of a graph  $G = (V, E)$  and two vertices  $u, v \in V$  (which may or may not be distinct).

For each of the problems below, transform the input into a graph and apply a standard graph algorithm that you've seen in class. Whenever you use a standard graph algorithm, you **must** provide the following information. (I recommend actually using a bulleted list.)

- What are the vertices?
  - What are the edges? Are they directed or undirected?
  - If the vertices and/or edges have associated values, what are they?
  - What problem do you need to solve on this graph?
  - What standard algorithm are you using to solve that problem?
  - What is the running time of your entire algorithm, *including* the time to build the graph, *as a function of the original input parameters?*
- 

1. Inspired by the previous lab, you decide to organize a Snakes and Ladders competition with  $n$  participants. In this competition, each game of Snakes and Ladders involves three players. After the game is finished, they are ranked first, second, and third. Each player may be involved in any (non-negative) number of games, and the number need not be equal among players.

At the end of the competition,  $m$  games have been played. You realize that you forgot to implement a proper rating system, and therefore decide to produce the overall ranking of all  $n$  players as you see fit. However, to avoid being too suspicious, if player  $A$  ranked better than player  $B$  in any game, then  $A$  must rank better than  $B$  in the overall ranking.

You are given the list of players and their ranking in each of the  $m$  games. Describe and analyze an algorithm that produces an overall ranking of the  $n$  players that is consistent with the individual game rankings, or correctly reports that no such ranking exists.

2. There are  $n$  galaxies connected by  $m$  intergalactic teleport-ways. Each teleport-way joins two galaxies and can be traversed in both directions. However, the company that runs the teleport-ways has established an extremely lucrative cost structure: Anyone can teleport *further* from their home galaxy at no cost whatsoever, but teleporting *toward* their home galaxy is prohibitively expensive.

Judy has decided to take a sabbatical tour of the universe by visiting as many galaxies as possible, starting at her home galaxy. To save on travel expenses, she wants to teleport away from her home galaxy at every step, except for the very last teleport home.

Describe and analyze an algorithm to compute the maximum number of galaxies that Judy can visit. Your input consists of an undirected graph  $G$  with  $n$  vertices and  $m$  edges describing the teleport-way network, an integer  $1 \leq s \leq n$  identifying Judy's home galaxy, and an array  $D[1..n]$  containing the distances of each galaxy from  $s$ .

#### To think about later:

3. Just before embarking on her universal tour, Judy wins the space lottery, giving her just enough money to afford *two* teleports toward her home galaxy. Describe a new algorithm to compute the maximum number of distinct galaxies Judy can visit. She can visit the same galaxy more than once, but only the first visit counts toward her total.

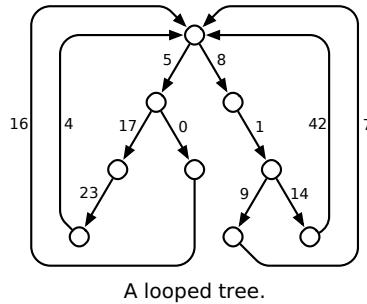
1. Describe and analyze an algorithm to compute the shortest path from vertex  $s$  to vertex  $t$  in a directed graph with weighted edges, where exactly *one* edge  $u \rightarrow v$  has negative weight. Assume the graph has no negative cycles. [Hint: Modify the input graph and run Dijkstra's algorithm. Alternatively, don't modify the input graph, but run Dijkstra's algorithm anyway.]
2. You just discovered your best friend from elementary school on Twitbook. You both want to meet as soon as possible, but you live in two different cities that are far apart. To minimize travel time, you agree to meet at an intermediate city, and then you simultaneously hop in your cars and start driving toward each other. But where *exactly* should you meet?

You are given a weighted graph  $G = (V, E)$ , where the vertices  $V$  represent cities and the edges  $E$  represent roads that directly connect cities. Each edge  $e$  has a weight  $w(e)$  equal to the time required to travel between the two cities. You are also given a vertex  $p$ , representing your starting location, and a vertex  $q$ , representing your friend's starting location.

Describe and analyze an algorithm to find the target vertex  $t$  that allows you and your friend to meet as soon as possible, assuming both of you leave home *right now*.

#### To think about later:

3. A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has a non-negative weight.



- (a) How much time would Dijkstra's algorithm require to compute the shortest path between two vertices  $u$  and  $v$  in a looped tree with  $n$  nodes?
- (b) Describe and analyze a faster algorithm.

1. Suppose that you have just finished computing the array  $dist[1..V, 1..V]$  of shortest-path distances between *all* pairs of vertices in an edge-weighted directed graph  $G$ . Unfortunately, you discover that you incorrectly entered the weight of a single edge  $u \rightarrow v$ , so all that precious CPU time was wasted. Or was it? Maybe your distances are correct after all!

In each of the following problems, let  $w(u \rightarrow v)$  denote the weight that you used in your distance computation, and let  $w'(u \rightarrow v)$  denote the correct weight of  $u \rightarrow v$ .

- (a) Suppose  $w(u \rightarrow v) > w'(u \rightarrow v)$ ; that is, the weight you used for  $u \rightarrow v$  was *larger* than its true weight. Describe an algorithm that repairs the distance array in  $O(V^2)$  time under this assumption. [Hint: For every pair of vertices  $x$  and  $y$ , either  $u \rightarrow v$  is on the shortest path from  $x$  to  $y$  or it isn't.]
  - (b) Maybe even that was too much work. Describe an algorithm that determines whether your original distance array is actually correct in  $O(1)$  time, again assuming that  $w(u \rightarrow v) > w'(u \rightarrow v)$ . [Hint: Either  $u \rightarrow v$  is the shortest path from  $u$  to  $v$  or it isn't.]
  - (c) **To think about later:** Describe an algorithm that determines in  $O(VE)$  time whether your distance array is actually correct, even if  $w(u \rightarrow v) < w'(u \rightarrow v)$ .
  - (d) **To think about later:** Argue that when  $w(u \rightarrow v) < w'(u \rightarrow v)$ , repairing the distance array *requires* recomputing shortest paths from scratch, at least in the worst case.
2. You—yes, *you*—can cause a major economic collapse with the power of graph algorithms! □ The *arbitrage* business is a money-making scheme that takes advantage of differences in currency exchange. In particular, suppose that 1 US dollar buys 120 Japanese yen; 1 yen buys 0.01 euros; and 1 euro buys 1.2 US dollars. Then, a trader starting with \$1 can convert their money from dollars to yen, then from yen to euros, and finally from euros back to dollars, ending with \$1.44! The cycle of currencies  $\$ \rightarrow ¥ \rightarrow € \rightarrow \$$  is called an *arbitrage cycle*. Of course, finding and exploiting arbitrage cycles before the prices are corrected requires extremely fast algorithms.

Suppose  $n$  different currencies are traded in your currency market. You are given the matrix  $R[1..n]$  of exchange rates between every pair of currencies; for each  $i$  and  $j$ , one unit of currency  $i$  can be traded for  $R[i, j]$  units of currency  $j$ . (Do *not* assume that  $R[i, j] \cdot R[j, i] = 1$ .)

- (a) Describe an algorithm that returns an array  $V[1..n]$ , where  $V[i]$  is the maximum amount of currency  $i$  that you can obtain by trading, starting with one unit of currency 1, assuming there are no arbitrage cycles.
- (b) Describe an algorithm to determine whether the given matrix of currency exchange rates creates an arbitrage cycle.
- \*(c) **To think about later:** Modify your algorithm from part (b) to actually return an arbitrage cycle, if such a cycle exists.

---

□No, you can't.

1. Suppose you are given a magic black box that somehow answers the following decision problem in *polynomial time*:

- INPUT: A boolean circuit  $K$  with  $n$  inputs and one output.
- OUTPUT: TRUE if there are input values  $x_1, x_2, \dots, x_n \in \{\text{TRUE}, \text{FALSE}\}$  that make  $K$  output TRUE, and FALSE otherwise.

Using this black box as a subroutine, describe an algorithm that solves the following related search problem in *polynomial time*:

- INPUT: A boolean circuit  $K$  with  $n$  inputs and one output.
- OUTPUT: Input values  $x_1, x_2, \dots, x_n \in \{\text{TRUE}, \text{FALSE}\}$  that make  $K$  output TRUE, or NONE if there are no such inputs.

[Hint: You can use the magic box more than once.]

2. An **independent set** in a graph  $G$  is a subset  $S$  of the vertices of  $G$ , such that no two vertices in  $S$  are connected by an edge in  $G$ . Suppose you are given a magic black box that somehow answers the following decision problem in *polynomial time*:

- INPUT: An undirected graph  $G$  and an integer  $k$ .
- OUTPUT: TRUE if  $G$  has an independent set of size  $k$ , and FALSE otherwise.

- (a) Using this black box as a subroutine, describe algorithms that solves the following optimization problem in *polynomial time*:

- INPUT: An undirected graph  $G$ .
- OUTPUT: The size of the largest independent set in  $G$ .

[Hint: You've seen this problem before.]

- (b) Using this black box as a subroutine, describe algorithms that solves the following search problem in *polynomial time*:

- INPUT: An undirected graph  $G$ .
- OUTPUT: An independent set in  $G$  of maximum size.

**To think about later:**

3. Formally, a **proper coloring** of a graph  $G = (V, E)$  is a function  $c: V \rightarrow \{1, 2, \dots, k\}$ , for some integer  $k$ , such that  $c(u) \neq c(v)$  for all  $uv \in E$ . Less formally, a valid coloring assigns each vertex of  $G$  a color, such that every edge in  $G$  has endpoints with different colors. The **chromatic number** of a graph is the minimum number of colors in a proper coloring of  $G$ .

Suppose you are given a magic black box that somehow answers the following decision problem *in polynomial time*:

- INPUT: An undirected graph  $G$  and an integer  $k$ .
- OUTPUT: TRUE if  $G$  has a proper coloring with  $k$  colors, and FALSE otherwise.

Using this black box as a subroutine, describe an algorithm that solves the following **coloring problem** *in polynomial time*:

- INPUT: An undirected graph  $G$ .
- OUTPUT: A valid coloring of  $G$  using the minimum possible number of colors.

[Hint: You can use the magic box more than once. The input to the magic box is a graph and **only** a graph, meaning **only** vertices and edges.]

Proving that a problem  $X$  is NP-hard requires several steps:

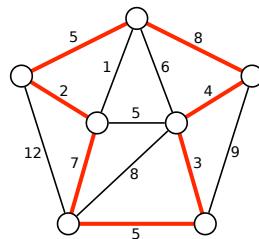
- Choose a problem  $Y$  that you already know is NP-hard (because we told you so in class).
  - Describe an algorithm to solve  $Y$ , using an algorithm for  $X$  as a subroutine. Typically this algorithm has the following form: Given an instance of  $Y$ , transform it into an instance of  $X$ , and then call the magic black-box algorithm for  $X$ .
  - **Prove** that your algorithm is correct. This always requires two separate steps, which are usually of the following form:
    - **Prove** that your algorithm transforms “good” instances of  $Y$  into “good” instances of  $X$ .
    - **Prove** that your algorithm transforms “bad” instances of  $Y$  into “bad” instances of  $X$ . Equivalently: Prove that if your transformation produces a “good” instance of  $X$ , then it was given a “good” instance of  $Y$ .
  - Argue that your algorithm for  $Y$  runs in polynomial time.
- 

1. Recall the following  $k$ COLOR problem: Given an undirected graph  $G$ , can its vertices be colored with  $k$  colors, so that every edge touches vertices with two different colors?
  - (a) Describe a direct polynomial-time reduction from 3COLOR to 4COLOR.
  - (b) Prove that  $k$ COLOR problem is NP-hard for any  $k \geq 3$ .
2. A *Hamiltonian cycle* in a graph  $G$  is a cycle that goes through every vertex of  $G$  exactly once. Deciding whether an arbitrary graph contains a Hamiltonian cycle is NP-hard.
 

A **tonian cycle** in a graph  $G$  is a cycle that goes through at least *half* of the vertices of  $G$ . Prove that deciding whether a graph contains a tonian cycle is NP-hard.

#### To think about later:

3. Let  $G$  be an undirected graph with weighted edges. A Hamiltonian cycle in  $G$  is **heavy** if the total weight of edges in the cycle is at least half of the total weight of all edges in  $G$ . Prove that deciding whether a graph contains a heavy Hamiltonian cycle is NP-hard.

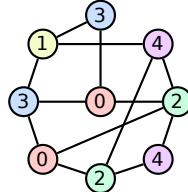


A heavy Hamiltonian cycle. The cycle has total weight 34; the graph has total weight 67.

Prove that each of the following problems is NP-hard.

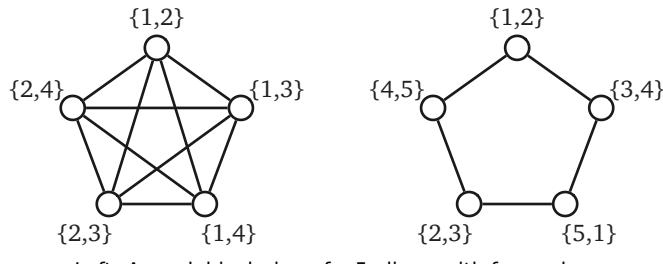
1. Given an undirected graph  $G$ , does  $G$  contain a simple path that visits all but 374 vertices?
2. Given an undirected graph  $G$ , does  $G$  have a spanning tree in which every node has degree at most 374?
3. Given an undirected graph  $G$ , does  $G$  have a spanning tree with at most 374 leaves?

1. Recall that a 5-coloring of a graph  $G$  is a function that assigns each vertex of  $G$  a “color” from the set  $\{0, 1, 2, 3, 4\}$ , such that for any edge  $uv$ , vertices  $u$  and  $v$  are assigned different “colors”. A 5-coloring is *careful* if the colors assigned to adjacent vertices are not only distinct, but differ by more than 1 (mod 5). Prove that deciding whether a given graph has a careful 5-coloring is NP-hard. [Hint: Reduce from the standard 5COLOR problem.]



A careful 5-coloring.

2. Prove that the following problem is NP-hard: Given an undirected graph  $G$ , find *any* integer  $k > 374$  such that  $G$  has a proper coloring with  $k$  colors but  $G$  does not have a proper coloring with  $k - 374$  colors.
3. A **bicoloring** of an undirected graph assigns each vertex a set of two colors. There are two types of bicoloring: In a *weak* bicoloring, the endpoints of each edge must use *different* sets of colors; however, these two sets may share one color. In a *strong* bicoloring, the endpoints of each edge must use *distinct* sets of colors; that is, they must use four colors altogether. Every strong bicoloring is also a weak bicoloring.
- Prove that finding the minimum number of colors in a weak bicoloring of a given graph is NP-hard.
  - Prove that finding the minimum number of colors in a strong bicoloring of a given graph is NP-hard.



Left: A weak bicoloring of a 5-clique with four colors.  
Right: A strong bicoloring of a 5-cycle with five colors.

Proving that a language  $L$  is undecidable by reduction requires several steps. (These are the essentially the same steps you already use to prove that a problem is NP-hard.)

- Choose a language  $L'$  that you already know is undecidable (because we told you so in class). The simplest choice is usually the standard halting language

$$\text{HALT} := \{\langle M, w \rangle \mid M \text{ halts on } w\}$$

- Describe an algorithm that decides  $L'$ , using an algorithm that decides  $L$  as a black box. Typically your reduction will have the following form:

Given an arbitrary string  $x$ , construct a special string  $y$ ,  
such that  $y \in L$  if and only if  $x \in L'$ .

In particular, if  $L = \text{HALT}$ , your reduction will have the following form:

Given the encoding  $\langle M, w \rangle$  of a Turing machine  $M$  and a string  $w$ ,  
construct a special string  $y$ , such that  
 $y \in L$  if and only if  $M$  halts on input  $w$ .

- Prove that your algorithm is correct. This proof almost always requires two separate steps:
  - Prove that if  $x \in L'$  then  $y \in L$ .
  - Prove that if  $x \notin L'$  then  $y \notin L$ .

**Very important:** Name every object in your proof, and *always* refer to objects by their names. Never refer to “the Turing machine” or “the algorithm” or “the input string” or (god forbid) “it” or “this”. Even in casual conversation, even if you’re “just” explaining your intuition, even when you’re just *thinking* about the reduction.

---

Prove that the following languages are undecidable.

1. ACCEPTILLINI :=  $\{\langle M \rangle \mid M \text{ accepts the string } \text{ILLINI}\}$
2. ACCEPTTHREE :=  $\{\langle M \rangle \mid M \text{ accepts exactly three strings}\}$
3. ACCEPTPALINDROME :=  $\{\langle M \rangle \mid M \text{ accepts at least one palindrome}\}$

**Solution (for problem 1):** For the sake of argument, suppose there is an algorithm DECIDEACCEPTILLINI that correctly decides the language ACCEPTILLINI. Then we can solve the halting problem as follows:

```

DECIDEHALT( $\langle M, w \rangle$ ):
  Encode the following Turing machine  $M'$ :
     $M'(x)$ :
      run  $M$  on input  $w$ 
      return TRUE
  if DECIDEACCEPTILLINI( $\langle M' \rangle$ )
    return TRUE
  else
    return FALSE

```

We prove this reduction correct as follows:

$\implies$  Suppose  $M$  halts on input  $w$ .

Then  $M'$  accepts *every* input string  $x$ .

In particular,  $M'$  accepts the string **ILLINI**.

So DECIDEACCEPTILLINI accepts the encoding  $\langle M' \rangle$ .

So DECIDEHALT correctly accepts the encoding  $\langle M, w \rangle$ .

$\impliedby$  Suppose  $M$  does not halt on input  $w$ .

Then  $M'$  diverges on *every* input string  $x$ .

In particular,  $M'$  does not accept the string **ILLINI**.

So DECIDEACCEPTILLINI rejects the encoding  $\langle M' \rangle$ .

So DECIDEHALT correctly rejects the encoding  $\langle M, w \rangle$ .

In both cases, DECIDEHALT is correct. But that's impossible, because HALT is undecidable. We conclude that the algorithm DECIDEACCEPTILLINI does not exist.  $\blacksquare$

As usual for undecidability proofs, this proof invokes *four* distinct Turing machines:

- The hypothetical algorithm DECIDEACCEPTILLINI.
- The new algorithm DECIDEHALT that we construct in the solution.
- The arbitrary machine  $M$  whose encoding is part of the input to DECIDEHALT.
- The special machine  $M'$  whose encoding DECIDEHALT constructs (from the encoding of  $M$  and  $w$ ) and then passes to DECIDEACCEPTILLINI.

**Rice's Theorem.** Let  $\mathcal{L}$  be any set of languages that satisfies the following conditions:

- There is a Turing machine  $Y$  such that  $\text{ACCEPT}(Y) \in \mathcal{L}$ .
- There is a Turing machine  $N$  such that  $\text{ACCEPT}(N) \notin \mathcal{L}$ .

The language  $\text{ACCEPTIN}(\mathcal{L}) := \{\langle M \rangle \mid \text{ACCEPT}(M) \in \mathcal{L}\}$  is undecidable.

---

Prove that the following languages are undecidable *using Rice's Theorem*:

1.  $\text{ACCEPTREGULAR} := \{\langle M \rangle \mid \text{ACCEPT}(M) \text{ is regular}\}$
2.  $\text{ACCEPTILLINI} := \{\langle M \rangle \mid M \text{ accepts the string } \text{ILLINI}\}$
3.  $\text{ACCEPTPALINDROME} := \{\langle M \rangle \mid M \text{ accepts at least one palindrome}\}$
4.  $\text{ACCEPTTHREE} := \{\langle M \rangle \mid M \text{ accepts exactly three strings}\}$
5.  $\text{ACCEPTUNDECIDABLE} := \{\langle M \rangle \mid \text{ACCEPT}(M) \text{ is undecidable}\}$

**To think about later.** Which of the following are undecidable? How would you prove that?

1.  $\text{ACCEPT}\{\{\varepsilon\}\} := \{\langle M \rangle \mid M \text{ accepts only the string } \varepsilon; \text{ that is, } \text{ACCEPT}(M) = \{\varepsilon\}\}$
2.  $\text{ACCEPT}\{\emptyset\} := \{\langle M \rangle \mid M \text{ does not accept any strings; that is, } \text{ACCEPT}(M) = \emptyset\}$
3.  $\text{ACCEPT}\emptyset := \{\langle M \rangle \mid \text{ACCEPT}(M) \text{ is not an acceptable language}\}$
4.  $\text{ACCEPT}=\text{REJECT} := \{\langle M \rangle \mid \text{ACCEPT}(M) = \text{REJECT}(M)\}$
5.  $\text{ACCEPT}\neq\text{REJECT} := \{\langle M \rangle \mid \text{ACCEPT}(M) \neq \text{REJECT}(M)\}$
6.  $\text{ACCEPT}\cup\text{REJECT} := \{\langle M \rangle \mid \text{ACCEPT}(M) \cup \text{REJECT}(M) = \Sigma^*\}$

**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

1. For each statement below, check “True” if the statement is *always* true and “False” otherwise. Each correct answer is worth +1 point; each incorrect answer is worth  $-1/2$  point; checking “I don’t know” is worth  $+1/4$  point; and flipping a coin is (on average) worth  $+1/4$  point. You do **not** need to prove your answer is correct.

**Read each statement very carefully.** Some of these are deliberately subtle.

- (a) If the moon is made of cheese, then Jeff is the Queen of England.
  - (b) The language  $\{\text{0}^m \text{1}^n \mid m, n \geq 0\}$  is not regular.
  - (c) For all languages  $L$ , the language  $L^*$  is regular.
  - (d) For all languages  $L \subset \Sigma^*$ , if  $L$  is recognized by a DFA, then  $\Sigma^* \setminus L$  can be represented by a regular expression.
  - (e) For all languages  $L$  and  $L'$ , if  $L \cap L' = \emptyset$  and  $L'$  is not regular, then  $L$  is regular.
  - (f) For all languages  $L$ , if  $L$  is not regular, then  $L$  does not have a finite fooling set.
  - (g) Let  $M = (\Sigma, Q, s, A, \delta)$  and  $M' = (\Sigma, Q, s, Q \setminus A, \delta)$  be arbitrary DFAs with identical alphabets, states, starting states, and transition functions, but with complementary accepting states. Then  $L(M) \cap L(M') = \emptyset$ .
  - (h) Let  $M = (\Sigma, Q, s, A, \delta)$  and  $M' = (\Sigma, Q, s, Q \setminus A, \delta)$  be arbitrary NFAs with identical alphabets, states, starting states, and transition functions, but with complementary accepting states. Then  $L(M) \cap L(M') = \emptyset$ .
  - (i) For all context-free languages  $L$  and  $L'$ , the language  $L \bullet L'$  is also context-free.
  - (j) Every non-context-free language is non-regular.
- 
2. For each of the following languages over the alphabet  $\Sigma = \{\text{0}, \text{1}\}$ , either *prove* that the language is regular or *prove* that the language is not regular. *Exactly one of these two languages is regular.*
- (a)  $\{\text{0}^n w \text{0}^n \mid w \in \Sigma^+ \text{ and } n > 0\}$
  - (b)  $\{w \text{0}^n w \mid w \in \Sigma^+ \text{ and } n > 0\}$

For example, both of these languages contain the string **00110100000110100**.

3. Let  $L = \{\texttt{0}^{2i}\texttt{1}^{i+2j}\texttt{0}^j \mid i, j \geq 0\}$  and let  $G$  be the following context free-grammar:

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow \epsilon \mid \texttt{00A1} \\ B &\rightarrow \epsilon \mid \texttt{11B0} \end{aligned}$$

- (a) **Prove** that  $L(G) \subseteq L$ .
- (b) **Prove** that  $L \subseteq L(G)$ .

[Hint: What are  $L(A)$  and  $L(B)$ ? Prove it!]

4. For any language  $L$ , let  $\text{SUFFIXES}(L) := \{x \mid yx \in L \text{ for some } y \in \Sigma^*\}$  be the language containing all suffixes of all strings in  $L$ . For example, if  $L = \{\texttt{010}, \texttt{101}, \texttt{110}\}$ , then  $\text{SUFFIXES}(L) = \{\epsilon, \texttt{0}, \texttt{1}, \texttt{01}, \texttt{10}, \texttt{010}, \texttt{101}, \texttt{110}\}$ .

**Prove** that for any regular language  $L$ , the language  $\text{SUFFIXES}(L)$  is also regular.

5. For each of the following languages  $L$ , give a regular expression that represents  $L$  **and** describe a DFA that recognizes  $L$ .

- (a) The set of all strings in  $\{\texttt{0}, \texttt{1}\}^*$  that do not contain the substring  $\texttt{0110}$ .
- (b) The set of all strings in  $\{\texttt{0}, \texttt{1}\}^*$  that contain exactly one of the substrings  $\texttt{01}$  or  $\texttt{10}$ .

You do **not** need to prove that your answers are correct.

**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

1. For each statement below, check “True” if the statement is *always* true and “False” otherwise. Each correct answer is worth +1 point; each incorrect answer is worth  $-1/2$  point; checking “I don’t know” is worth  $+1/4$  point; and flipping a coin is (on average) worth  $+1/4$  point. You do **not** need to prove your answer is correct.

**Read each statement very carefully.** Some of these are deliberately subtle.

- (a) If  $2 + 2 = 5$ , then Jeff is the Queen of England.
  - (b) The language  $\{\text{0}^m \# \text{0}^n \mid m, n \geq 0\}$  is regular.
  - (c) For all languages  $L$ , the language  $L^*$  is regular.
  - (d) For all languages  $L \subset \Sigma^*$ , if  $L$  cannot be recognized by a DFA, then  $\Sigma^* \setminus L$  cannot be represented by a regular expression.
  - (e) For all languages  $L$  and  $L'$ , if  $L \cap L' = \emptyset$  and  $L'$  is regular, then  $L$  is regular.
  - (f) For all languages  $L$ , if  $L$  has a finite fooling set, then  $L$  is regular.
  - (g) Let  $M = (\Sigma, Q, s, A, \delta)$  and  $M' = (\Sigma, Q, s, Q \setminus A, \delta)$  be arbitrary NFAs with identical alphabets, states, starting states, and transition functions, but with complementary accepting states. Then  $L(M) \cup L(M') = \Sigma^*$ .
  - (h) Let  $M = (\Sigma, Q, s, A, \delta)$  and  $M' = (\Sigma, Q, s, Q \setminus A, \delta)$  be arbitrary NFAs with identical alphabets, states, starting states, and transition functions, but with complementary accepting states. Then  $L(M) \cap L(M') = \emptyset$ .
  - (i) For all context-free languages  $L$  and  $L'$ , the language  $L \bullet L'$  is also context-free.
  - (j) Every non-regular language is context-free.
- 
2. For each of the following languages over the alphabet  $\Sigma = \{\text{0}, \text{1}\}$ , either *prove* that the language is regular or *prove* that the language is not regular. *Exactly one of these two languages is regular.*
- (a)  $\{w\text{0}^n w \mid w \in \Sigma^+ \text{ and } n > 0\}$
  - (b)  $\{\text{0}^n w \text{0}^n \mid w \in \Sigma^+ \text{ and } n > 0\}$

For example, both of these languages contain the string **00110100000110100**.

3. Let  $L = \{1^m 0^n \mid m \leq n \leq 2m\}$  and let  $G$  be the following context free-grammar:

$$S \rightarrow 1S0 \mid 1S00 \mid \epsilon$$

- (a) **Prove** that  $L(G) \subseteq L$ .
- (b) **Prove** that  $L \subseteq L(G)$ .

4. For any language  $L$ , let  $\text{PREFIXES}(L) := \{x \mid xy \in L \text{ for some } y \in \Sigma^*\}$  be the language containing all prefixes of all strings in  $L$ . For example, if  $L = \{\textcolor{red}{000}, \textcolor{red}{100}, \textcolor{red}{110}, \textcolor{red}{111}\}$ , then  $\text{PREFIXES}(L) = \{\epsilon, \textcolor{red}{0}, \textcolor{red}{1}, \textcolor{red}{00}, \textcolor{red}{10}, \textcolor{red}{11}, \textcolor{red}{000}, \textcolor{red}{100}, \textcolor{red}{110}, \textcolor{red}{111}\}$ .

**Prove** that for any regular language  $L$ , the language  $\text{PREFIXES}(L)$  is also regular.

5. For each of the following languages  $L$ , give a regular expression that represents  $L$  **and** describe a DFA that recognizes  $L$ .

- (a) The set of all strings in  $\{0, 1\}^*$  that contain either both or neither of the substrings  $01$  and  $10$ .
- (b) The set of all strings in  $\{0, 1\}^*$  that do not contain the substring  $1001$ .

You do **not** need to prove that your answers are correct.

**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

1. For each statement below, check “True” if the statement is *always* true and “False” otherwise. Each correct answer is worth +1 point; each incorrect answer is worth  $-1/2$  point; checking “I don’t know” is worth  $+1/4$  point; and flipping a coin is (on average) worth  $+1/4$  point. You do **not** need to prove your answer is correct.

**Read each statement very carefully.** Some of these are deliberately subtle.

- (a) If 100 is a prime number, then Jeff is the Queen of England.
  - (b) The language  $\{0^m 0^{n+m} 0^n \mid m, n \geq 0\}$  is regular.
  - (c) For all languages  $L$ , the language  $L^*$  is regular.
  - (d) For all languages  $L \subset \Sigma^*$ , if  $L$  can be recognized by a DFA, then  $\Sigma^* \setminus L$  cannot be represented by a regular expression.
  - (e) For all languages  $L$  and  $L'$ , if  $L \subseteq L'$  and  $L'$  is regular, then  $L$  is regular.
  - (f) For all languages  $L$ , if  $L$  has a finite fooling set, then  $L$  is not regular.
  - (g) Let  $M = (\Sigma, Q, s, A, \delta)$  and  $M' = (\Sigma, Q, s, Q \setminus A, \delta)$  be arbitrary NFAs with identical alphabets, states, starting states, and transition functions, but with complementary accepting states. Then  $L(M) \cup L(M') = \Sigma^*$ .
  - (h) Let  $M = (\Sigma, Q, s, A, \delta)$  and  $M' = (\Sigma, Q, s, Q \setminus A, \delta)$  be arbitrary NFAs with identical alphabets, states, starting states, and transition functions, but with complementary accepting states. Then  $L(M) \cap L(M') = \emptyset$ .
  - (i) For all context-free languages  $L$  and  $L'$ , the language  $L \bullet L'$  is also context-free.
  - (j) Every regular language is context-free.
- 
2. For each of the following languages over the alphabet  $\Sigma = \{0, 1\}$ , either *prove* that the language is regular or *prove* that the language is not regular. *Exactly one of these two languages is regular.*
- (a)  $\{w0^n w \mid w \in \Sigma^+ \text{ and } n > 0\}$
  - (b)  $\{0^n w 0^n \mid w \in \Sigma^+ \text{ and } n > 0\}$

For example, both of these languages contain the string **00110100000110100**.

3. Let  $L = \{1^m 0^n \mid n \leq m \leq 2n\}$  and let  $G$  be the following context free-grammar:

$$S \rightarrow 1S0 \mid 11S0 \mid \epsilon$$

- (a) **Prove** that  $L(G) \subseteq L$ .
- (b) **Prove** that  $L \subseteq L(G)$ .

4. For any language  $L$ , let  $\text{PREFIXES}(L) := \{x \mid xy \in L \text{ for some } y \in \Sigma^*\}$  be the language containing all prefixes of all strings in  $L$ . For example, if  $L = \{\textcolor{red}{000}, \textcolor{red}{100}, \textcolor{red}{110}, \textcolor{red}{111}\}$ , then  $\text{PREFIXES}(L) = \{\epsilon, \textcolor{red}{0}, \textcolor{red}{1}, \textcolor{red}{00}, \textcolor{red}{10}, \textcolor{red}{11}, \textcolor{red}{000}, \textcolor{red}{100}, \textcolor{red}{110}, \textcolor{red}{111}\}$ .

**Prove** that for any regular language  $L$ , the language  $\text{PREFIXES}(L)$  is also regular.

5. For each of the following languages  $L$ , give a regular expression that represents  $L$  **and** describe a DFA that recognizes  $L$ .

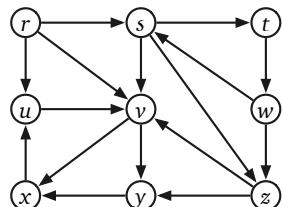
- (a) The set of all strings in  $\{0, 1\}^*$  that contain either both or neither of the substrings  $01$  and  $10$ .
- (b) The set of all strings in  $\{0, 1\}^*$  that do not contain the substring  $1010$ .

You do **not** need to prove that your answers are correct.

**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

1. *Clearly* indicate the following structures in the directed graph below, or write NONE if the indicated structure does not exist. (There are several copies of the graph in the answer booklet.)

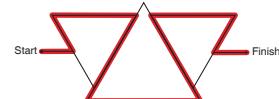
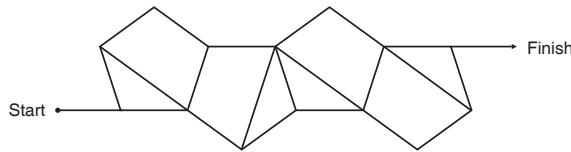
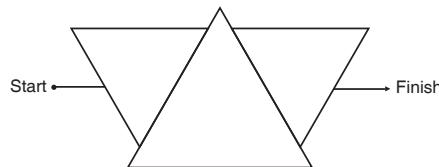


- (a) A depth-first spanning tree rooted at  $r$ .
- (b) A breadth-first spanning tree rooted at  $r$ .
- (c) A topological order.
- (d) The strongly connected components.

2. The following puzzles appear in my younger daughter's math workbook. (I've put the solutions on the right so you don't waste time solving them during the exam.)

**PRACTICE**

Complete each angle maze below by tracing a path from start to finish that has only acute angles.



Describe and analyze an algorithm to solve arbitrary acute-angle mazes.

You are given a connected undirected graph  $G$ , whose vertices are points in the plane and whose edges are line segments. Edges do not intersect, except at their endpoints. For example, a drawing of the letter X would have five vertices and four edges; the first maze above has 13 vertices and 15 edges. You are also given two vertices Start and Finish.

Your algorithm should return TRUE if  $G$  contains a walk from Start to Finish that has only acute angles, and FALSE otherwise. Formally, a walk through  $G$  is valid if, for any two consecutive edges  $u \rightarrow v \rightarrow w$  in the walk, either  $\angle uvw = 180^\circ$  or  $0 < \angle uvw < 90^\circ$ . Assume you have a subroutine that can compute the angle between any two segments in  $O(1)$  time. Do **not** assume that angles are multiples of  $1^\circ$ .

---

Jason Batterson and Shannon Rogers, *Beast Academy Math: Practice 3A*, 2012. See <https://www.beastacademy.com/resources/printables.php> for more examples.

3. Suppose you are given a sorted array  $A[1..n]$  of distinct numbers that has been *rotated*  $k$  steps, for some **unknown** integer  $k$  between 1 and  $n-1$ . That is, the prefix  $A[1..k]$  is sorted in increasing order, the suffix  $A[k+1..n]$  is sorted in increasing order, and  $A[n] < A[1]$ . For example, you might be given the following 16-element array (where  $k = 10$ ):

9	13	16	18	19	23	28	31	37	42	-4	0	2	5	7	8
---	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---

Describe and analyze an efficient algorithm to determine if the given array contains a given number  $x$ . The input to your algorithm is the array  $A[1..n]$  and the number  $x$ ; your algorithm is **not** given the integer  $k$ .

4. You have a collection of  $n$  lockboxes and  $m$  gold keys. Each key unlocks *at most* one box; however, each box might be unlocked by one key, by multiple keys, or by no keys at all. There are only two ways to open each box once it is locked: Unlock it properly (which requires having a matching key in your hand), or smash it to bits with a hammer.

Your baby brother, who loves playing with shiny objects, has somehow managed to lock all your keys inside the boxes! Luckily, your home security system recorded everything, so you know exactly which keys (if any) are inside each box. You need to get all the keys back out of the boxes, because they are made of gold. Clearly you have to smash at least one box.

- (a) Your baby brother has found the hammer and is eagerly eyeing one of the boxes. Describe and analyze an algorithm to determine if it is possible to retrieve all the keys without smashing any box except the one your brother has chosen.
- (b) Describe and analyze an algorithm to compute the minimum number of boxes that must be smashed to retrieve all the keys.

5. It's almost time to show off your flippin' sweet dancing skills! Tomorrow is the big dance contest you've been training for your entire life, except for that summer you spent with your uncle in Alaska hunting wolverines. You've obtained an advance copy of the the list of  $n$  songs that the judges will play during the contest, in chronological order.

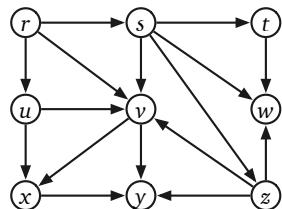
You know all the songs, all the judges, and your own dancing ability extremely well. For each integer  $k$ , you know that if you dance to the  $k$ th song on the schedule, you will be awarded exactly  $Score[k]$  points, but then you will be physically unable to dance for the next  $Wait[k]$  songs (that is, you cannot dance to songs  $k+1$  through  $k+Wait[k]$ ). The dancer with the highest total score at the end of the night wins the contest, so you want your total score to be as high as possible.

Describe and analyze an efficient algorithm to compute the maximum total score you can achieve. The input to your sweet algorithm is the pair of arrays  $Score[1..n]$  and  $Wait[1..n]$ .

**Write your answers in the separate answer booklet.**

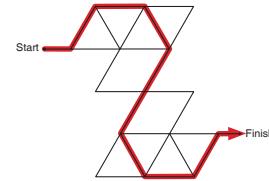
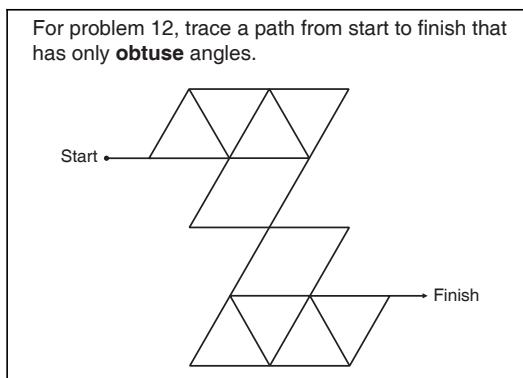
Please return this question sheet and your cheat sheet with your answers.

1. *Clearly* indicate the following structures in the directed graph below, or write NONE if the indicated structure does not exist. (There are several copies of the graph in the answer booklet.)



- (a) A depth-first spanning tree rooted at  $r$ .
- (b) A breadth-first spanning tree rooted at  $r$ .
- (c) A topological order.
- (d) The strongly connected components.

2. The following puzzle appears in my younger daughter's math workbook.  $\square$  (I've put the solution on the right so you don't waste time solving it during the exam.)



Describe and analyze an algorithm to solve arbitrary obtuse-angle mazes.

You are given a connected undirected graph  $G$ , whose vertices are points in the plane and whose edges are line segments. Edges do not intersect, except at their endpoints. For example, a drawing of the letter X would have five vertices and four edges; the maze above has 17 vertices and 26 edges. You are also given two vertices Start and Finish.

Your algorithm should return TRUE if  $G$  contains a walk from Start to Finish that has only obtuse angles, and FALSE otherwise. Formally, a walk through  $G$  is valid if  $90^\circ < \angle uvw \leq 180^\circ$  for every pair of consecutive edges  $u \rightarrow v \rightarrow w$  in the walk. Assume you have a subroutine that can compute the angle between any two segments in  $O(1)$  time. Do **not** assume that angles are multiples of  $1^\circ$ .

---

$\square$ Jason Batterson and Shannon Rogers, *Beast Academy Math: Practice 3A*, 2012. See <https://www.beastacademy.com/resources/printables.php> for more examples.

3. Suppose you are given two unsorted arrays  $A[1..n]$  and  $B[1..n]$  containing  $2n$  distinct integers, such that  $A[1] < B[1]$  and  $A[n] > B[n]$ . Describe and analyze an efficient algorithm to compute an index  $i$  such that  $A[i] < B[i]$  and  $A[i+1] > B[i+1]$ . [Hint: Why does such an index  $i$  always exist?]
4. You have a collection of  $n$  lockboxes and  $m$  gold keys. Each key unlocks *at most* one box; however, each box might be unlocked by one key, by multiple keys, or by no keys at all. There are only two ways to open each box once it is locked: Unlock it properly (which requires having a matching key in your hand), or smash it to bits with a hammer.

Your baby brother, who loves playing with shiny objects, has somehow managed to lock all your keys inside the boxes! Luckily, your home security system recorded everything, so you know which keys (if any) are inside each box. You need to get all the keys back out of the boxes, because they are made of gold. Clearly you have to smash at least one box.

- (a) Your baby brother has found the hammer and is eagerly eyeing one of the boxes. Describe and analyze an algorithm to determine if it is possible to retrieve all the keys without smashing any box except the one your brother has chosen.
- (b) Describe and analyze an algorithm to compute the minimum number of boxes that must be smashed to retrieve all the keys.
5. A *shuffle* of two strings  $X$  and  $Y$  is formed by interspersing the characters into a new string, keeping the characters of  $X$  and  $Y$  in the same order. For example, the string **BANANAANANAS** is a shuffle of the strings **BANANA** and **ANANAS** in several different ways.

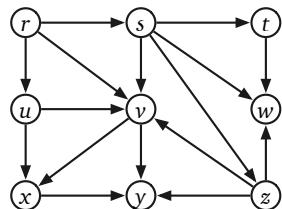
**BANANAANANAS**      **BANANAANANAS**      **BANANAANANAS**

Given three strings  $A[1..m]$ ,  $B[1..n]$ , and  $C[1..m+n]$ , describe and analyze an algorithm to determine whether  $C$  is a shuffle of  $A$  and  $B$ .

**Write your answers in the separate answer booklet.**

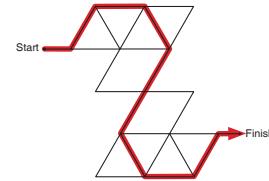
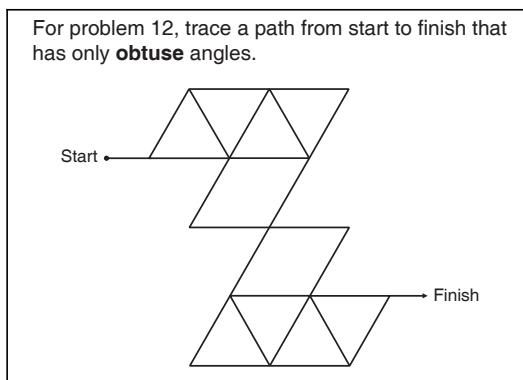
Please return this question sheet and your cheat sheet with your answers.

1. *Clearly* indicate the following structures in the directed graph below, or write NONE if the indicated structure does not exist. (There are several copies of the graph in the answer booklet.)



- (a) A depth-first spanning tree rooted at  $r$ .
- (b) A breadth-first spanning tree rooted at  $r$ .
- (c) A topological order.
- (d) The strongly connected components.

2. The following puzzle appears in my younger daughter's math workbook.  $\square$  (I've put the solution on the right so you don't waste time solving it during the exam.)



Describe and analyze an algorithm to solve arbitrary obtuse-angle mazes.

You are given a connected undirected graph  $G$ , whose vertices are points in the plane and whose edges are line segments. Edges do not intersect, except at their endpoints. For example, a drawing of the letter X would have five vertices and four edges; the maze above has 17 vertices and 26 edges. You are also given two vertices Start and Finish.

Your algorithm should return TRUE if  $G$  contains a walk from Start to Finish that has only obtuse angles, and FALSE otherwise. Formally, a walk through  $G$  is valid if  $90^\circ < \angle uvw \leq 180^\circ$  for every pair of consecutive edges  $u \rightarrow v \rightarrow w$  in the walk. Assume you have a subroutine that can compute the angle between any two segments in  $O(1)$  time. Do **not** assume that angles are multiples of  $1^\circ$ .

---

$\square$ Jason Batterson and Shannon Rogers, *Beast Academy Math: Practice 3A*, 2012. See <https://www.beastacademy.com/resources/printables.php> for more examples.

3. Suppose you are given two unsorted arrays  $A[1..n]$  and  $B[1..n]$  containing  $2n$  distinct integers, such that  $A[1] < B[1]$  and  $A[n] > B[n]$ . Describe and analyze an efficient algorithm to compute an index  $i$  such that  $A[i] < B[i]$  and  $A[i+1] > B[i+1]$ . [Hint: Why does such an index  $i$  always exist?]
4. You have a collection of  $n$  lockboxes and  $m$  gold keys. Each key unlocks *at most* one box; however, each box might be unlocked by one key, by multiple keys, or by no keys at all. There are only two ways to open each box once it is locked: Unlock it properly (which requires having a matching key in your hand), or smash it to bits with a hammer.

Your baby brother, who loves playing with shiny objects, has somehow managed to lock all your keys inside the boxes! Luckily, your home security system recorded everything, so you know which keys (if any) are inside each box. You need to get all the keys back out of the boxes, because they are made of gold. Clearly you have to smash at least one box.

- (a) Your baby brother has found the hammer and is eagerly eyeing one of the boxes. Describe and analyze an algorithm to determine if it is possible to retrieve all the keys without smashing any box except the one your brother has chosen.
- (b) Describe and analyze an algorithm to compute the minimum number of boxes that must be smashed to retrieve all the keys.
5. A palindrome is any string that is exactly the same as its reversal, like **I**, or **DEED**, or **RACECAR**, or **AMANAPLANACATACANALPANAMA**.

Describe and analyze an algorithm to find the length of the *longest subsequence* of a given string that is also a palindrome. For example, the longest palindrome subsequence of **MAHDYDYNAMICICPROGRAMZLETMESHOWYOUTHEM** is **MHYMRORMYHM**, so given that string as input, your algorithm should output the number 11.

**CS/ECE 374: Algorithms and Models of Computation, Fall 2016**  
**Final Exam (Version X) — December 15, 2016**

Name:											
NetID:											
Section:	A 9–10 Spencer	B 10–11 Yipu	C 11–12 Charlie	D 12–1 Chao	E 1–2 Alex	F 1–2 Tana	G 2–3 Mark Konstantinos	H 2–3 Konstantinos	I don't know	J 3–4 Mark Konstantinos	K 3–4 Mark Konstantinos

#	1	2	3	4	5	6	Total
Score							
Max	20	10	10	10	10	10	70
Grader							

- 
- ***Don't panic!***
  - Please print your name and your NetID and circle your discussion section in the boxes above.
  - This is a closed-book, closed-notes, closed-electronics exam. If you brought anything except your writing implements and your two double-sided 8½" × 11" cheat sheets, please put it away for the duration of the exam. In particular, you may not use *any* electronic devices.
  - **Please read the entire exam before writing anything.** Please ask for clarification if any question is unclear.
  - **The exam lasts 180 minutes.**
  - If you run out of space for an answer, continue on the back of the page, or on the blank pages at the end of this booklet, **but please tell us where to look.** Alternatively, feel free to tear out the blank pages and use them as scratch paper.
  - As usual, answering any (sub)problem with “I don’t know” (and nothing else) is worth 25% partial credit. **Yes, even for problem 1.** Correct, complete, but suboptimal solutions are *always* worth more than 25%. A blank answer is not the same as “I don’t know”.
  - **Beware the Three Deadly Sins.** Give complete solutions, not examples. Don’t use weak induction. Declare all your variables.
  - If you use a greedy algorithm, you **must** prove that it is correct to receive any credit. Otherwise, proofs are required only when we explicitly ask for them.
  - **Please return your cheat sheets and all scratch paper with your answer booklet.**
  - ***Good luck!*** And have a great winter break!
-

1. For each of the following questions, indicate *every* correct answer by marking the “Yes” box, and indicate *every* incorrect answer by marking the “No” box. **Assume P  $\neq$  NP.** If there is any other ambiguity or uncertainty, mark the “No” box. For example:

<input checked="" type="checkbox"/> Yes	No	$2 + 2 = 4$
Yes	<input checked="" type="checkbox"/> No	$x + y = 5$
Yes	<input checked="" type="checkbox"/> No	3SAT can be solved in polynomial time.
<input checked="" type="checkbox"/> Yes	No	Jeff is not the Queen of England.

There are 40 yes/no choices altogether.

- Each correct choice is worth  $+\frac{1}{2}$  point.
  - Each incorrect choice is worth  $-\frac{1}{4}$  point.
  - To indicate “I don’t know”, write IDK next to the boxes; each IDK is worth  $+\frac{1}{8}$  point.
- 

- (a) Which of the following statements is true for *every* language  $L \subseteq \{0, 1\}^*$ ?

<input checked="" type="checkbox"/> Yes	No	$L$ contains the empty string $\varepsilon$ .
Yes	No	$L^*$ is infinite.
Yes	No	$L^*$ is regular.
Yes	No	$L$ is infinite or $L$ is decidable (or both).
Yes	No	If $L$ is the union of two regular languages, then $L$ is regular.
Yes	No	If $L$ is the union of two decidable languages, then $L$ is decidable.
Yes	No	If $L$ is the union of two undecidable languages, then $L$ is undecidable.
<input checked="" type="checkbox"/> Yes	No	$L$ is accepted by some NFA with 374 states if and only if $L$ is accepted by some DFA with 374 states.
<input checked="" type="checkbox"/> Yes	No	If $L \notin P$ , then $L$ is not regular.

---

(b) Which of the following languages over the alphabet  $\{\text{0}, \text{1}\}$  are *regular*?

Yes	No
-----	----

$\{\text{0}^m \text{1}^n \mid m \geq 0 \text{ and } n \geq 0\}$

Yes	No
-----	----

All strings with the same number of 0s and 1s

Yes	No
-----	----

Binary representations of all prime numbers less than  $10^{100}$

Yes	No
-----	----

$\{ww \mid w \text{ is a palindrome}\}$

Yes	No
-----	----

$\{wxw \mid w \text{ is a palindrome and } x \in \{\text{0}, \text{1}\}^*\}$

Yes	No
-----	----

$\{\langle M \rangle \mid M \text{ accepts a finite number of non-palindromes}\}$

---

(c) Which of the following languages over the alphabet  $\{\text{0}, \text{1}\}$  are *decidable*?

Yes	No
-----	----

$\emptyset$

Yes	No
-----	----

$\{\text{0}^n \text{1}^{2n} \text{0}^n \text{1}^{2n} \mid n \geq 0\}$

Yes	No
-----	----

$\{ww \mid w \text{ is a palindrome}\}$

Yes	No
-----	----

$\{\langle M \rangle \mid M \text{ accepts a finite number of non-palindromes}\}$

Yes	No
-----	----

$\{\langle M, w \rangle \mid M \text{ accepts } ww\}$

Yes	No
-----	----

$\{\langle M, w \rangle \mid M \text{ accepts } ww \text{ after at most } |w|^2 \text{ transitions}\}$

---

(d) Which of the following languages can be proved undecidable *using Rice's Theorem*?

Yes	No
-----	----

$\{\langle M \rangle \mid M \text{ accepts an infinite number of strings}\}$

Yes	No
-----	----

$\{\langle M \rangle \mid M \text{ accepts either } \langle M \rangle \text{ or } \langle M \rangle^R\}$

Yes	No
-----	----

$\{\langle M \rangle \mid M \text{ does not accept exactly 374 palindromes}\}$

Yes	No
-----	----

$\{\langle M \rangle \mid M \text{ accepts some string } w \text{ after at most } |w|^2 \text{ transitions}\}$

---

- (e) Which of the following is a good English specification of a recursive function that can be used to compute the edit distance between two strings  $A[1..n]$  and  $B[1..n]$ ?

<input type="checkbox"/> Yes	<input type="checkbox"/> No	<i>Edit(i, j)</i> is the answer for $i$ and $j$ .
<input type="checkbox"/> Yes	<input type="checkbox"/> No	<i>Edit(i, j)</i> is the edit distance between $A[i]$ and $B[j]$ .
<input type="checkbox"/> Yes	<input type="checkbox"/> No	<i>Edit[1..n, 1..n]</i> stores the edit distances for all prefixes.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	<i>Edit(i, j)</i> is the edit distance between $A[i..n]$ and $B[j..n]$ .
<input type="checkbox"/> Yes	<input type="checkbox"/> No	<i>Edit[i, j]</i> is the value stored at row $i$ and column $j$ of the table.

- (f) Suppose we want to prove that the following language is undecidable.

$$\text{MUGGLE} := \{\langle M \rangle \mid M \text{ accepts } \text{SCIENCE} \text{ but rejects } \text{MAGIC}\}$$

Professor Potter, your instructor in Defense Against Models of Computation and Other Dark Arts, suggests a reduction from the standard halting language

$$\text{HALT} := \{\langle M, w \rangle \mid M \text{ halts on inputs } w\}.$$

Specifically, suppose there is a Turing machine DETECTOMUGGETUM that decides MUGGLE. Professor Potter claims that the following algorithm decides HALT.

```

DECIDEHALT( $\langle M, w \rangle$ ):
  Encode the following Turing machine:
    RUBBERDUCK( $x$ ):
      run  $M$  on input  $w$ 
      if  $x = \text{MAGIC}$ 
        return FALSE
      else
        return TRUE
  return DETECTOMUGGETUM( $\langle \text{RUBBERDUCK} \rangle$ )

```

Which of the following statements is true for all inputs  $\langle M, w \rangle$ ?

<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $M$ rejects $w$ , then RUBBERDUCK rejects <b>MAGIC</b> .
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $M$ accepts $w$ , then DETECTOMUGGETUM accepts $\langle \text{RUBBERDUCK} \rangle$ .
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $M$ rejects $w$ , then DECIDEHALT rejects $\langle M, w \rangle$ .
<input type="checkbox"/> Yes	<input type="checkbox"/> No	DECIDEHALT decides the language HALT. (That is, Professor Potter's reduction is actually correct.)
<input type="checkbox"/> Yes	<input type="checkbox"/> No	DECIDEHALT actually runs (or simulates) RUBBERDUCK.

(g) Consider the following pair of languages:

- HAMPATH :=  $\{G \mid G \text{ is a directed graph with a Hamiltonian path}\}$
- ACYCLIC :=  $\{G \mid G \text{ is a directed acyclic graph}\}$

(For concreteness, assume that in both of these languages, graphs are represented by their adjacency matrices.) Which of the following *must* be true, assuming P  $\neq$  NP?

Yes	No

ACYCLIC  $\in$  NP

ACYCLIC  $\cap$  HAMPATH  $\in$  P

HAMPATH is decidable.

There is no polynomial-time reduction from HAMPATH to ACYCLIC.

There is no polynomial-time reduction from ACYCLIC to HAMPATH.

---

2. A *quasi-satisfying assignment* for a 3CNF boolean formula  $\Phi$  is an assignment of truth values to the variables such that *at most one* clause in  $\Phi$  does not contain a true literal. *Prove* that it is NP-hard to determine whether a given 3CNF boolean formula has a quasi-satisfying assignment.

3. Recall that a *run* in a string is a maximal non-empty substring in which all symbols are equal. For example, the string **0001111000** consists of three runs: a run of three **0**s, a run of four **1**s, and another run of three **0**s.

- (a) Let  $L$  be the set of all strings in  $\{0, 1\}^*$  in which every run of **0**s has odd length and every run of **1**s has even length. For example,  $L$  contains  $\epsilon$  and **00000** and **0001111000**, but  $L$  does not contain **1** or **0000** or **110111000**.

Describe both a regular expression for  $L$  and a DFA that accepts  $L$ .

- (b) Let  $L'$  be the set of all non-empty strings in  $\{0, 1\}^*$  in which the *number* of runs is equal to the *length* of the first run. For example,  $L'$  contains **0** and **1100** and **0000101**, but  $L'$  does not contain **0000** or **110111000** or  $\epsilon$ .

*Prove* that  $L'$  is not a regular language.

4. Your cousin Elmo is visiting you for Christmas, and he's brought a different card game. Like your previous game with Elmo, this game is played with a row of  $n$  cards, each labeled with an integer (which could be positive, negative, or zero). Both players can see all  $n$  card values. Otherwise, the game is almost completely different.

On each turn, the current player must take the leftmost card. The player can either keep the card or give it to their opponent. If they keep the card, their turn ends and their opponent takes the next card; however, if they give the card to their opponent, the current player's turn continues with the next card. In short, the player that does *not* get the  $i$ th card decides who gets the  $(i + 1)$ th card. The game ends when all cards have been played. Each player adds up their card values, and whoever has the higher total wins.

For example, suppose the initial cards are  $[3, -1, 4, 1, 5, 9]$ , and Elmo plays first. Then the game might proceed as follows:

- Elmo keeps the 3, ending his turn.
- You give Elmo the  $-1$ .
- You keep the 4, ending your turn.
- Elmo gives you the 1.
- Elmo gives you the 5.
- Elmo keeps the 9, ending his turn. All cards are gone, so the game is over.
- Your score is  $1 + 4 + 5 = 10$  and Elmo's score is  $3 - 1 + 9 = 11$ , so Elmo wins.

Describe an algorithm to compute the highest possible score you can earn from a given row of cards, assuming Elmo plays first and plays perfectly. Your input is the array  $C[1..n]$  of card values. For example, if the input is  $[3, -1, 4, 1, 5, 9]$ , your algorithm should return the integer 10.

5. **Prove** that each of the following languages is undecidable:

- (a)  $\{\langle M \rangle \mid M \text{ accepts RICESTHEOREM}\}$
- (b)  $\{\langle M \rangle \mid M \text{ rejects RICESTHEOREM}\}$  [Hint: Use part (a), **not** Rice's theorem]

6. There are  $n$  galaxies connected by  $m$  intergalactic teleport-ways. Each teleport-way joins two galaxies and can be traversed in both directions. Also, each teleport-way  $uv$  has an associated cost of  $c(uv)$  galactic credits, for some positive integer  $c(uv)$ . The same teleport-way can be used multiple times in either direction, but the same toll must be paid every time it is used.

Judy wants to travel from galaxy  $s$  to galaxy  $t$ , but teleportation is rather unpleasant, so she wants to minimize the number of times she has to teleport. However, she also wants the total cost to be a multiple of 10 galactic credits, because carrying small change is annoying.

Describe and analyze an algorithm to compute the minimum number of times Judy must teleport to travel from galaxy  $s$  to galaxy  $t$  so that the total cost of all teleports is an integer multiple of 10 galactic credits. Your input is a graph  $G = (V, E)$  whose vertices are galaxies and whose edges are teleport-ways; every edge  $uv$  in  $G$  stores the corresponding cost  $c(uv)$ .

*[Hint: This is **not** the same Intergalactic Judy problem that you saw in lab.]*

(scratch paper)

(scratch paper)

**You may assume the following problems are NP-hard:**

**CIRCUITSAT:** Given a boolean circuit, are there any input values that make the circuit output TRUE?

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

**MAXINDEPENDENTSET:** Given an undirected graph  $G$ , what is the size of the largest subset of vertices in  $G$  that have no edges among them?

**MAXCLIQUE:** Given an undirected graph  $G$ , what is the size of the largest complete subgraph of  $G$ ?

**MINVERTEXCOVER:** Given an undirected graph  $G$ , what is the size of the smallest subset of vertices that touch every edge in  $G$ ?

**3COLOR:** Given an undirected graph  $G$ , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

**HAMILTONIANPATH:** Given an undirected graph  $G$ , is there a path in  $G$  that visits every vertex exactly once?

**HAMILTONIANCYCLE:** Given an undirected graph  $G$ , is there a cycle in  $G$  that visits every vertex exactly once?

**DIRECTEDHAMILTONIANCYCLE:** Given an directed graph  $G$ , is there a directed cycle in  $G$  that visits every vertex exactly once?

**TRAVELINGSALESMAN:** Given a graph  $G$  (either directed or undirected) with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in  $G$ ?

**DRAUGHTS:** Given an  $n \times n$  international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

**SUPER MARIO:** Given an  $n \times n$  level for Super Mario Brothers, can Mario reach the castle?

**You may assume the following languages are undecidable:**

SELFREJECT :=  $\{\langle M \rangle \mid M \text{ rejects } \langle M \rangle\}$

SELFACCEPT :=  $\{\langle M \rangle \mid M \text{ accepts } \langle M \rangle\}$

SELFHALT :=  $\{\langle M \rangle \mid M \text{ halts on } \langle M \rangle\}$

SELFDIVERGE :=  $\{\langle M \rangle \mid M \text{ does not halt on } \langle M \rangle\}$

REJECT :=  $\{\langle M, w \rangle \mid M \text{ rejects } w\}$

ACCEPT :=  $\{\langle M, w \rangle \mid M \text{ accepts } w\}$

HALT :=  $\{\langle M, w \rangle \mid M \text{ halts on } w\}$

DIVERGE :=  $\{\langle M, w \rangle \mid M \text{ does not halt on } w\}$

NEVERREJECT :=  $\{\langle M \rangle \mid \text{REJECT}(M) = \emptyset\}$

NEVERACCEPT :=  $\{\langle M \rangle \mid \text{ACCEPT}(M) = \emptyset\}$

NEVERHALT :=  $\{\langle M \rangle \mid \text{HALT}(M) = \emptyset\}$

NEVERDIVERGE :=  $\{\langle M \rangle \mid \text{DIVERGE}(M) = \emptyset\}$

## CS 473 ✦ Spring 2017

## ❖ Homework 0 ❖

Due Wednesday, January 25, 2017 at 8pm

---

- **This homework tests your familiarity with prerequisite material:** designing, describing, and analyzing elementary algorithms; fundamental graph problems and algorithms; and especially facility with recursion and induction. Notes on most of this prerequisite material are available on the course web page.
  - **Each student must submit individual solutions for this homework.** For all future homeworks, groups of up to three students will be allowed to submit joint solutions.
  - **Submit your solutions electronically on Gradescope as PDF files.**
    - Submit a separate file for each numbered problem.
    - You can find a L<sup>A</sup>T<sub>E</sub>X solution template on the course web site (soon); please use it if you plan to typeset your homework.
    - If you must submit scanned handwritten solutions, please use dark ink (not pencil) on blank white printer paper (not notebook or graph paper), use a high-quality scanner (not a phone camera), and print the resulting PDF file on a black-and-white printer to verify readability before you submit.
- 

## ❖ Some important course policies ❖

- **You may use any source at your disposal**—paper, electronic, or human—but you **must** cite **every** source that you use, and you **must** write everything yourself in your own words. See the academic integrity policies on the course web site for more details.
- The answer “*I don’t know*” (and *nothing else*) is worth 25% partial credit on any problem or subproblem, on any homework or exam, except for extra-credit problems. We will accept synonyms like “No idea” or “WTF” or “- \ ( •\_• ) / -”, but you must write *something*.
- **Avoid the Deadly Sins!** There are a few dangerous writing (and thinking) habits that will trigger an **automatic zero** on any homework or exam problem. Yes, really.
  - Always give complete solutions, not just examples.
  - Every algorithm requires an English specification.
  - Greedy algorithms require formal correctness proofs.
  - Never use weak induction.

---

See the course web site for more information.

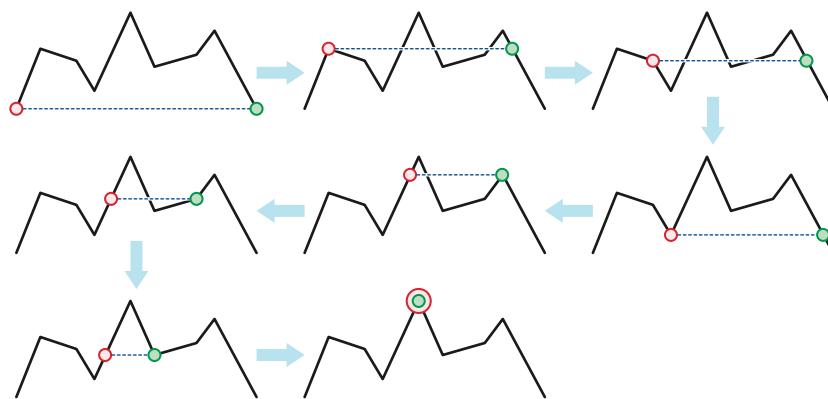
If you have any questions about these policies,  
please don’t hesitate to ask in class, in office hours, or on Piazza.

---

1. Every cheesy romance movie has a scene where the romantic couple, after a long and frustrating separation, suddenly see each other across a long distance, and then slowly approach one another with unwavering eye contact as the music rolls in and the rain lifts and the sun shines through the clouds and the music swells and everyone starts dancing with rainbows and kittens and chocolate unicorns and... . $\square$

Suppose a romantic couple—in grand computer science tradition, named Alice and Bob—enters their favorite park at the east and west entrances and immediately establish eye-contact. They can't just run directly to each other; instead, they must stay on the path that zig-zags through the part between the east and west entrances. To maintain the proper dramatic tension, Alice and Bob must traverse the path so that they always lie on a direct east-west line.

We can describe the zigzag path as two arrays  $X[0..n]$  and  $Y[0..n]$ , containing the  $x$ - and  $y$ -coordinates of the corners of the path, in order from the southwest endpoint to the southeast endpoint. The  $X$  array is sorted in increasing order, and  $Y[0] = Y[n]$ . The path is a sequence of straight line segments connecting these corners.



Alice and Bob meet. Alice walks backward in step 2, and Bob walks backward in steps 5 and 6.

- (a) Suppose  $Y[0] = Y[n] = 0$  and  $Y[i] > 0$  for every other index  $i$ ; that is, the endpoints of the path are strictly below every other point on the path. Prove that under these conditions, Alice and Bob can meet.

*[Hint: Describe a graph that models all possible locations and transitions of the couple along the path. What are the vertices of this graph? What are the edges? What can you say about the degrees of the vertices?]*

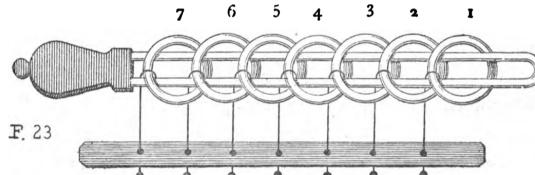
- (b) If the endpoints of the path are *not* below every other vertex, Alice and Bob might still be able to meet, or they might not. Describe an algorithm to decide whether Alice and Bob can meet, without either breaking east-west eye contact or stepping off the path, given the arrays  $X[0..n]$  and  $Y[0..n]$  as input.

*[Hint: Build the graph from part (a). (How?) What problem do you need to solve on this graph? Call a textbook algorithm to solve that problem. (Do not regurgitate the textbook algorithm.) What is your overall running time as a function of  $n$ ?]*

---

$\square$ Fun fact: Damien Chazelle, the director of *Whiplash* and *La La Land*, is the son of Princeton computer science professor Bernard Chazelle.

2. The Tower of Hanoi is a relatively recent descendant of a much older mechanical puzzle known as the Chinese rings, Baguenaudier (a French word meaning “to wander about aimlessly”), Meleda, Patience, Tiring Irons, Prisoner’s Lock, Spin-Out, and many other names. This puzzle was already well known in both China and Europe by the 16th century. The Italian mathematician Luca Pacioli described the 7-ring puzzle and its solution in his unpublished treatise *De Viribus Quantitatis*, written between 1498 and 1506;□ only a few years later, the Ming-dynasty poet Yang Shen described the 9-ring puzzle as “a toy for women and children”.



A drawing of a 7-ring Baguenaudier, from *Récréations Mathématiques* by Édouard Lucas (1891)

The Baguenaudier puzzle has many physical forms, but it typically consists of a long metal loop and several rings, which are connected to a solid base by movable rods. The loop is initially threaded through the rings as shown in the figure above; the goal of the puzzle is to remove the loop.

More abstractly, we can model the puzzle as a sequence of bits, one for each ring, where the  $i$ th bit is 1 if the loop passes through the  $i$ th ring and 0 otherwise. (Here we index the rings from right to left, as shown in the figure.) The puzzle allows two legal moves:

- You can always flip the 1st (= rightmost) bit.
- If the bit string ends with exactly  $i$  0s, you can flip the  $(i + 2)$ th bit.

The goal of the puzzle is to transform a string of  $n$  1s into a string of  $n$  0s. For example, the following sequence of 21 moves solve the 5-ring puzzle:

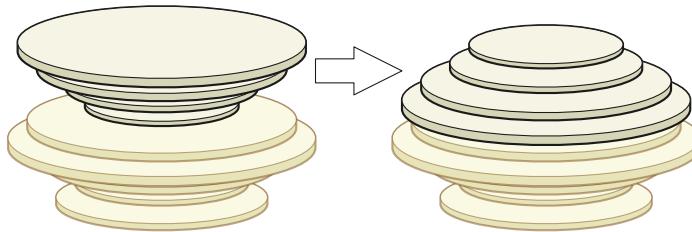
$$\begin{aligned}
 & \mathbf{11111} \xrightarrow{1} \mathbf{11110} \xrightarrow{3} \mathbf{11010} \xrightarrow{1} \mathbf{11011} \xrightarrow{2} \mathbf{11001} \xrightarrow{1} \mathbf{11000} \xrightarrow{5} \mathbf{01000} \\
 & \xrightarrow{1} \mathbf{01001} \xrightarrow{2} \mathbf{01011} \xrightarrow{1} \mathbf{01010} \xrightarrow{3} \mathbf{01110} \xrightarrow{1} \mathbf{01111} \xrightarrow{2} \mathbf{01101} \xrightarrow{1} \mathbf{01100} \xrightarrow{4} \mathbf{00100} \\
 & \xrightarrow{1} \mathbf{00101} \xrightarrow{2} \mathbf{00111} \xrightarrow{1} \mathbf{00110} \xrightarrow{3} \mathbf{00010} \xrightarrow{1} \mathbf{00011} \xrightarrow{2} \mathbf{00001} \xrightarrow{1} \mathbf{00000}
 \end{aligned}$$

- Describe an algorithm to solve the Baguenaudier puzzle. Your input is the number of rings  $n$ ; your algorithm should print a sequence of moves that solves the  $n$ -ring puzzle. For example, given the integer 5 as input, your algorithm should print the sequence 1, 3, 1, 2, 1, 5, 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1.
- Exactly how many moves does your algorithm perform, as a function of  $n$ ? Prove your answer is correct.
- [Extra credit]** Call a sequence of moves *reduced* if no move is the inverse of the previous move. Prove that for any non-negative integer  $n$ , there is *exactly one* reduced sequence of moves that solves the  $n$ -ring Baguenaudier puzzle. [Hint: See problem 1!]

---

□*De Viribus Quantitatis* [*On the Powers of Numbers*] is an important early work on recreational mathematics and perhaps the oldest surviving treatise on magic. Pacioli is better known for *Summa de Arithmetica*, a near-complete encyclopedia of late 15th-century mathematics, which included the first description of double-entry bookkeeping.

3. Suppose you are given a stack of  $n$  pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a *flip*—insert a spatula under the top  $k$  pancakes, for some integer  $k$  between 1 and  $n$ , and flip them all over.



Flipping the top four pancakes.

- (a) Describe an algorithm to sort an arbitrary stack of  $n$  pancakes using as few flips as possible. *Exactly* how many flips does your algorithm perform in the worst case?
- (b) Now suppose one side of each pancake is burned. Describe an algorithm to sort an arbitrary stack of  $n$  pancakes, so that the burned side of every pancake is facing down, using as few flips as possible. *Exactly* how many flips does your algorithm perform in the worst case?

[Hint: This problem has **nothing** to do with the Tower of Hanoi!]

# CS 473 ✦ Spring 2017

## ❖ Homework 1 ❖

Due Wednesday, February 1, 2017 at 8pm

---

Starting with this homework, groups of up to three people can submit joint solutions. Each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the Gradescope names and email addresses of each group member. In addition, whoever submits the homework must tell Gradescope who their other group members are.

---

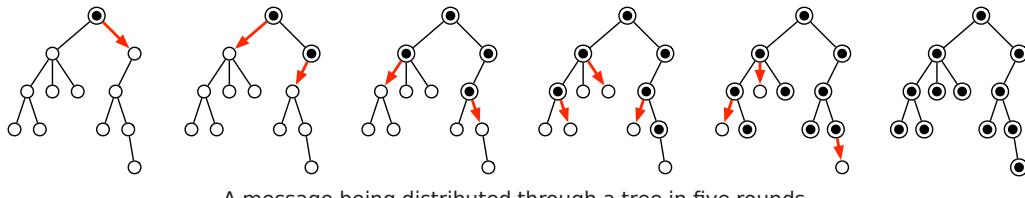
1. A palindrome is any string that is exactly the same as its reversal, like **I**, or **DEED**, or **RACECAR**, or **AMANAPLANACATACANALPANAMA**.

Any string can be decomposed into a sequence of palindromes. For example, the string **BUBBASEESABANANA** (“Bubba sees a banana.”) can be broken into palindromes in the following ways (and many others):

**BUB • BASEESAB • ANANA**  
**B • U • BB • A • SEES • ABA • NAN • A**  
**B • U • BB • A • SEES • A • B • ANANA**  
**B • U • B • B • A • S • E • E • S • A • B • A • N • ANA**

Describe and analyze an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string **BUBBASEESABANANA**, your algorithm would return the integer 3.

2. Suppose we need to distribute a message to all the nodes in a rooted tree. Initially, only the root node knows the message. In a single round, any node that knows the message can forward it to at most one of its children.



A message being distributed through a tree in five rounds.

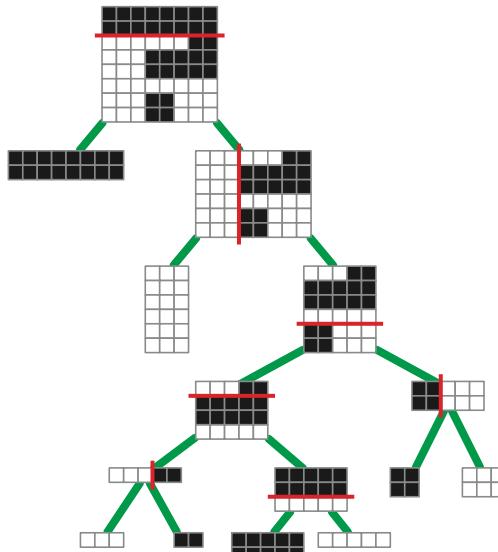
- Describe an algorithm to compute the minimum number of rounds required for the message to be delivered to all nodes in a *binary* tree.
- Describe an algorithm to compute the minimum number of rounds required for the message to be delivered to all nodes in an *arbitrary rooted* tree.

[Hint: Don't forget to justify your algorithm's correctness; you may find the lecture notes on greedy algorithms helpful. Any algorithm for this part also solves part (a).]

3. Suppose you are given an  $m \times n$  bitmap, represented by an array  $M[1..m, 1..n]$  of 0s and 1s. A *solid block* in  $M$  is a contiguous subarray  $M[i..i', j..j']$  in which all bits are equal.

A *guillotine subdivision* is a compact data structure to represent bitmaps as a recursive decomposition into solid blocks. If the entire bitmap  $M$  is a solid block, there is nothing to do. Otherwise, we cut  $M$  into two smaller bitmaps along a horizontal or vertical line, and then decompose the two smaller bitmaps recursively.  $\square$

Any guillotine subdivision can be represented as a binary tree, where each internal node stores the position and orientation of a cut, and each leaf stores a single bit 0 or 1 indicating the contents of the corresponding block. The *size* of a guillotine subdivision is the number of leaves in the corresponding binary tree (that is, the final number of solid blocks), and the *depth* of a guillotine subdivision is the depth of the corresponding binary tree.



A guillotine subdivision with size 8 and depth 5.

- (a) Describe and analyze an algorithm to compute a guillotine subdivision of  $M$  of minimum *size*.
- (b) Describe and analyze an algorithm to compute a guillotine subdivision of  $M$  of minimum *depth*.

---

$\square$ Guillotine subdivisions are similar to kd-trees, except that the cuts in a guillotine subdivision are *not* required to alternate between horizontal and vertical.

**Standard dynamic programming rubric.** For problems worth 10 points:

- 3 points for a clear **English** specification of the recursive function you are trying to evaluate. (Otherwise, we don't even know what you're *trying* to do.)
  - + 2 points for describing the function itself. (For example: " $OPT(i, j)$  is the edit distance between  $A[1..i]$  and  $B[1..j]$ .)"
  - + 1 point for stating how to call your recursive function to get the final answer. (For example: "We need to compute  $OPT(m, n)$ .)"
  - + An English description of the **algorithm** is not sufficient. We want an English description of the underlying recursive **problem**. In particular, the description should specify precisely the role of each input parameter.
  - + No credit for the rest of the problem if the English description is missing. (This is a Deadly Sin.)
- 4 points for a correct recurrence, described either using mathematical notation or as pseudocode for a recursive algorithm.
  - + 1 point for base case(s).  $-1/2$  for one *minor* bug, like a typo or an off-by-one error.
  - + 3 points for recursive case(s).  $-1$  for each *minor* bug, like a typo or an off-by-one error. **No credit for the rest of the problem if the recursive case(s) are incorrect.**
- 3 points for details of the iterative dynamic programming algorithm
  - + 1 point for describing the memoization data structure
  - + 1 point for describing a correct evaluation order; a clear picture is usually sufficient. If you use nested loops, be sure to specify the nesting order.
  - + 1 point for time analysis
- It is *not* necessary to state a space bound.
- For problems that ask for an algorithm that computes an optimal *structure*—such as a subset, partition, subsequence, or tree—an algorithm that computes only the *value* or *cost* of the optimal structure is sufficient for full credit, unless the problem says otherwise.
- Official solutions usually include pseudocode for the final iterative dynamic programming algorithm, **but iterative psuedocode is not required for full credit**. If your solution includes iterative pseudocode, you do not need to separately describe the recurrence, memoization structure, or evaluation order. (But you still need to specify the underlying recursive function in English.)
- Official solutions will provide target time bounds. Algorithms that are faster than this target are worth more points; slower algorithms are worth fewer points, typically by 2 or 3 points (out of 10) for each factor of  $n$ . Partial credit is scaled to the new maximum score, and all points above 10 are recorded as extra credit.

We rarely include these target time bounds in the actual questions, because when we have included them, significantly more students turned in algorithms that meet the target time bound but didn't work (earning 0/10) instead of correct algorithms that are slower than the target time bound (earning 8/10).

# CS 473 ✦ Spring 2017

## ❖ Homework 2 ❖

Due Wednesday, February 8, 2017 at 8pm

---

There are only two problems, but the first one counts double.

---

1. Suppose you are given a two-dimensional array  $M[1..n, 1..n]$  of numbers, which could be positive, negative, or zero, and which are *not* necessarily integers. The **maximum subarray problem** asks to find the largest sub of elements in any contiguous subarray of the form  $M[i..i', j..j']$ . In this problem we'll develop an algorithm for the maximum subarray problem that runs in  $O(n^3)$  time.

The algorithm is a combination of divide and conquer and dynamic programming. Let  $L$  be a horizontal line through  $M$  that splits the rows (roughly) in half. After some preprocessing, the algorithm finds the maximum-sum subarray that crosses  $L$ , the maximum-sum subarray above  $L$ , and the maximum-sum subarray below  $L$ . The first subarray is found by dynamic programming; the last two subarrays are found recursively.

- (a) For any indices  $i$  and  $j$ , let  $\text{Sum}(i, j)$  denote the sum of all elements in the subarray  $M[1..i, 1..j]$ . Describe an algorithm to compute  $\text{Sum}(i, j)$  for all indices  $i$  and  $j$  in  $O(n^2)$  time.
- (b) Describe a simple(!!) algorithm to solve the maximum subarray problem in  $O(n^4)$  time, using the output of your algorithm for part (a).
- (c) Describe an algorithm to find the maximum-sum subarray *that crosses L* in  $O(n^3)$  time, using the output of your algorithm for part (a). [Hint: Consider the top half and the bottom half of  $M$  separately.]
- (d) Describe a divide-and-conquer algorithm to find the maximum-sum subarray in  $M$  in  $O(n^3)$  time, using your algorithm for part (c) as a subroutine. [Hint: Why is the running time  $O(n^3)$  and not  $O(n^3 \log n)$ ?]

In fact, the subproblem in part (c) — and thus the entire maximum subarray problem — can be solved in  $n^3/2^{\Omega(\sqrt{\log n})}$  time using a recent algorithm of Ryan Williams. Williams' algorithm can also be used to compute all-pairs shortest paths in the same slightly subcubic running time. The divide-and-conquer strategy itself is due to Tadao Takaoka.

There is a simpler  $O(n^3)$ -time algorithm for the maximum subarray problem, based on Kadane's [O\(n\)-time algorithm for the one-dimensional problem](#). (For every pair of indices  $i$  and  $i'$ , find the best subarray of the form  $M[i..i', j..j']$  in  $O(n)$  time.) It's unclear whether this approach can be sped up using Williams' algorithm (or its predecessors) without the divide-and-conquer layer.

An algorithm for the maximum subarray problem (or all-pairs shortest paths) that runs in  $O(n^{2.9999999})$  time would be a major breakthrough.

2. The Doctor and River Song decide to play a game on a directed acyclic graph  $G$ , which has one source  $s$  and one sink  $t$ .<sup>□</sup>

Each player has a token on one of the vertices of  $G$ . At the start of the game, The Doctor's token is on the source vertex  $s$ , and River's token is on the sink vertex  $t$ . The players alternate turns, with The Doctor moving first. On each of his turns, the Doctor moves his token forward along a directed edge; on each of her turns, River moves her token *backward* along a directed edge.

If the two tokens ever meet on the same vertex, River wins the game. (“Hello, Sweetie!”) If the Doctor's token reaches  $t$  or River's token reaches  $s$  before the two tokens meet, then the Doctor wins the game.

Describe and analyze an algorithm to determine who wins this game, assuming both players play perfectly. That is, if the Doctor can win *no matter how River moves*, then your algorithm should output “Doctor”, and if River can win *no matter how the Doctor moves*, your algorithm should output “River”. (Why are these the only two possibilities?) The input to your algorithm is the graph  $G$ .

---

<sup>□</sup>The labels  $s$  and  $t$  may be abbreviations for the Untempered Schism and the Time Vortex, or the Shining World of the Seven Systems (otherwise known as Gallifrey) and Trenzalore, or Skaro and Telos, or Something else Timey-wimey.

Due Wednesday, February 15, 2017 at 8pm

---

o. [Warmup only; do not submit solutions]

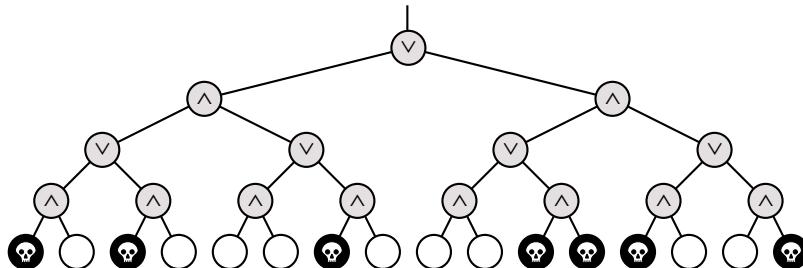
After sending his loyal friends Rosencrantz and Guildenstern off to Norway, Hamlet decides to amuse himself by repeatedly flipping a fair coin until the sequence of flips satisfies some condition. For each of the following conditions, compute the *exact* expected number of flips until that condition is met.

- (a) Hamlet flips heads.
- (b) Hamlet flips both heads and tails (in different flips, of course).
- (c) Hamlet flips heads twice.
- (d) Hamlet flips heads twice in a row.
- (e) Hamlet flips heads followed immediately by tails.
- (f) Hamlet flips more heads than tails.
- (g) Hamlet flips the same positive number of heads and tails.

[Hint: Be careful! If you're relying on intuition instead of a proof, you're probably wrong.]

1. Consider the following non-standard algorithm for shuffling a deck of  $n$  cards, initially numbered in order from 1 on the top to  $n$  on the bottom. At each step, we remove the top card from the deck and *insert* it randomly back into the deck, choosing one of the  $n$  possible positions uniformly at random. The algorithm ends immediately after we pick up card  $n - 1$  and insert it randomly into the deck.
  - (a) Prove that this algorithm uniformly shuffles the deck, meaning each permutation of the deck has equal probability. [Hint: Prove that at all times, the cards below card  $n - 1$  are uniformly shuffled.]
  - (b) What is the *exact* expected number of steps executed by the algorithm? [Hint: Split the algorithm into phases that end when card  $n - 1$  changes position.]

2. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with  $4^n$  leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after  $2n$  moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.



You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are AND gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

- (a) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy!]*
- (b) Unfortunately, Death won't give you enough time to look at every node in the tree. Describe a *randomized* algorithm that determines whether you can win in  $O(3^n)$  expected time. *[Hint: Consider the case  $n = 1$ .]*
- \*(c) **[Extra credit]** Describe and analyze a randomized algorithm that determines whether you can win in  $O(c^n)$  expected time, for some constant  $c < 3$ . *[Hint: You may not need to change your algorithm from part (b) at all!]*

3. The following randomized variant of “one-armed quicksort” selects the  $k$ th smallest element in an unsorted array  $A[1..n]$ . As usual,  $\text{PARTITION}(A[1..n], p)$  partitions the array  $A$  into three parts by comparing the pivot element  $A[p]$  to every other element, using  $n - 1$  comparisons, and returns the new index of the pivot element.

```
QUICKSELECT( $A[1..n]$ ,  $k$ ) :  
     $r \leftarrow \text{PARTITION}(A[1..n], \text{RANDOM}(n))$   
    if  $k < r$   
        return QUICKSELECT( $A[1..r - 1]$ ,  $k$ )  
    else if  $k > r$   
        return QUICKSELECT( $A[r + 1..n]$ ,  $k - r$ )  
    else  
        return  $A[k]$ 
```

- (a) State a recurrence for the expected running time of QUICKSELECT, as a function of  $n$  and  $k$ .
- (b) What is the *exact* probability that QUICKSELECT compares the  $i$ th smallest and  $j$ th smallest elements in the input array? The correct answer is a simple function of  $i$ ,  $j$ , and  $k$ . [Hint: Check your answer by trying a few small examples.]
- (c) What is the *exact* probability that in one of the recursive calls to QUICKSELECT, the first argument is the subarray  $A[i..j]$ ? The correct answer is a simple function of  $i$ ,  $j$ , and  $k$ . [Hint: Check your answer by trying a few small examples.]
- (d) Show that for any  $n$  and  $k$ , the expected running time of QUICKSELECT is  $\Theta(n)$ . You can use either the recurrence from part (a) or the probabilities from part (b) or (c).

# CS 473 ✦ Spring 2017

## ❖ Homework 4 ❖

Due Wednesday, March 1, 2017 at 8pm

- 
1. Recall that a *priority search tree* is a binary tree in which every node has both a *search key* and a *priority*, arranged so that the tree is simultaneously a binary search tree for the keys and a min-heap for the priorities. A **heater** is a priority search tree in which the *priorities* are given by the user, and the *search keys* are distributed uniformly and independently at random in the real interval  $[0, 1]$ . Intuitively, a heater is a sort of anti-treap.

The following problems consider an  $n$ -node heater  $T$  whose priorities are the integers from 1 to  $n$ . We identify nodes in  $T$  by their *priorities*; thus, “node 5” means the node in  $T$  with *priority* 5. For example, the min-heap property implies that node 1 is the root of  $T$ . Finally, let  $i$  and  $j$  be integers with  $1 \leq i < j \leq n$ .

- (a) What is the *exact* expected depth of node  $j$  in an  $n$ -node heater? Answering the following subproblems will help you:
    - i. Prove that in a random permutation of the  $(i + 1)$ -element set  $\{1, 2, \dots, i, j\}$ , elements  $i$  and  $j$  are adjacent with probability  $2/(i + 1)$ .
    - ii. Prove that node  $i$  is an ancestor of node  $j$  with probability  $2/(i + 1)$ . [Hint: Use the previous question!]
    - iii. What is the probability that node  $i$  is a *descendant* of node  $j$ ? [Hint: Do **not** use the previous question!]
  - (b) Describe and analyze an algorithm to insert a new item into a heater. **Analyze the expected running time as a function of the number of nodes.**
  - (c) Describe an algorithm to delete the minimum-priority item (the root) from an  $n$ -node heater. What is the expected running time of your algorithm?
2. Suppose we are given a coin that may or may not be biased, and we would like to compute an accurate *estimate* of the probability of heads. Specifically, if the actual unknown probability of heads is  $p$ , we would like to compute an estimate  $\tilde{p}$  such that

$$\Pr[|\tilde{p} - p| > \varepsilon] < \delta$$

where  $\varepsilon$  is a given **accuracy** or **error** parameter, and  $\delta$  is a given **confidence** parameter.

The following algorithm is a natural first attempt; here `FLIP()` returns the result of an independent flip of the unknown coin.

<code>MEANESTIMATE(<math>\varepsilon</math>):</code>
<code>count</code> $\leftarrow 0$
<code>for</code> $i \leftarrow 1$ to $N$
<code>if</code> <code>FLIP()</code> = HEADS
<code>count</code> $\leftarrow$ <code>count</code> + 1
<code>return</code> <code>count</code> / $N$

- (a) Let  $\tilde{p}$  denote the estimate returned by `MEANESTIMATE( $\varepsilon$ )`. Prove that  $E[\tilde{p}] = p$ .

- (b) Prove that if we set  $N = \lceil \alpha/\varepsilon^2 \rceil$  for some appropriate constant  $\alpha$ , then we have  $\Pr[|\tilde{p} - p| > \varepsilon] < 1/4$ . [Hint: Use Chebyshev's inequality.]
- (c) We can increase the previous estimator's confidence by running it multiple times, independently, and returning the *median* of the resulting estimates.

```

MEDIANOFMEANSESTIMATE( $\delta, \varepsilon$ ):
    for  $j \leftarrow 1$  to  $K$ 
         $estimate[j] \leftarrow \text{MEANESTIMATE}(\varepsilon)$ 
    return  $\text{MEDIAN}(estimate[1..K])$ 

```

Let  $p^*$  denote the estimate returned by  $\text{MEDIANOFMEANSESTIMATE}(\delta, \varepsilon)$ . Prove that if we set  $N = \lceil \alpha/\varepsilon^2 \rceil$  (inside  $\text{MEANESTIMATE}$ ) and  $K = \lceil \beta \ln(1/\delta) \rceil$ , for some appropriate constants  $\alpha$  and  $\beta$ , then  $\Pr[|p^* - p| > \varepsilon] < \delta$ . [Hint: Use Chernoff bounds.]

# CS 473 ✦ Spring 2017

## ∞ Homework 5 ∞

Due Wednesday, March 8, 2017 at 8pm

---

1. **Reservoir sampling** is a method for choosing an item uniformly at random from an arbitrarily long stream of data.

```
GETONESAMPLE(stream  $S$ ):
     $\ell \leftarrow 0$ 
    while  $S$  is not done
         $x \leftarrow$  next item in  $S$ 
         $\ell \leftarrow \ell + 1$ 
        if  $\text{RANDOM}(\ell) = 1$ 
             $sample \leftarrow x$       (*)
        return  $sample$ 
```

At the end of the algorithm, the variable  $\ell$  stores the length of the input stream  $S$ ; this number is *not* known to the algorithm in advance. If  $S$  is empty, the output of the algorithm is (correctly!) undefined.

In the following, consider an arbitrary non-empty input stream  $S$ , and let  $n$  denote the (unknown) length of  $S$ .

- Prove that the item returned by  $\text{GETONESAMPLE}(S)$  is chosen uniformly at random from  $S$ .
- What is the *exact* expected number of times that  $\text{GETONESAMPLE}(S)$  executes line (\*)?
- What is the *exact* expected value of  $\ell$  when  $\text{GETONESAMPLE}(S)$  executes line (\*) for the *last* time?
- What is the *exact* expected value of  $\ell$  when either  $\text{GETONESAMPLE}(S)$  executes line (\*) for the *second* time (or the algorithm ends, whichever happens first)?
- Describe and analyze an algorithm that returns a subset of  $k$  distinct items chosen uniformly at random from a data stream of length at least  $k$ . The integer  $k$  is given as part of the input to your algorithm. Prove that your algorithm is correct.

For example, if  $k = 2$  and the stream contains the sequence  $\langle \spadesuit, \heartsuit, \diamondsuit, \clubsuit \rangle$ , the algorithm should return the subset  $\{\diamondsuit, \clubsuit\}$  with probability  $1/6$ .

2. **Tabulated hashing** uses tables of random numbers to compute hash values. Suppose  $|\mathcal{U}| = 2^w \times 2^w$  and  $m = 2^\ell$ , so the items being hashed are pairs of  $w$ -bit strings (or  $2w$ -bit strings broken in half) and hash values are  $\ell$ -bit strings.

Let  $A[0..2^w - 1]$  and  $B[0..2^w - 1]$  be arrays of independent random  $\ell$ -bit strings, and define the hash function  $h_{A,B} : \mathcal{U} \rightarrow [m]$  by setting

$$h_{A,B}(x, y) := A[x] \oplus B[y]$$

where  $\oplus$  denotes bit-wise exclusive-or. Let  $\mathcal{H}$  denote the set of all possible functions  $h_{A,B}$ . Filling the arrays  $A$  and  $B$  with independent random bits is equivalent to choosing a hash function  $h_{A,B} \in \mathcal{H}$  uniformly at random.

- (a) Prove that  $\mathcal{H}$  is 2-uniform.
- (b) Prove that  $\mathcal{H}$  is 3-uniform. [*Hint: Solve part (a) first.*]
- (c) Prove that  $\mathcal{H}$  is **not** 4-uniform.

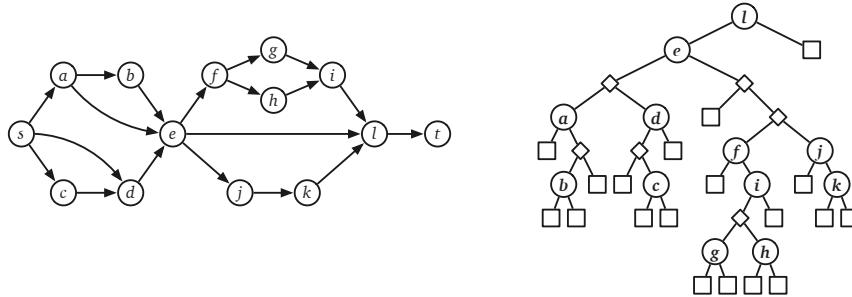
Yes, “see part (b)” is worth full credit for part (a), but only if your solution to part (b) is correct.

**CS 473 ✦ Spring 2017**  
**❖ Homework 6 ❖**

Due Wednesday, March 16, 2017 at 8pm

1. Suppose you are given a directed graph  $G = (V, E)$ , two vertices  $s$  and  $t$ , a capacity function  $c: E \rightarrow \mathbb{R}^+$ , and a second function  $f: E \rightarrow \mathbb{R}$ . Describe and analyze an algorithm to determine whether  $f$  is a maximum  $(s, t)$ -flow in  $G$ . [Hint: Don't make any "obvious" assumptions!]
  
2. Suppose you are given a flow network  $G$  with **integer** edge capacities and an **integer** maximum flow  $f^*$  in  $G$ . Describe algorithms for the following operations:
  - (a) **INCREMENT( $e$ ):** Increase the capacity of edge  $e$  by 1 and update the maximum flow.
  - (b) **DECREMENT( $e$ ):** Decrease the capacity of edge  $e$  by 1 and update the maximum flow.
 Both algorithms should modify  $f^*$  so that it is still a maximum flow, but more quickly than recomputing a maximum flow from scratch.
  
3. An  **$(s, t)$ -series-parallel** graph is a directed acyclic graph with two designated vertices  $s$  (the *source*) and  $t$  (the *target* or *sink*) and with one of the following structures:
  - **Base case:** A single directed edge from  $s$  to  $t$ .
  - **Series:** The union of an  $(s, u)$ -series-parallel graph and a  $(u, t)$ -series-parallel graph that share a common vertex  $u$  but no other vertices or edges.
  - **Parallel:** The union of two smaller  $(s, t)$ -series-parallel graphs with the same source  $s$  and target  $t$ , but with no other vertices or edges in common.

Every  $(s, t)$ -series-parallel graph  $G$  can be represented by a **decomposition tree**, which is a binary tree with three types of nodes: leaves corresponding to single edges in  $G$ , series nodes (each labeled by some vertex), and parallel nodes (unlabeled).



An series-parallel graph and its decomposition tree.

- (a) Suppose you are given a directed graph  $G$  with two special vertices  $s$  and  $t$ . Describe and analyze an algorithm that either builds a decomposition tree for  $G$  or correctly reports that  $G$  is not  $(s, t)$ -series-parallel. [Hint: Build the tree from the bottom up.]
- (b) Describe and analyze an algorithm to compute a maximum  $(s, t)$ -flow in a given  $(s, t)$ -series-parallel flow network with arbitrary edge capacities. [Hint: In light of part (a), you can assume that you are actually given the decomposition tree.]

## CS 473 ✦ Spring 2017

### ❖ Homework 7 ❖

Due Wednesday, March 29, 2017 at 8pm

- 
1. Suppose we are given an array  $A[1..m][1..n]$  of non-negative real numbers. We want to *round*  $A$  to an integer matrix, by replacing each entry  $x$  in  $A$  with either  $\lfloor x \rfloor$  or  $\lceil x \rceil$ , without changing the sum of entries in any row or column of  $A$ . For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

- (a) Describe and analyze an efficient algorithm that either rounds  $A$  in this fashion, or correctly reports that no such rounding is possible.
- (b) Prove that a legal rounding is possible *if and only if* the sum of entries in each row is an integer, and the sum of entries in each column is an integer. In other words, prove that either your algorithm from part (a) returns a legal rounding, or a legal rounding is *obviously* impossible.
2. Quentin, Alice, and the other Brakebills Physical Kids are planning an excursion through the Netherlands to Fillory. The Netherlands is a vast, deserted city composed of several plazas, each containing a single fountain that can magically transport people to a different world. Adjacent plazas are connected by gates, which have been cursed by the Beast. The gates between plazas are open only for five minutes every hour, all simultaneously—from 12:00 to 12:05, then from 1:00 to 1:05, and so on—and are otherwise locked. During those five minutes, if more than one person passes through any single gate, the Beast will detect their presence. □ Moreover, anyone attempting to open a locked gate, or attempting to pass through more than one gate within the same five-minute period will turn into a niffin. □ However, any number of people can safely pass through *different* gates at the same time and/or pass through the same gate at *different* times.

You are given a map of the Netherlands, which is a graph  $G$  with a vertex for each fountain and an edge for each gate, with the fountains to Earth and Fillory clearly marked. Suppose you are also given a positive integer  $h$ . Describe and analyze an algorithm to compute the maximum number of people that can walk from the Earth fountain to the Fillory fountain in at most  $h$  hours—that is, after the gates have opened at most  $h$  times—without anyone alerting the Beast or turning into a niffin. [Hint: Build a different graph.]

---

□This is very bad.

□This is very very bad.

**Rubric (graph reductions):** For a problem worth 10 points, solved by reduction to maximum flow:

- 2 points for a complete description of the relevant flow network, specifying the set of vertices, the set of edges (being careful about direction), the source and target vertices  $s$  and  $t$ , and the capacity of every edge. (If the flow network is part of the original input, just say that.)
- 1 point for a description of the algorithm to construct this flow network from the stated input. This could be as simple as “We can construct the flow network in  $O(n^3)$  time by brute force.”
- 1 point for precisely specifying the problem to be solved on the flow network (for example: “maximum flow from  $x$  to  $y$ ”) and the algorithm (For example: “Ford-Fulkerson” or “Orlin”) to solve that problem. Do not regurgitate the details of the maximum-flow algorithm itself.
- 1 point for a description of the algorithm to extract the answer to the stated problem from the maximum flow. This could be as simple as “Return TRUE if the maximum flow value is at least 42 and FALSE otherwise.”
- **4 points for a proof that your reduction is correct.** This proof will almost always have two components (worth 2 points each). For example, if your algorithm returns a boolean, you should prove that its True answers are correct and that its False answers are correct. If your algorithm returns a number, you should prove that number is neither too large nor too small.
- 1 point for the running time of the overall algorithm, expressed as a function of the original input parameters, *not* just the number of vertices and edges in your flow network. You may assume that maximum flows can be computed in  $O(VE)$  time.

Reductions to other flow-based problems described in class or in the notes (for example: edge-disjoint paths, maximum bipartite matching, minimum-cost circulation) or to other standard graph problems (for example: reachability, topological sort, minimum spanning tree, all-pairs shortest paths) have similar requirements.

**CS 473 ✦ Spring 2017**  
**❖ Homework 8 ❖**

Due Wednesday, April 12, 2017 at 8pm

---

1. Recall that a ***path cover*** of a directed acyclic graph is a collection of directed paths, such that every vertex in  $G$  appears in at least one path. We previously saw how to compute *disjoint* path covers (where each vertex lies on *exactly* one path) by reduction to maximum bipartite matching. Your task in this problem is to compute path covers *without* the disjointness constraint.
  - (a) Suppose you are given a dag  $G$  with a unique source  $s$  and a unique sink  $t$ . Describe an algorithm to find the smallest path cover of  $G$  in which every path starts at  $s$  and ends at  $t$ .
  - (b) Describe an algorithm to find the smallest path cover of an arbitrary dag  $G$ , with no additional restrictions on the paths. [Hint: Use part (a).]
2. Recall that an  ***$(s, t)$ -series-parallel*** graph is an directed acyclic graph with two designated vertices  $s$  (the *source*) and  $t$  (the *target* or *sink*) and with one of the following structures:
  - **Base case:** A single directed edge from  $s$  to  $t$ .
  - **Series:** The union of an  $(s, u)$ -series-parallel graph and a  $(u, t)$ -series-parallel graph that share a common vertex  $u$  but no other vertices or edges.
  - **Parallel:** The union of two smaller  $(s, t)$ -series-parallel graphs with the same source  $s$  and target  $t$ , but with no other vertices or edges in common.

Any series-parallel graph can be represented by a binary *decomposition tree*, whose interior nodes correspond to series compositions and parallel compositions, and whose leaves correspond to individual edges. In a previous homework, we saw how to construct the decomposition tree for any series-parallel graph in  $O(V + E)$  time, and then how to compute a maximum  $(s, t)$ -flow in  $O(V + E)$  time.

Describe an efficient algorithm to compute a *minimum-cost* maximum flow from  $s$  to  $t$  in an  $(s, t)$ -series-parallel graph  $G$  in which every edge has capacity 1 and arbitrary cost. [Hint: First consider the special case where  $G$  has only two vertices but lots of edges.]

3. Every year, Professor Dumbledore assigns the instructors at Hogwarts to various faculty committees. There are  $n$  faculty members and  $c$  committees. Each committee member has submitted a list of their *prices* for serving on each committee; each price could be positive, negative, zero, or even infinite. For example, Professor Snape might declare that he would serve on the Student Recruiting Committee for 1000 Galleons, that he would *pay* 10000 Galleons to serve on the Defense Against the Dark Arts Course Revision Committee, and that he would not serve on the Muggle Relations committee for any price.

Conversely, Dumbledore knows how many instructors are needed for each committee, as well as a list of instructors who would be suitable members for each committee. (For example: “Dark Arts Revision: 5 members, anyone but Snape.”) If Dumbledore assigns an instructor to a committee, he must pay that instructor’s price from the Hogwarts treasury.

Dumbledore needs to assign instructors to committees so that (1) each committee is full, (3) no instructor is assigned to more than three committees, (2) only suitable and willing instructors are assigned to each committee, and (4) the total cost of the assignment is as small as possible. Describe and analyze an efficient algorithm that either solves Dumbledore's problem, or correctly reports that there is no valid assignment whose total cost is finite.

## CS 473 ✦ Spring 2017

## ❖ Homework 9 ❖

Due Wednesday, April 19, 2017 at 8pm

---

1. Suppose you are given an arbitrary directed graph  $G = (V, E)$  with arbitrary edge weights  $\ell : E \rightarrow \mathbb{R}$ . Each edge in  $G$  is colored either red, white, or blue to indicate how you are permitted to modify its weight:

- You may increase, but not decrease, the length of any red edge.
- You may decrease, but not increase, the length of any blue edge.
- You may not change the length of any black edge.

The **cycle nullification** problem asks whether it is possible to modify the edge weights—subject to these color constraints—so that *every cycle in  $G$  has length 0*. Both the given weights and the new weights of the individual edges can be positive, negative, or zero. To keep the following problems simple, assume that  $G$  is strongly connected.

- (a) Describe a linear program that is feasible if and only if it is possible to make every cycle in  $G$  have length 0. [Hint: Pick an arbitrary vertex  $s$ , and let  $\text{dist}(v)$  denote the length of every walk from  $s$  to  $v$ .]
  - (b) Construct the dual of the linear program from part (a). [Hint: Choose a convenient objective function for your primal LP.]
  - (c) Give a self-contained description of the combinatorial problem encoded by the dual linear program from part (b), and prove *directly* that it is equivalent to the original cycle nullification problem. Do not use the words “linear”, “program”, or “dual”. Yes, you have seen this problem before.
  - (d) Describe and analyze an algorithm to determine in  $O(EV)$  time whether it is possible to make every cycle in  $G$  have length 0, using your dual formulation from part (c). Do not use the words “linear”, “program”, or “dual”.
2. There is no problem 2.

**CS 473 ✦ Spring 2017**  
**❖ Homework 10 ❖**

Due Wednesday, April 26, 2017 at 8pm

- 
1. An *integer linear program* is a linear program with the additional explicit constraint that the variables must take *only* integer values. The ILP-FEASIBILITY problem asks whether there is an integer vector that satisfies a given system of linear inequalities—or more concisely, whether a given integer linear program is feasible.

Describe a polynomial-time reduction from 3SAT to ILP-FEASIBILITY. Your reduction implies that ILP-FEASIBILITY is NP-hard.

2. There are two different versions of the Hamiltonian cycle problem, one for directed graphs and one for undirected graphs. We saw a proof in class (and there are two proofs in the notes) that the *directed* Hamiltonian cycle problem is NP-hard.
  - (a) Describe a polynomial-time reduction from the *undirected* Hamiltonian cycle problem to the *directed* Hamiltonian cycle problem. Prove your reduction is correct.
  - (b) Describe a polynomial-time reduction from the *directed* Hamiltonian cycle problem to the *undirected* Hamiltonian cycle problem. Prove your reduction is correct.
  - (c) Which of these two reductions implies that the *undirected* Hamiltonian cycle problem is NP-hard?

3. Recall that a 3CNF formula is a conjunction (AND) of several distinct clauses, each of which is a disjunction (OR) of exactly three distinct literals, where each literal is either a variable or its negation.

Suppose you are given a magic black box that can determine **in polynomial time**, whether an arbitrary given 3CNF formula is satisfiable. Describe and analyze a **polynomial-time** algorithm that either computes a satisfying assignment for a given 3CNF formula or correctly reports that no such assignment exists, using the magic black box as a subroutine. [Hint: Call the magic black box more than once. First imagine an even more magical black box that can decide SAT for arbitrary boolean formulas, not just 3CNF formulas.]

**Rubric (for all polynomial-time reductions):** 10 points =

- + 3 points for the reduction itself
  - For an NP-hardness proof, the reduction must be from a known NP-hard problem. You can use any of the NP-hard problems listed in the lecture notes (except the one you are trying to prove NP-hard, of course).
- + 3 points for the “if” proof of correctness
- + 3 points for the “only if” proof of correctness
- + 1 point for writing “polynomial time”
  - An incorrect polynomial-time reduction that still satisfies half of the correctness proof is worth at most 4/10.
  - A reduction in the wrong direction is worth 0/10.

**CS 473 ✦ Spring 2017**  
**❖ Homework 11 ❖**

“Due” Wednesday, May 3, 2017 at 8pm

---

**This homework will not be graded.**

**However, material covered by this homework *may* appear on the final exam.**

---

1. Let  $\Phi$  be a boolean formula in conjunctive normal form, with exactly three literals in each clause. Recall that an assignment of boolean values to the variables in  $\Phi$  *satisfies* a clause if at least one of its literals is TRUE. The **maximum satisfiability problem** for 3CNF formulas, usually called MAX3SAT, asks for the maximum number of clauses that can be simultaneously satisfied by a single assignment.

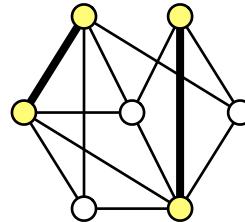
Solving MAX3SAT exactly is clearly also NP-hard; this question asks about approximation algorithms. Let  $\text{Max3Sat}(\Phi)$  denote the maximum number of clauses in  $\Phi$  that can be simultaneously satisfied by one variable assignment.

- (a) Suppose we assign variables in  $\Phi$  to be TRUE or FALSE using independent fair coin flips. Prove that the expected number of satisfied clauses is at least  $\frac{7}{8}\text{Max3Sat}(\Phi)$ .  
(b) Let  $k^+$  denote the number of clauses satisfied by setting every variable in  $\Phi$  to TRUE, and let  $k^-$  denote the number of clauses satisfied by setting every variable in  $\Phi$  to FALSE. Prove that  $\max\{k^+, k^-\} \geq \text{Max3Sat}(\Phi)/2$ .  
(c) Let  $\text{Min3Unsat}(\Phi)$  denote the *minimum* number of clauses that can be simultaneously left *unsatisfied* by a single assignment. Prove that it is NP-hard to approximate  $\text{Min3Unsat}(\Phi)$  within a factor of  $10^{10^{100}}$ .
2. Consider the following algorithm for approximating the minimum vertex cover of a connected graph  $G$ : **Return the set of all non-leaf nodes of an arbitrary depth-first spanning tree.** (Recall that a depth-first spanning tree is a rooted tree; the root is not considered a leaf, even if it has only one neighbor in the tree.)
  - (a) Prove that this algorithm returns a vertex cover of  $G$ .
  - (b) Prove that this algorithm returns a 2-approximation to the smallest vertex cover of  $G$ .
  - (c) Describe an infinite family of connected graphs for which this algorithm returns a vertex cover of size *exactly*  $2 \cdot \text{OPT}$ . This family implies that the analysis in part (b) is tight. [Hint: First find just **one** such graph, with few vertices.]

3. Consider the following modification of the “dumb” 2-approximation algorithm for minimum vertex cover that we saw in class. The only change is that we return a set of edges instead of a set of vertices.

```
APPROXMINMAXMATCHING( $G$ ):
 $M \leftarrow \emptyset$ 
while  $G$  has at least one edge
   $uv \leftarrow$  any edge in  $G$ 
   $G \leftarrow G \setminus \{u, v\}$ 
   $M \leftarrow M \cup \{uv\}$ 
return  $M$ 
```

- (a) Prove that the output subgraph  $M$  is a *matching*—no pair of edges in  $M$  share a common vertex.
- (b) Prove that  $M$  is a *maximal matching*— $M$  is not a proper subgraph of another matching in  $G$ .
- (c) Prove that  $M$  contains at most twice as many edges as the *smallest maximal matching* in  $G$ .
- (d) Describe an infinite family of graphs  $G$  such that the matching returned by APPROXMINMAXMATCHING( $G$ ) contains exactly twice as many edges as the smallest maximum matching in  $G$ . This family implies that the analysis in part (c) is tight. [Hint: First find just **one** such graph, with few vertices.]



The smallest maximal matching in a graph.

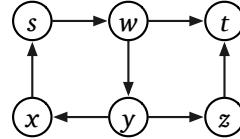
**CS 473 ✦ Spring 2017**  
**❖ Midterm 1 ❖**  
**February 21, 2016**

---

- Recall that a **walk** in a directed graph  $G$  is an arbitrary sequence of vertices  $v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_k$ , such that  $v_{i-1} \rightarrow v_i$  is an edge in  $G$  for every index  $i$ . A **path** is a walk in which no vertex appears more than once.

Suppose you are given a directed graph  $G = (V, E)$  and two vertices  $s$  and  $t$ . Describe and analyze an algorithm to determine if there is a walk in  $G$  from  $s$  to  $t$  whose length is a multiple of 3.

For example, given the graph shown below, with the indicated vertices  $s$  and  $t$ , your algorithm should return TRUE, because the walk  $s \rightarrow w \rightarrow y \rightarrow x \rightarrow s \rightarrow w \rightarrow t$  has length 6.



[Hint: Build a (different) graph.]

- A **shuffle** of two strings  $X$  and  $Y$  is formed by interspersing the characters into a new string, keeping the characters of  $X$  and  $Y$  in the same order. A **smooth** shuffle of  $X$  and  $Y$  is a shuffle of  $X$  and  $Y$  that never uses more than two consecutive symbols of either string. For example,

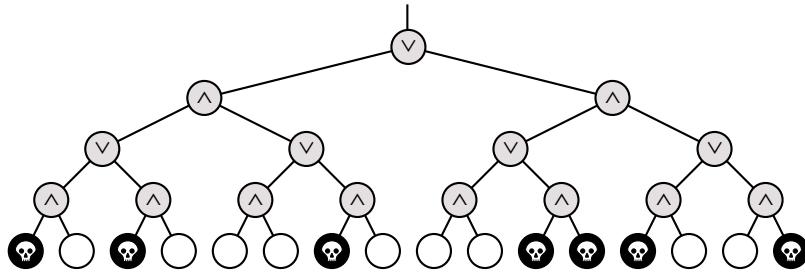
- **prDoygnaraMmmIcng** is a smooth shuffle of **DYNAMIC** and **programming**.
- **DYprNogrAAmMmICing** is a shuffle of **DYNAMIC** and **programming**, but it is not a smooth shuffle (because of the substrings **ogr** and **ing**).

Describe and analyze an algorithm to decide, given three strings  $X$ ,  $Y$ , and  $Z$ , whether  $Z$  is a smooth shuffle of  $X$  and  $Y$ .

- (a) Describe an algorithm that simulates a fair coin, using independent rolls of a fair three-sided die as your only source of randomness. Your algorithm should return either HEADS or TAILS, each with probability 1/2.  
(b) What is the expected number of die rolls performed by your algorithm in part (a)?  
(c) Describe an algorithm that simulates a fair three-sided die, using independent fair coin flips as your only source of randomness. Your algorithm should return either 1, 2, or 3, each with probability 1/3.  
(d) What is the expected number of coin flips performed by your algorithm in part (c)?

4. Death knocks on Dirk Gently's door one cold blustery morning and challenges him to a game. Emboldened by his experience with algorithms students, Death presents Dirk with a complete binary tree with  $4^n$  leaves, each colored either black or white. There is a token at the root of the tree. To play the game, Dirk and Death will take turns moving the token from its current node to one of its children. The game will end after  $2n$  moves, when the token lands on a leaf. If the final leaf is black, Dirk dies; if it's white, Dirk lives forever. Dirk moves first, so Death gets the last turn.

(Yes, this is precisely the same game from Homework 3.)



Unfortunately, Dirk slept through Death's explanation of the rules, so he decides to just play randomly. Whenever it's Dirk's turn, he flips a fair coin and moves left on heads, or right on tails, confident that the Fundamental Interconnectedness of All Things will keep him alive, unless it doesn't. Death plays much more purposefully, of course, always choosing the move that maximizes the probability that Dirk loses the game.

- (a) Describe an algorithm that computes *the probability* that Dirk wins the game against Death.
- (b) Realizing that Dirk is not taking the game seriously, Death gives up in desperation and decides to also play randomly! Describe an algorithm that computes *the probability* that Dirk wins the game again Death, assuming *both* players flip fair coins to decide their moves.

For both algorithms, the input consists of the integer  $n$  (specifying the depth of the tree) and an array of  $4^n$  bits specifying the colors of leaves in left-to-right order.

**CS 473 ✦ Spring 2017**  
**❖ Midterm 2 ❖**  
**April 4, 2016**

---

1. Let  $G = (V, E)$  be an arbitrary undirected graph. Suppose we color each vertex of  $G$  uniformly and independently at random from a set of three colors: red, green, or blue. An edge of  $G$  is *monochromatic* if both of its endpoints have the same color.

- (a) What is the *exact* expected number of monochromatic edges? (Your answer should be a simple function of  $V$  and  $E$ .)
- (b) For each edge  $e \in E$ , define an indicator variable  $X_e$  that equals 1 if  $e$  is monochromatic and 0 otherwise. **Prove** that

$$\Pr[(X_a = 1) \wedge (X_b = 1)] = \Pr[X_a = 1] \cdot \Pr[X_b = 1]$$

for every pair of edges  $a \neq b$ . This claim implies that the random variables  $X_e$  are pairwise independent.

- (c) **Prove** that there is a graph  $G$  such that

$$\Pr[(X_a = 1) \wedge (X_b = 1) \wedge (X_c = 1)] \neq \Pr[X_a = 1] \cdot \Pr[X_b = 1] \cdot \Pr[X_c = 1]$$

for some triple of distinct edges  $a, b, c$  in  $G$ . This claim implies that the random variables  $X_e$  are *not necessarily* 3-wise independent.

2. The White Rabbit has a very poor memory, and so he is constantly forgetting his regularly scheduled appointments with the Queen of Hearts. In an effort to avoid further beheadings of court officials, The King of Hearts has installed an app on Rabbit's pocket watch to automatically remind Rabbit of any upcoming appointments. For each reminder Rabbit receives, Rabbit has a 50% chance of actually remembering his appointment (decided by an independent fair coin flip).

First, suppose the King of Hearts sends Rabbit  $k$  separate reminders for a *single* appointment.

- (a) What is the *exact* probability that Rabbit will remember his appointment? Your answer should be a simple function of  $k$ .
- (b) What value of  $k$  should the King choose so that the probability that Rabbit will remember this appointment is at least  $1 - 1/n^\alpha$ ? Your answer should be a simple function of  $n$  and  $\alpha$ .

Now suppose the King of Hearts sends Rabbit  $k$  separate reminders for each of  $n$  different appointments. (That's  $nk$  reminders altogether.)

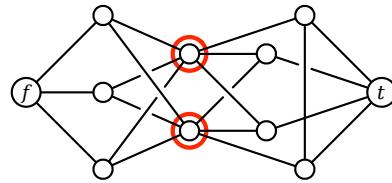
- (c) What is the *exact* expected number of appointments that Rabbit will remember? Your answer should be a simple function of  $n$  and  $k$ .
- (d) What value of  $k$  should the King choose so that the probability that Rabbit remembers *every* appointment is at least  $1 - 1/n^\alpha$ ? Again, your answer should be a simple function of  $n$  and  $\alpha$ .

*[Hint: There is a simple solution that does not use tail inequalities.]*

3. The Island of Sodor is home to an extensive rail network. Recently, several cases of a deadly contagious disease (either swine flu or zombies; reports are unclear) have been reported in the village of Ffarquhar. The controller of the Sodor railway plans to close certain railway stations to prevent the disease from spreading to Tidmouth, his home town. No trains can pass through a closed station. To minimize expense (and public notice), he wants to close as few stations as possible. However, he cannot close the Ffarquhar station, because that would expose him to the disease, and he cannot close the Tidmouth station, because then he couldn't visit his favorite pub.

Describe and analyze an algorithm to find the minimum number of stations that must be closed to block all rail travel from Ffarquhar to Tidmouth. The Sodor rail network is represented by an undirected graph, with a vertex for each station and an edge for each rail connection between two stations. Two special vertices  $f$  and  $t$  represent the stations in Ffarquhar and Tidmouth.

For example, given the following input graph, your algorithm should return the integer 2.



4. The Department of Commuter Silence at Shampoo-Banana University has a flexible curriculum with a complex set of graduation requirements. The department offers  $n$  different courses, and there are  $m$  different requirements. Each requirement specifies a subset of the  $n$  courses and the number of courses that must be taken from that subset. The subsets for different requirements may overlap, but each course can only be used to satisfy *at most one* requirement.

For example, suppose there are  $n = 5$  courses  $A, B, C, D, E$  and  $m = 2$  graduation requirements:

- You must take at least 2 courses from the subset  $\{A, B, C\}$ .
- You must take at least 2 courses from the subset  $\{C, D, E\}$ .

Then a student who has only taken courses  $B, C, D$  cannot graduate, but a student who has taken either  $A, B, C, D$  or  $B, C, D, E$  can graduate.

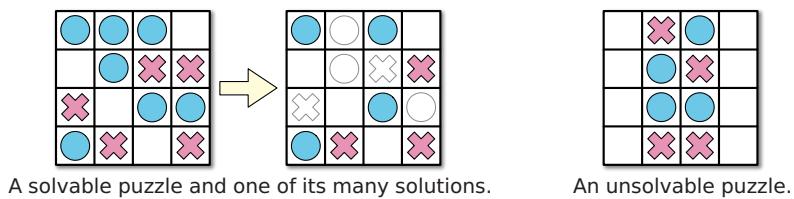
Describe and analyze an algorithm to determine whether a given student can graduate. The input to your algorithm is the list of  $m$  requirements (each specifying a subset of the  $n$  courses and the number of courses that must be taken from that subset) and the list of courses the student has taken.

CS 473 ✦ Spring 2017  
❖ Final Exam ❖  
May 10, 2017

1. A **three-dimensional matching** in an undirected graph  $G$  is a collection of vertex-disjoint triangles. A three-dimensional matching is *maximal* if it is not a proper subgraph of a larger three-dimensional matching in the same graph.
    - (a) Let  $M$  and  $M'$  be two arbitrary maximal three-dimensional matchings in the same underlying graph  $G$ . **Prove** that  $|M| \leq 3 \cdot |M'|$ .
    - (b) Finding the *largest* three-dimensional matching in a given graph is NP-hard. Describe and analyze a fast 3-approximation algorithm for this problem.
    - (c) Finding the *smallest maximal* three-dimensional matching in a given graph is NP-hard. Describe and analyze a fast 3-approximation algorithm for this problem.
  2. Let  $G = (V, E)$  be an arbitrary dag with a unique source  $s$  and a unique sink  $t$ . Suppose we compute a random walk from  $s$  to  $t$ , where at each node  $v$  (except  $t$ ), we choose an outgoing edge  $v \rightarrow w$  uniformly at random to determine the successor of  $v$ .
    - (a) Describe and analyze an algorithm to compute, for every vertex  $v$ , the probability that the random walk visits  $v$ .
    - (b) Describe and analyze an algorithm to compute the expected number of edges in the random walk.

Assume all relevant arithmetic operations can be performed exactly in  $O(1)$  time.

3. Consider the following solitaire game. The puzzle consists of an  $n \times m$  grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) every row contains at least one stone, and (2) no column contains stones of both colors. For some initial configurations of stones, reaching this goal is impossible.



*Prove* that it is NP-hard to determine, given an initial configuration of red and blue stones, whether the puzzle can be solved.

4. Suppose you are given a bipartite graph  $G = (L \sqcup R, E)$  and a maximum matching  $M$  in  $G$ . Describe and analyze fast algorithms for the following problems:

- (a)  $\text{INSERT}(e)$ : Insert a new edge  $e$  into  $G$  and update the maximum matching. (You can assume that  $e$  is not already an edge in  $G$ , and that  $G + e$  is still bipartite.)
- (b)  $\text{DELETE}(e)$ : Delete the existing edge  $e$  from  $G$  and update the maximum matching. (You can assume that  $e$  is in fact an edge in  $G$ .)

Your algorithms should modify  $M$  so that it is still a maximum matching, faster than recomputing a maximum matching from scratch.

5. You are applying to participate in this year's Trial of the Pyx, the annual ceremony where samples of all British coinage are tested, to ensure that they conform as strictly as possible to legal standards. As a test of your qualifications, your interviewer at the Worshipful Company of Goldsmiths has given you a bag of  $n$  commemorative Alan Turing half-guinea coins, exactly two of which are counterfeit. One counterfeit coin is very slightly lighter than a genuine Turing; the other is very slightly heavier. Together, the two counterfeit coins have *exactly* the same weight as two genuine coins. Your task is to identify the two counterfeit coins.

The weight difference between the real and fake coins is too small to be detected by anything other than the Royal Pyx Coin Balance. You can place any two disjoint sets of coins in each of the Balance's two pans; the Balance will then indicate which of the two subsets has larger total weight, or that the two subsets have the same total weight. Unfortunately, each use of the Balance requires completing a complicated authorization form (in triplicate), submitting a blood sample, and scheduling the Royal Bugle Corps, so you *really* want to use the Balance as few times as possible.

- (a) Suppose you *randomly* choose  $n/2$  of your  $n$  coins to put on one pan of the Balance, and put the remaining  $n/2$  coins on the other pan. What is the probability that the two subsets have equal weight?
- (b) Describe and analyze a randomized algorithm to identify the two fake coins. What is the expected number of times your algorithm uses the Balance? To simplify the algorithm, you may assume that  $n$  is a power of 2.

6. Suppose you are given a set  $L$  of  $n$  line segments in the plane, where each segment has one endpoint on the vertical line  $x = 0$  and one endpoint on the vertical line  $x = 1$ , and all  $2n$  endpoints are distinct. Describe and analyze an algorithm to compute the largest subset of  $L$  in which no pair of segments intersects.

### Some Useful Inequalities

Let  $X = \sum_{i=1}^n X_i$ , where each  $X_i$  is a 0/1 random variable, and let  $\mu = E[X]$ .

- **Markov's Inequality:**  $\Pr[X \geq x] \leq \mu/x$  for all  $x > 0$ .
- **Chebyshev's Inequality:** If  $X_1, X_2, \dots, X_n$  are pairwise independent, then for all  $\delta > 0$ :

$$\Pr[X \geq (1 + \delta)\mu] < \frac{1}{\delta^2\mu} \quad \text{and} \quad \Pr[X \leq (1 - \delta)\mu] < \frac{1}{\delta^2\mu}$$

- **Chernoff Bounds:** If  $X_1, X_2, \dots, X_n$  are fully independent, then for all  $0 < \delta \leq 1$ :

$$\Pr[X \geq (1 + \delta)\mu] \leq \exp(-\delta^2\mu/3) \quad \text{and} \quad \Pr[X \leq (1 - \delta)\mu] \leq \exp(-\delta^2\mu/2)$$

### Some Useful Algorithms

- **RANDOM( $k$ ):** Returns an element of  $\{1, 2, \dots, k\}$ , chosen independently and uniformly at random, in  $O(1)$  time. For example, RANDOM(2) can be used for a fair coin flip.
- **Ford and Fulkerson's maximum flow algorithm:** Returns a maximum  $(s, t)$ -flow  $f^*$  in a given flow network in  $O(E \cdot |f^*|)$  time. If all input capacities are integers, then all output flow values are also integers.
- **Orlin's maximum flow algorithm:** Returns a maximum  $(s, t)$ -flow in a given flow network in  $O(VE)$  time. If all input capacities are integers, then all output flow values are also integers.
- **Orlin's minimum-cost flow algorithm:** Returns a minimum-cost flow in a given flow network in  $O(E^2 \log^2 V)$  time. If all input capacities, costs, and balances are integers, then all output flow values are also integers.

### Some Useful NP-hard Problems:

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three distinct literals per clause, does the formula have a satisfying assignment?

**MAXINDEPENDENTSET:** Given an undirected graph  $G$ , what is the size of the largest subset of vertices in  $G$  that have no edges among them?

**MAXCLIQUE:** Given an undirected graph  $G$ , what is the size of the largest complete subgraph of  $G$ ?

**MINVERTEXCOVER:** Given an undirected graph  $G$ , what is the size of the smallest subset of vertices that touch every edge in  $G$ ?

**MINSETCOVER:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subcollection whose union is  $S$ ?

**MINHITTINGSET:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subset of  $S$  that intersects every subset  $S_i$ ?

**3COLOR:** Given an undirected graph  $G$ , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

**HAMILTONIANCYCLE:** Given a graph  $G$  (either directed or undirected), is there a cycle in  $G$  that visits every vertex exactly once?

**FEASIBLEILP:** Given a matrix  $A \in \mathbb{Z}^{n \times d}$  and a vector  $b \in \mathbb{Z}^n$ , determine whether the set of feasible integer points  $\max\{x \in \mathbb{Z}^d \mid Ax \leq b, x \geq 0\}$  is empty.

**HYDRAULICPRESS:** And here we go!

**Common Grading Rubrics**

(For problems out of 10 points)

**General Principles:**

- Faster algorithms are worth more points, and slower algorithms are worth fewer points, typically by 2 or 3 points (out of 10) for each factor of  $n$ . Partial credit is scaled to the new maximum score, and all points above 10 are recorded as extra credit.
- A clear, correct, and correctly analyzed algorithm, no matter how slow, is always worth more than “I don’t know”. An incorrect algorithm, no matter how fast, may be worth nothing.
- Proofs of correctness are required on exams if and only if we explicitly ask for them.

**Dynamic Programming:**

- 3 points for a **clear English specification** of the underlying recursive function = 2 for describing the function itself + 1 for describing how to call the function to get your final answer. We want an English description of the underlying recursive *problem*, not just the algorithm/recurrence. In particular, your description should specify precisely the role of each input parameter. **No credit for the rest of the problem if the English description is missing; this is a Deadly Sin.**
- 4 points for correct recurrence = 1 for base case(s) + 3 for recursive case(s). **No credit for iterative details if the recursive case(s) are incorrect.**
- 3 points for iterative details = 1 for memoization structure + 1 for evaluation order + 1 for time analysis. Complete iterative pseudocode is *not* required for full credit.

**Graph Reductions:**

- 4 points for a complete description of the relevant graph, including vertices, edges (including whether directed or undirected), numerical data (weights, lengths, capacities, costs, balances, and the like), source and target vertices, and so on. If the graph is part of the original input, just say that.
- 4 points for other details of the reduction, including how to build the graph from the original input, the precise problem to be solved on the graph, the precise algorithm used to solve that problem, and how to extract your final answer from the output of that algorithm.
- 2 points for running time of the overall algorithm, expressed as a function of the original input parameters, *not* just the number of vertices and edges in the graph.

**NP-hardness Proofs:**

- 3 points for a complete description of the reduction, including an appropriate NP-hard problem to reduce from, how to transform the input, and how to transform the output.
- 6 points for the proof of correctness = 3 for the “if” part + 3 for the “only if” part.
- 1 points for “polynomial time”.

# CS/ECE 374 A ✦ Spring 2018

## ❖ Homework o ❖

Due Tuesday, January 23, 2018 at 8pm

---

- **Each student must submit individual solutions for this homework.** For all future homeworks, groups of up to three students can submit joint solutions.
  - **Submit your solutions electronically to Gradescope as PDF files.** Submit a separate PDF file for each numbered problem. If you plan to typeset your solutions, please use the `LATEX` solution template on the course web site. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app (or an actual scanner).
  - You are **not** required to sign up on Gradescope or Piazza with your real name and your illinois.edu email address; you may use any email address and alias of your choice. However, to give you credit for the homework, we need to know who Gradescope thinks you are. **Please fill out the web form linked from the course web page.**
- 

### ❖ Some important course policies ❖

- **You may use any source at your disposal**—paper, electronic, or human—but you **must** cite **every** source that you use, and you **must** write everything yourself in your own words. See the academic integrity policies on the course web site for more details.
- The answer “*I don’t know*” (and *nothing else*) is worth 25% partial credit on any required problem or subproblem on any homework or exam. We will accept synonyms like “No idea” or “WTF” or “\(\emptyset\)/”, but you must write *something*.

On the other hand, only the homework problems you submit actually contribute to your overall course grade, so submitting “I don’t know” for an entire numbered homework problem will almost certainly hurt your grade more than submitting nothing at all.

- **Avoid the Three Deadly Sins!** Any homework or exam solution that breaks any of the following rules will be given an **automatic zero**, unless the solution is otherwise perfect. Yes, we really mean it. We’re not trying to be scary or petty (Honest!), but we do want to break a few common bad habits that seriously impede mastery of the course material.

- Always give complete solutions, not just examples.
- Always declare all your variables, in English. In particular, always describe the specific problem your algorithm is supposed to solve.
- Never use weak induction.

---

See the course web site for more information.

If you have any questions about these policies,  
please don’t hesitate to ask in class, in office hours, or on Piazza.

---

1. The famous Basque computational arborist Gorka Oihanean has a favorite 26-node binary tree, in which each node is labeled with a letter of the alphabet. Inorder and postorder traversals of his tree visits the nodes in the following orders:

Inorder: F E V I B H N X G W A Z O D J S R M U T C K Q P L Y

Postorder: F V B I E N A Z W G X J S D M U R O H K C Q Y L P T

- (a) List the nodes in Professor Oihanean's tree according to a preorder traversal.  
 (b) Draw Professor Oihanean's tree.

You do *not* need to prove that your answers are correct.

2. For any string  $w \in \{0, 1\}^*$ , let  $\text{swap}(w)$  denote the string obtained from  $w$  by swapping the first and second symbols, the third and fourth symbols, and so on. For example:

$$\text{swap}(10110001101) = 01110010011.$$

The *swap* function can be formally defined as follows:

$$\text{swap}(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ w & \text{if } w = 0 \text{ or } w = 1 \\ ba \bullet \text{swap}(x) & \text{if } w = abx \text{ for some } a, b \in \{0, 1\} \text{ and } x \in \{0, 1\}^* \end{cases}$$

- (a) Prove by induction that  $|\text{swap}(w)| = |w|$  for every string  $w$ .  
 (b) Prove by induction that  $\text{swap}(\text{swap}(w)) = w$  for every string  $w$ .

You may assume without proof that  $|x \bullet y| = |x| + |y|$ , or any other result proved in class, in lab, or in the lecture notes. Otherwise, your proofs must be formal and self-contained, and they must invoke the *formal* definitions of length  $|w|$ , concatenation  $\bullet$ , and the *swap* function. Do not appeal to intuition!

3. Consider the set of strings  $L \subseteq \{0, 1\}^*$  defined recursively as follows:

- The empty string  $\varepsilon$  is in  $L$ .
- For any string  $x$  in  $L$ , the string  $0x$  is also in  $L$ .
- For any strings  $x$  and  $y$  in  $L$ , the string  $1x1y$  is also in  $L$ .
- These are the only strings in  $L$ .

- (a) Prove that the string 101110101101011 is in  $L$ .  
 (b) Prove that every string  $w \in L$  contains an even number of 1s.  
 (c) Prove that every string  $w \in \{0, 1\}^*$  with an even number of 1s is a member of  $L$ .

Let  $\#(a, w)$  denote the number of times symbol  $a$  appears in string  $w$ ; for example,

$$\#(0, 101110101101011) = 5 \quad \text{and} \quad \#(1, 101110101101011) = 10.$$

You may assume without proof that  $\#(a, uv) = \#(a, u) + \#(a, v)$  for any symbol  $a$  and any strings  $u$  and  $v$ , or any other result proved in class, in lab, or in the lecture notes. Otherwise, your proofs must be formal and self-contained.

Each homework assignment will include at least one solved problem, similar to the problems assigned in that homework, together with the grading rubric we would apply *if* this problem appeared on a homework or exam. These model solutions illustrate our recommendations for structure, presentation, and level of detail in your homework solutions. Of course, the actual content of your solutions won't match the model solutions, because your problems are different!

## Solved Problems

4. The **reversal**  $w^R$  of a string  $w$  is defined recursively as follows:

$$w^R := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ x^R \bullet a & \text{if } w = a \cdot x \end{cases}$$

A **palindrome** is any string that is equal to its reversal, like **AMANAPLANACANALPANAMA**, **RACECAR**, **POOP**, **I**, and the empty string.

- (a) Give a recursive definition of a palindrome over the alphabet  $\Sigma$ .
- (b) Prove  $w = w^R$  for every palindrome  $w$  (according to your recursive definition).
- (c) Prove that every string  $w$  such that  $w = w^R$  is a palindrome (according to your recursive definition).

You may assume without proof the following statements for all strings  $x$ ,  $y$ , and  $z$ :

- Reversal reversal:  $(x^R)^R = x$
- Concatenation reversal:  $(x \bullet y)^R = y^R \bullet x^R$
- Right cancellation: If  $x \bullet z = y \bullet z$ , then  $x = y$ .

### Solution:

- (a) A string  $w \in \Sigma^*$  is a palindrome if and only if either
- $w = \varepsilon$ , or
  - $w = a$  for some symbol  $a \in \Sigma$ , or
  - $w = axa$  for some symbol  $a \in \Sigma$  and some *palindrome*  $x \in \Sigma^*$ .

**Rubric:** 2 points =  $\frac{1}{2}$  for each base case + 1 for the recursive case. No credit for the rest of the problem unless this part is correct.

- (b) Let  $w$  be an arbitrary palindrome.

Assume that  $x = x^R$  for every palindrome  $x$  such that  $|x| < |w|$ .

There are three cases to consider (mirroring the definition of “palindrome”):

- If  $w = \varepsilon$ , then  $w^R = \varepsilon$  by definition, so  $w = w^R$ .
- If  $w = a$  for some symbol  $a \in \Sigma$ , then  $w^R = a$  by definition, so  $w = w^R$ .
- Finally, suppose  $w = axa$  for some symbol  $a \in \Sigma$  and some palindrome

$x \in P$ . In this case, we have

$$\begin{aligned}
 w^R &= (a \cdot x \cdot a)^R \\
 &= (x \cdot a)^R \cdot a && \text{by definition of reversal} \\
 &= a^R \cdot x^R \cdot a && \text{by concatenation reversal} \\
 &= a \cdot x^R \cdot a && \text{by definition of reversal} \\
 &= a \cdot x \cdot a && \text{by the inductive hypothesis} \\
 &= w && \text{by assumption}
 \end{aligned}$$

In all three cases, we conclude that  $w = w^R$ . ■

**Rubric:** 4 points: standard induction rubric (scaled)

(c) Let  $w$  be an arbitrary string such that  $w = w^R$ .

Assume that every string  $x$  such that  $|x| < |w|$  and  $x = x^R$  is a palindrome.

There are three cases to consider (mirroring the definition of “palindrome”):

- If  $w = \varepsilon$ , then  $w$  is a palindrome by definition.
- If  $w = a$  for some symbol  $a \in \Sigma$ , then  $w$  is a palindrome by definition.
- Otherwise, we have  $w = ax$  for some symbol  $a$  and some non-empty string  $x$ . The definition of reversal implies that  $w^R = (ax)^R = x^R a$ . Because  $x$  is non-empty, its reversal  $x^R$  is also non-empty. Thus,  $x^R = by$  for some symbol  $b$  and some string  $y$ . It follows that  $w^R = bya$ , and therefore  $w = (w^R)^R = (bya)^R = ay^R b$ .

[At this point, we need to prove that  $a = b$  and that  $y$  is a palindrome.]

Our assumption that  $w = w^R$  implies that  $bya = ay^R b$ .

The recursive definition of string equality immediately implies  $a = b$ .

Because  $a = b$ , we have  $w = ay^R a$  and  $w^R = aya$ .

The recursive definition of string equality implies  $y^R a = ya$ .

Right cancellation implies that  $y^R = y$ .

The inductive hypothesis now implies that  $y$  is a palindrome.

We conclude that  $w$  is a palindrome by definition.

In all three cases, we conclude that  $w$  is a palindrome. ■

**Rubric:** 4 points: standard induction rubric (scaled).

**Standard induction rubric.** For problems worth 10 points:

- + 1 for explicitly considering an *arbitrary* object.
- + 2 for a valid **strong** induction hypothesis
  - **Deadly Sin!** Automatic zero for stating a weak induction hypothesis, unless the rest of the proof is *absolutely perfect*.
- + 2 for explicit exhaustive case analysis
  - No credit here if the case analysis omits an infinite number of objects. (For example: all odd-length palindromes.)
  - -1 if the case analysis omits a finite number of objects. (For example: the empty string.)
  - -1 for making the reader infer the case conditions. Spell them out!
  - No penalty if the cases overlap (for example: even length at least 2, odd length at least 3, and length at most 5.)
- + 1 for cases that do not invoke the inductive hypothesis (“base cases”)
  - No credit here if one or more “base cases” are missing.
- + 2 for correctly applying the **stated** inductive hypothesis
  - No credit here for applying a **different** inductive hypothesis, even if that different inductive hypothesis would be valid.
- + 2 for other details in cases that invoke the inductive hypothesis (“inductive cases”)
  - No credit here if one or more “inductive cases” are missing.

For (sub)problems worth less than 10 points, scale and round to the nearest half-integer.

# CS/ECE 374 A ✦ Spring 2018

## ❖ Homework 1 ❖

Due Tuesday, January 30, 2018 at 8pm

---

**Starting with this homework, groups of up to three people can submit joint solutions.** Each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the Gradescope names and email addresses of each group member. In addition, whoever submits the homework must tell Gradescope who their other group members are.

---

1. For each of the following languages over the alphabet  $\{0, 1\}$ , give a regular expression that describes that language, and *briefly* argue why your expression is correct.
  - (a) All strings except **001**.
  - (b) All strings that end with the suffix **001001**.
  - (c) All strings that contain the substring **001**.
  - (d) All strings that contain the subsequence **001**.
  - (e) All strings that do not contain the substring **001**.
  - (f) All strings that do not contain the subsequence **001**.
  
2. Let  $L$  denote the set of all strings in  $\{0, 1\}^*$  that contain all four strings **00**, **01**, **10**, and **11** as substrings. For example, the strings **110011** and **01001011101001** are in  $L$ , but the strings **00111** and **1010101** are not.

*Formally* describe a DFA with input alphabet  $\Sigma = \{0, 1\}$  that accepts the language  $L$ , by explicitly describing the states  $Q$ , the start state  $s$ , the accept states  $A$ , and the transition function  $\delta$ . Do not attempt to *draw* your DFA; the smallest DFA for this language has 20 states, which is too many for a drawing to be understandable.

Argue that your machine accepts every string in  $L$  and nothing else, by explaining what each state in your DFA *means*. Formal descriptions without English explanations will receive no credit, even if they are correct. (See the standard DFA rubric for more details.)

***This is an exercise in clear communication.*** We are not only asking you to design a *correct* DFA. We are also asking you to clearly, precisely, and convincingly explain your DFA to another human being who understands DFAs but has *not* thought about this particular problem. Excessive formality and excessive brevity will hurt you just as much as imprecision and handwaving.

3. Let  $L$  be the set of all strings in  $\{0, 1\}^*$  that contain *exactly one* occurrence of the substring  $010$ .

- (a) Give a regular expression for  $L$ , and briefly argue why your expression is correct.  
*[Hint: You may find the shorthand notation  $A^+ = AA^*$  useful.]*

- (b) Describe a DFA over the alphabet  $\Sigma = \{0, 1\}$  that accepts the language  $L$ .

Argue that your machine accepts every string in  $L$  and nothing else, by explaining what each state in your DFA *means*. You may either draw the DFA or describe it formally, but the states  $Q$ , the start state  $s$ , the accepting states  $A$ , and the transition function  $\delta$  must be clearly specified. Drawings or formal descriptions without English explanations will receive no credit, even if they are correct.

## Solved problem

4. **C comments** are the set of strings over alphabet  $\Sigma = \{\star, /, A, \diamond, \downarrow\}$  that form a proper comment in the C program language and its descendants, like C++ and Java. Here  $\downarrow$  represents the newline character,  $\diamond$  represents any other whitespace character (like the space and tab characters), and  $A$  represents any non-whitespace character other than  $\star$  or  $/$ .<sup>1</sup> There are two types of C comments:

- Line comments: Strings of the form  $// \dots \downarrow$
- Block comments: Strings of the form  $/* \dots */$

Following the C99 standard, we explicitly disallow **nesting** comments of the same type. A line comment starts with  $//$  and ends at the first  $\downarrow$  after the opening  $//$ . A block comment starts with  $/*$  and ends at the the first  $*/$  completely after the opening  $/*$ ; in particular, every block comment has at least two  $*$ s. For example, each of the following strings is a valid C comment:

$/* **/$        $// \diamond // \diamond \downarrow$        $/* // \diamond \diamond \downarrow **/$        $/* \diamond // \diamond \downarrow \diamond */$

On the other hand, *none* of the following strings is a valid C comment:

$/* /$        $// \diamond // \diamond \downarrow \diamond \downarrow$        $/* \diamond /* \diamond */ \diamond */$

- (a) Describe a regular expression for the set of all C comments.

**Solution:**

$$//((/+ \star + A + \diamond)^* \downarrow + /* ( / + A + \diamond + \downarrow + \star^*(A + \diamond + \downarrow))^* \star^*/)$$

The first subexpression matches all line comments, and the second subexpression matches all block comments. Within a block comment, we can freely use any symbol other than  $\star$ , but any run of  $\star$ s must be followed by a character in  $(A + \diamond + \downarrow)$  or by the closing slash of the comment. ■

---

<sup>1</sup>The actual C commenting syntax is considerably more complex than described here, because of character and string literals.

- The opening  $/*$  or  $//$  of a comment must not be inside a string literal (" $\dots$ ") or a (multi-)character literal (' $\dots$ ').
- The opening double-quote of a string literal must not be inside a character literal (' $\text{'}$ ') or a comment.
- The closing double-quote of a string literal must not be escaped ( $\text{\'}$ )
- The opening single-quote of a character literal must not be inside a string literal (" $\dots$ ' $\dots$ ') or a comment.
- The closing single-quote of a character literal must not be escaped ( $\text{\'}$ )
- A backslash escapes the next symbol if and only if it is not itself escaped ( $\text{\\}$ ) or inside a comment.

For example, the string  $/* \\ \\ * / /* / /* \star */$  is a valid string literal (representing the 5-character string  $/* \star /$ , which is itself a valid block comment!) followed immediately by a valid block comment. **For this homework question, just pretend that the characters ' $\text{'}$ ', ' $\text{''}$ ', and ' $\text{\\}$  don't exist.**

Commenting in C++ is even more complicated, thanks to the addition of raw string literals. Don't ask.

Some C and C++ compilers do support nested block comments, in violation of the language specification. A few other languages, like OCaml, explicitly allow nesting block comments.

**Rubric:** Standard regular expression rubric

- (b) Describe a regular expression for the set of all strings composed entirely of blanks ( $\diamond$ ), newlines ( $\downarrow$ ), and C comments.

**Solution:**

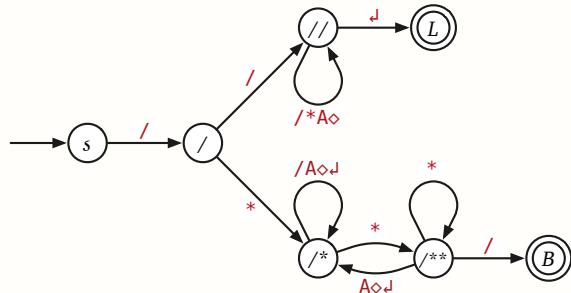
$$(\diamond + \downarrow + //(/ + * + A + \diamond)^* \downarrow + /*(/ + A + \diamond + \downarrow + ***(A + \diamond + \downarrow))^* ***/)^*$$

This regular expression has the form  $((\text{whitespace}) + (\text{comment}))^*$ , where  $\langle \text{whitespace} \rangle$  is the regular expression  $\diamond + \downarrow$  and  $\langle \text{comment} \rangle$  is the regular expression from part (a). ■

**Rubric:** Standard regular expression rubric

- (c) Describe a DFA that accepts the set of all C comments.

**Solution:** The following eight-state DFA recognizes the language of C comments. All missing transitions lead to a hidden reject state.



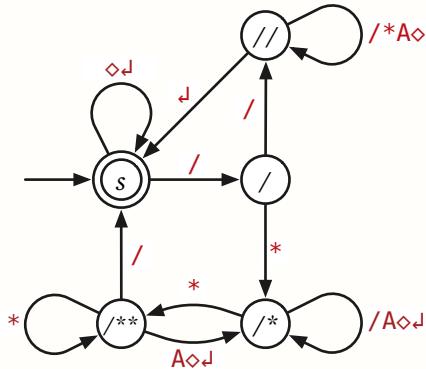
The states are labeled mnemonically as follows:

- $s$  — We have not read anything.
- $/$  — We just read the initial  $/$ .
- $//$  — We are reading a line comment.
- $L$  — We have just read a complete line comment.
- $/*$  — We are reading a block comment, and we did not just read a  $*$  after the opening  $/*$ .
- $/***$  — We are reading a block comment, and we just read a  $*$  after the opening  $/*$ .
- $B$  — We have just read a complete block comment.

**Rubric:** Standard DFA design rubric

- (d) Describe a DFA that accepts the set of all strings composed entirely of blanks ( $\diamond$ ), newlines ( $\downarrow$ ), and C comments.

**Solution:** By merging the accepting states of the previous DFA with the start state and adding white-space transitions at the start state, we obtain the following six-state DFA. Again, all missing transitions lead to a hidden reject state.



The states are labeled mnemonically as follows:

- $s$  — We are between comments.
- $/$  — We just read the initial  $/$  of a comment.
- $//$  — We are reading a line comment.
- $/*$  — We are reading a block comment, and we did not just read a  $*$  after the opening  $/*$ .
- $/**$  — We are reading a block comment, and we just read a  $*$  after the opening  $/*$ .

■

**Rubric:** Standard DFA design rubric

**Standard regular expression rubric.** For problems worth 10 points:

- 2 points for a syntactically correct regular expression.
- **Homework only:** 4 points for a *brief* English explanation of your regular expression. This is how you argue that your regular expression is correct.
  - **Deadly Sin (“Declare your variables.”): No credit for the problem if the English explanation is missing, even if the regular expression is correct.**
  - For longer expressions, you should explain each of the major components of your expression, and separately explain how those components fit together.
  - We do not want a *transcription*; don’t just translate the regular-expression notation into English.
- 4 points for correctness. (8 points on exams, with all penalties doubled)
  - –1 for a single mistake: one typo, excluding exactly one string in the target language, or including exactly one string not in the target language.
  - –2 for incorrectly including/excluding more than one but a finite number of strings.
  - –4 for incorrectly including/excluding an infinite number of strings.
- Regular expressions that are longer than necessary may be penalized. Regular expressions that are *significantly* longer than necessary may get no credit at all.

**Standard DFA design rubric.** For problems worth 10 points:

- 2 points for an unambiguous description of a DFA, including the states set  $Q$ , the start state  $s$ , the accepting states  $A$ , and the transition function  $\delta$ .
  - **For drawings:** Use an arrow from nowhere to indicate  $s$ , and doubled circles to indicate accepting states  $A$ . If  $A = \emptyset$ , say so explicitly. If your drawing omits a reject state, say so explicitly. **Draw neatly!** If we can’t read your solution, we can’t give you credit for it.,
  - **For text descriptions:** You can describe the transition function either using a 2d array, using mathematical notation, or using an algorithm.
  - **For product constructions:** You must give a complete description of the states and transition functions of the DFAs you are combining (as either drawings or text), together with the accepting states of the product DFA.
- **Homework only:** 4 points for *briefly* and correctly explaining the purpose of each state *in English*. This is how you justify that your DFA is correct.
  - **Deadly Sin (“Declare your variables.”): No credit for the problem if the English description is missing, even if the DFA is correct.**
  - For product constructions, explaining the states in the factor DFAs is sufficient.
- 4 points for correctness. (8 points on exams, with all penalties doubled)
  - –1 for a single mistake: a single misdirected transition, a single missing or extra accept state, rejecting exactly one string that should be accepted, or accepting exactly one string that should be accepted.
  - –2 for incorrectly accepting/rejecting more than one but a finite number of strings.
  - –4 for incorrectly accepting/rejecting an infinite number of strings.
- DFA drawings with too many states may be penalized. DFA drawings with *significantly* too many states may get no credit at all.
- Half credit for describing an NFA when the problem asks for a DFA.

# CS/ECE 374 A ✦ Spring 2018

## ❖ Homework 2 ❖

Due Tuesday, February 6, 2018 at 8pm

---

1. Prove that the following languages are *not* regular.

- (a)  $\{\textbf{0}^a \textbf{1} \textbf{0}^b \textbf{1} \textbf{0}^c \mid a + b = c\}$
- (b)  $\{w \in (\textbf{0} + \textbf{1})^* \mid \#(\textbf{0}, w) \leq 2 \cdot \#(\textbf{1}, w)\}$
- (c)  $\{\textbf{0}^m \textbf{1}^n \mid m + n > 0 \text{ and } \gcd(m, n) = 1\}$

Here  $\gcd(m, n)$  denotes the *greatest common divisor* of  $m$  and  $n$ : the largest integer  $d$  such that both  $m/d$  and  $n/d$  are integers. In particular,  $\gcd(1, n) = 1$  and  $\gcd(0, n) = n$  for every positive integer  $n$ .

2. For each of the following regular expressions, describe or draw two finite-state machines:

- An NFA that accepts the same language, constructed from the given regular expression using Thompson's algorithm (described in class and in the notes).
- An equivalent DFA, constructed from your NFA using the incremental subset algorithm (described in class and in the notes). For each state in your DFA, identify the corresponding subset of states in your NFA. Your DFA should have no unreachable states.

- (a)  $(\textbf{0} + \textbf{1})^* \cdot \textbf{0001} \cdot (\textbf{0} + \textbf{1})^*$
- (b)  $(\textbf{1} + \textbf{01} + \textbf{001})^* \textbf{0}^*$

3. For each of the following languages over the alphabet  $\Sigma = \{\textbf{0}, \textbf{1}\}$ , either prove that the language is regular (by constructing an appropriate DFA, NFA, or regular expression) or prove that the language is not regular (by constructing an infinite fooling set). Recall that  $\Sigma^+$  denotes the set of all *nonempty* strings over  $\Sigma$ .

- (a) Strings in which the substrings **00** and **11** do not appear the same number of times. For example, **1100011**  $\notin L$  because both substrings appear twice, but **01000011**  $\in L$ .
- (b) Strings in which the substrings **01** and **10** do not appear the same number of times. For example, **1100011**  $\notin L$  because both substrings appear twice, but **01000011**  $\in L$ .
- (c)  $\{wxw \mid w, x \in \Sigma^*\}$
- (d)  $\{wxw \mid w, x \in \Sigma^+\}$

[Hint: Exactly two of these languages are regular. Strings can be empty.]

## Solved problem

4. For each of the following languages, either prove that the language is regular (by constructing an appropriate DFA, NFA, or regular expression) or prove that the language is not regular (by constructing an infinite fooling set).

Recall that a *palindrome* is a string that equals its own reversal:  $w = w^R$ . Every string of length 0 or 1 is a palindrome.

- (a) Strings in  $(0 + 1)^*$  in which no prefix of length at least 2 is a palindrome.

**Solution: Regular:**  $\epsilon + 01^* + 10^*$ . Call this language  $L_a$ .

Let  $w$  be an arbitrary non-empty string in  $(0 + 1)^*$ . Without loss of generality, assume  $w = 0x$  for some string  $x$ . There are two cases to consider.

- If  $x$  contains a 0, then we can write  $w = 01^n 0y$  for some integer  $n$  and some string  $y$ ; but this is impossible, because the prefix  $01^n 0$  is a palindrome of length at least 2.
- Otherwise,  $x = 1^n$  for some integer  $n$ . Every prefix of  $w$  has the form  $01^m$  for some integer  $m \leq n$ . Any palindrome that starts with 0 must end with 0, so the only palindrome prefixes of  $w$  are  $\epsilon$  and 0, both of which have length less than 2.

We conclude that  $0x \in L_a$  if and only if  $x \in 1^*$ . A similar argument implies that  $1x \in L_a$  if and only if  $x \in 0^*$ . Finally, trivially,  $\epsilon \in L_a$ . ■

**Rubric:** 2½ points = ½ for “regular” + 1 for regular expression + 1 for justification. This is more detail than necessary for full credit.

- (b) Strings in  $(0 + 1 + 2)^*$  in which no prefix of length at least 2 is a palindrome.

**Solution: Not regular.** Call this language  $L_b$ .

I claim that the infinite language  $F = (012)^+$  is a fooling set for  $L_b$ .

Let  $x$  and  $y$  be arbitrary distinct strings in  $F$ .

Then  $x = (012)^i$  and  $y = (012)^j$  for some positive integers  $i \neq j$ .

Without loss of generality, assume  $i < j$ .

Let  $z$  be the suffix  $(210)^i$ .

- $xz = (012)^i(210)^i$  is a palindrome of length  $6i \geq 2$ , so  $xz \notin L_b$ .
- $yz = (012)^j(210)^i$  has no palindrome prefixes except  $\epsilon$  and 0, because  $i < j$ , so  $yz \in L_b$ .

We conclude that  $F$  is a fooling set for  $L_b$ , as claimed.

Because  $F$  is infinite,  $L_b$  cannot be regular. ■

**Rubric:** 2½ points = ½ for “not regular” + 2 for fooling set proof (standard rubric, scaled).

- (c) Strings in  $(0 + 1)^*$  in which no prefix of length at least 3 is a palindrome.

**Solution: Not regular.** Call this language  $L_c$ .

I claim that the infinite language  $F = (\underline{001101})^+$  is a fooling set for  $L_c$ .

Let  $x$  and  $y$  be arbitrary distinct strings in  $F$ .

Then  $x = (\underline{001101})^i$  and  $y = (\underline{001101})^j$  for some positive integers  $i \neq j$ .

Without loss of generality, assume  $i < j$ .

Let  $z$  be the suffix  $(\underline{101100})^i$ .

- $xz = (\underline{001101})^i(\underline{101100})^i$  is a palindrome of length  $12i \geq 2$ , so  $xz \notin L_b$ .
- $yz = (\underline{001101})^j(\underline{101100})^i$  has no palindrome prefixes except  $\epsilon$  and  $0$ , because  $i < j$ , so  $yz \in L_b$ .

We conclude that  $F$  is a fooling set for  $L_c$ , as claimed.

Because  $F$  is infinite,  $L_c$  cannot be regular. ■

**Rubric:** 2½ points = ½ for “not regular” + 2 for fooling set proof (standard rubric, scaled).

- (d) Strings in  $(0 + 1)^*$  in which no *substring* of length at least 3 is a palindrome.

**Solution: Regular.** Call this language  $L_d$ .

Every palindrome of length at least 3 contains a palindrome substring of length 3 or 4. Thus, the complement language  $\overline{L_d}$  is described by the regular expression

$$(0 + 1)^*(000 + \underline{010} + \underline{101} + \underline{111} + \underline{0110} + \underline{1001})(0 + 1)^*$$

Thus,  $\overline{L_d}$  is regular, so its complement  $L_d$  is also regular. ■

**Solution: Regular.** Call this language  $L_d$ .

In fact,  $L_d$  is *finite*! Appending either  $0$  or  $1$  to any of the underlined strings creates a palindrome suffix of length 3 or 4.

$$\epsilon + 0 + 1 + \underline{00} + \underline{01} + \underline{10} + \underline{11} + \underline{001} + \underline{011} + \underline{100} + \underline{110} + \underline{0011} + \underline{1100}$$

**Rubric:** 2½ points = ½ for “regular” + 2 for proof:

- 1 for expression for  $\overline{L_d}$  + 1 for applying closure
- 1 for regular expression + 1 for justification

**Standard fooling set rubric.** For problems worth 5 points:

- 2 points for the fooling set:
  - + 1 for explicitly describing the proposed fooling set  $F$ .
  - + 1 if the proposed set  $F$  is actually a fooling set for the target language.
  - No credit for the proof if the proposed set is not a fooling set.
  - No credit for the *problem* if the proposed set is finite.
- 3 points for the proof:
  - The proof must correctly consider *arbitrary* strings  $x, y \in F$ .
    - No credit for the proof unless both  $x$  and  $y$  are *always* in  $F$ .
    - No credit for the proof unless  $x$  and  $y$  can be *any* strings in  $F$ .
  - + 1 for correctly describing a suffix  $z$  that distinguishes  $x$  and  $y$ .
  - + 1 for proving either  $xz \in L$  or  $yz \in L$ .
  - + 1 for proving either  $yz \notin L$  or  $xz \notin L$ , respectively.

As usual, scale partial credit (rounded to nearest  $\frac{1}{2}$ ) for problems worth fewer points.

# CS/ECE 374 A ✦ Spring 2018

## ∞ Homework 3 ∞

Due Tuesday, February 13, 2018 at 8pm

---

1. Describe context-free grammars for the following languages over the alphabet  $\Sigma = \{\texttt{0}, \texttt{1}\}$ . For each non-terminal in your grammars, describe in English the language generated by that non-terminal.
  - (a)  $\{\texttt{0}^a \texttt{1}^b \texttt{0}^c \mid a + b = c\}$
  - (b)  $\{w \in (\texttt{0} + \texttt{1})^* \mid \#(\texttt{0}, w) \leq 2 \cdot \#(\texttt{1}, w)\}$
  - (c) Strings in which the substrings **00** and **11** appear the same number of times. For example, **1100011**  $\in L$  because both substrings appear twice, but **01000011**  $\notin L$ . [Hint: This is the complement of the language you considered in HW2.]
2. Let  $inc: \{\texttt{0}, \texttt{1}\}^* \rightarrow \{\texttt{0}, \texttt{1}\}^*$  denote the *increment* function, which transforms the binary representation of an arbitrary integer  $n$  into the binary representation of  $n + 1$ , truncated to the same number of bits. For example:

$$inc(\texttt{0010}) = \texttt{0011} \quad inc(\texttt{0111}) = \texttt{1000} \quad inc(\texttt{1111}) = \texttt{0000} \quad inc(\epsilon) = \epsilon$$

Let  $L \subseteq \{\texttt{0}, \texttt{1}\}^*$  be an arbitrary regular language. Prove that  $inc(L) = \{inc(w) \mid w \in L\}$  is also regular.

3. A *shuffle* of two strings  $x$  and  $y$  is any string obtained by interleaving the symbols in  $x$  and  $y$ , but keeping them in the same order. For example, the following strings are shuffles of **HOGWARTS** and **BRAKEBILLS**:

**HOGWARTSBRAKEBILLS    HOGBRAKEWARTSBILLS    BHROAGKWEABRITLSLS**

More formally, a string  $z$  is a shuffle of strings  $x$  and  $y$  if and only if (at least) one of the following conditions holds:

- $x = \epsilon$  and  $z = y$
- $y = \epsilon$  and  $z = x$
- $x = ax'$  and  $z = az'$  where  $z'$  is a shuffle of  $x'$  and  $y$
- $y = ay'$  and  $z = az'$  where  $z'$  is a shuffle of  $x$  and  $y'$

For any two languages  $L$  and  $L'$  over the alphabet  $\{\texttt{0}, \texttt{1}\}$ , define

$$shuffles(L, L') = \{z \in \{\texttt{0}, \texttt{1}\}^* \mid z \text{ is a shuffle of some } x \in L \text{ and } y \in L'\}$$

Prove that if  $L$  and  $L'$  are regular languages, then  $shuffles(L, L')$  is also a regular language.

## Solved problem

4. (a) Fix an arbitrary regular language  $L$ . Prove that the language  $\text{half}(L) := \{w \mid ww \in L\}$  is also regular.

**Solution:** Let  $M = (\Sigma, Q, s, A, \delta)$  be an arbitrary DFA that accepts  $L$ . We define a new NFA  $M' = (\Sigma, Q', s', A', \delta')$  with  $\epsilon$ -transitions that accepts  $\text{half}(L)$ , as follows:

$$Q' = (Q \times Q \times Q) \cup \{s'\}$$

$s'$  is an explicit state in  $Q'$

$$A' = \{(h, h, q) \mid h \in Q \text{ and } q \in A\}$$

$$\delta'(s', \epsilon) = \{(s, h, h) \mid h \in Q\}$$

$$\delta'(s', a) = \emptyset$$

$$\delta'((p, h, q), \epsilon) = \emptyset$$

$$\delta'((p, h, q), a) = \{(\delta(p, a), h, \delta(q, a))\}$$

$M'$  reads its input string  $w$  and simulates  $M$  reading the input string  $ww$ . Specifically,  $M'$  simultaneously simulates two copies of  $M$ , one reading the left half of  $ww$  starting at the usual start state  $s$ , and the other reading the right half of  $ww$  starting at some intermediate state  $h$ .

- The new start state  $s'$  non-deterministically guesses the “halfway” state  $h = \delta^*(s, w)$  without reading any input; this is the only non-determinism in  $M'$ .
- State  $(p, h, q)$  means the following:
  - The left copy of  $M$  (which started at state  $s$ ) is now in state  $p$ .
  - The initial guess for the halfway state is  $h$ .
  - The right copy of  $M$  (which started at state  $h$ ) is now in state  $q$ .
- $M'$  accepts if and only if the left copy of  $M$  ends at state  $h$  (so the initial non-deterministic guess  $h = \delta^*(s, w)$  was correct) and the right copy of  $M$  ends in an accepting state.

■

**Solution (smartass):** A complete solution is given in the lecture notes. ■

**Rubric:** 5 points: standard langage transformation rubric (scaled). Yes, the smartass solution would be worth full credit.

- (b) Describe a regular language  $L$  such that the language  $\text{double}(L) := \{ww \mid w \in L\}$  is not regular. Prove your answer is correct.

**Solution:** Consider the regular language  $L = 0^*1$ .

Expanding the regular expression lets us rewrite  $L = \{0^n1 \mid n \geq 0\}$ . It follows that  $\text{double}(L) = \{0^n10^n1 \mid n \geq 0\}$ . I claim that this language is not regular.

Let  $x$  and  $y$  be arbitrary distinct strings in  $L$ .

Then  $x = 0^i1$  and  $y = 0^j1$  for some integers  $i \neq j$ .

Then  $x$  is a distinguishing suffix of these two strings, because

- $xx \in \text{double}(L)$  by definition, but
- $yx = 0^i10^j1 \notin \text{double}(L)$  because  $i \neq j$ .

We conclude that  $L$  is a fooling set for  $\text{double}(L)$ .

Because  $L$  is infinite,  $\text{double}(L)$  cannot be regular. ■

**Solution:** Consider the regular language  $L = \Sigma^* = (0 + 1)^*$ .

I claim that the language  $\text{double}(\Sigma^*) = \{ww \mid w \in \Sigma^*\}$  is not regular.

Let  $F$  be the infinite language  $01^*0$ .

Let  $x$  and  $y$  be arbitrary distinct strings in  $F$ .

Then  $x = 01^i0$  and  $y = 01^j0$  for some integers  $i \neq j$ .

The string  $z = 1^i$  is a distinguishing suffix of these two strings, because

- $xz = 01^i01^i = ww$  where  $w = 01^i$ , so  $xz \in \text{double}(\Sigma^*)$ , but
- $yx = 01^j01^i \notin \text{double}(\Sigma^*)$  because  $i \neq j$ .

We conclude that  $F$  is a fooling set for  $\text{double}(\Sigma^*)$ .

Because  $F$  is infinite,  $\text{double}(\Sigma^*)$  cannot be regular. ■

**Rubric:** 5 points:

- 2 points for describing a regular language  $L$  such that  $\text{double}(L)$  is not regular.
- 1 point for describing an infinite fooling set for  $\text{double}(L)$ :
  - +  $\frac{1}{2}$  for explicitly describing the proposed fooling set  $F$ .
  - +  $\frac{1}{2}$  if the proposed set  $F$  is actually a fooling set.
  - No credit for the proof if the proposed set is not a fooling set.
  - No credit for the *problem* if the proposed set is finite.
- 2 points for the proof:
  - +  $\frac{1}{2}$  for correctly considering *arbitrary* strings  $x$  and  $y$ 
    - No credit for the proof unless both  $x$  and  $y$  are *always* in  $F$ .
    - No credit for the proof unless both  $x$  and  $y$  can be *any* string in  $F$ .
  - +  $\frac{1}{2}$  for correctly stating a suffix  $z$  that distinguishes  $x$  and  $y$ .
  - +  $\frac{1}{2}$  for proving either  $xz \in L$  or  $yz \in L$ .
  - +  $\frac{1}{2}$  for proving either  $yz \notin L$  or  $xz \notin L$ , respectively.

These are not the only correct solutions. These are not the only fooling sets for these languages.

**Standard language transformation rubric.** For problems worth 10 points:

- + 2 for a formal, complete, and unambiguous description of the output automaton, including the states, the start state, the accepting states, and the transition function, as functions of an *arbitrary* input DFA. The description must state whether the output automaton is a DFA, an NFA without  $\epsilon$ -transitions, or an NFA with  $\epsilon$ -transitions.
  - No points for the rest of the problem if this is missing.
- + 2 for a *brief* English explanation of the output automaton. We explicitly do *not* want a formal proof of correctness, or an English *transcription*, but a few sentences explaining how your machine works and justifying its correctness. What is the overall idea? What do the states represent? What is the transition function doing? Why these accepting states?
  - **Deadly Sin:** No points for the rest of the problem if this is missing.
- + 6 for correctness
  - + 3 for accepting *all* strings in the target language
  - + 3 for accepting *only* strings in the target language
  - 1 for a single mistake in the formal description (for example a typo)
    - Double-check correctness when the input language is  $\emptyset$ , or  $\{\epsilon\}$ , or  $\theta^*$ , or  $\Sigma^*$ .

# CS/ECE 374 A ✦ Spring 2018

## ❖ Homework 4 ❖

Due Tuesday, February 27, 2018 at 8pm

---

- At the end of the second act of the action blockbuster *Fast and Impossible XIII½: Guardians of Expendable Justice Reloaded*, the villainous Dr. Metaphor hypnotizes the entire Hero League/Force/Squad, arranges them in a long line at the edge of a cliff, and instructs each hero to shoot the closest taller heroes to their left and right, at a prearranged signal.

Suppose we are given the heights of all  $n$  heroes, in clockwise order around the circle, in an array  $Ht[1..n]$ . (To avoid salary arguments, the producers insisted that no two heroes have the same height.) Then we can compute the Left and Right targets of each hero in  $O(n^2)$  time using the following algorithm.

```
WHO TARGETS WHOM( $Ht[1..n]$ ):  
    for  $j \leftarrow 1$  to  $n$   
        <<Find the left target  $L[j]$  for hero  $j$ >>  
         $L[j] \leftarrow \text{NONE}$   
        for  $i \leftarrow 1$  to  $j - 1$   
            if  $Ht[i] > Ht[j]$   
                 $L[j] \leftarrow i$   
        <<Find the right target  $R[j]$  for hero  $j$ >>  
         $R[j] \leftarrow \text{NONE}$   
        for  $k \leftarrow n$  down to  $j + 1$   
            if  $Ht[k] > Ht[j]$   
                 $R[j] \leftarrow k$   
    return  $L[1..n], R[1..n]$ 
```

- Describe a divide-and-conquer algorithm that computes the output of WHO TARGETS-WHOM in  $O(n \log n)$  time.
- Prove that at least  $\lfloor n/2 \rfloor$  of the  $n$  heroes are targets. That is, prove that the output arrays  $R[0..n-1]$  and  $L[0..n-1]$  contain at least  $\lfloor n/2 \rfloor$  distinct values (other than `NONE`).
- Alas, Dr. Metaphor's diabolical plan is successful. At the prearranged signal, all the heroes simultaneously shoot their targets, and all targets fall over the cliff, apparently dead. Metaphor repeats his dastardly experiment over and over; after each massacre, he forces the remaining heroes to choose new targets, following the same algorithm, and then shoot their targets at the next signal. Eventually, only the shortest member of the Hero Crew/Alliance/Posse is left alive.<sup>1</sup>

Describe an algorithm that computes the number of rounds before Dr. Metaphor's deadly process finally ends. For full credit, your algorithm should run in  $O(n)$  time.

---

<sup>1</sup>In the thrilling final act, Retcon the Squirrel, the last surviving member of the Hero Team/Group/Society, saves everyone by traveling back in time and retroactively replacing the other  $n - 1$  heroes with lifelike balloon sculptures.

2. Describe and analyze a recursive algorithm to reconstruct an arbitrary binary tree, given its preorder and inorder node sequences as input.

The input to your algorithm is a pair of arrays  $Pre[1..n]$  and  $In[1..n]$ , each containing a permutation of the same set of  $n$  distinct symbols. Your algorithm should return an  $n$ -node binary tree whose nodes are labeled with those  $n$  symbols (or an error code if no binary tree is consistent with the input arrays). You solved an instance of this problem in Homework 0.

3. Suppose we are given a set  $S$  of  $n$  items, each with a *value* and a *weight*. For any element  $x \in S$ , we define two subsets:

- $S_{<x}$  is the set of all elements of  $S$  whose value is smaller than the value of  $x$ .
- $S_{>x}$  is the set of all elements of  $S$  whose value is larger than the value of  $x$ .

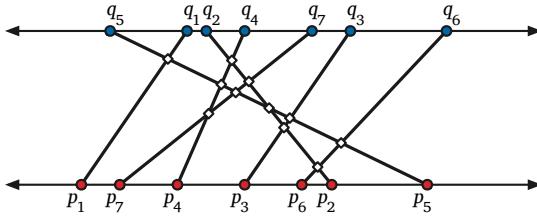
For any subset  $R \subseteq S$ , let  $w(R)$  denote the sum of the weights of elements in  $R$ . The **weighted median** of  $R$  is any element  $x$  such that  $w(S_{<x}) \leq w(S)/2$  and  $w(S_{>x}) \leq w(S)/2$ .

Describe and analyze an algorithm to compute the weighted median of a given weighted set in  $O(n)$  time. Your input consists of two unsorted arrays  $S[1..n]$  and  $W[1..n]$ , where for each index  $i$ , the  $i$ th element has value  $S[i]$  and weight  $W[i]$ . You may assume that all values are distinct and all weights are positive.

[Hint: Use or modify the linear-time selection algorithm described in class on Thursday.]

## Solved problem

4. Suppose we are given two sets of  $n$  points, one set  $\{p_1, p_2, \dots, p_n\}$  on the line  $y = 0$  and the other set  $\{q_1, q_2, \dots, q_n\}$  on the line  $y = 1$ . Consider the  $n$  line segments connecting each point  $p_i$  to the corresponding point  $q_i$ . Describe and analyze a divide-and-conquer algorithm to determine how many pairs of these line segments intersect, in  $O(n \log n)$  time. See the example below.



Seven segments with endpoints on parallel lines, with 11 intersecting pairs.

Your input consists of two arrays  $P[1..n]$  and  $Q[1..n]$  of  $x$ -coordinates; you may assume that all  $2n$  of these numbers are distinct. No proof of correctness is necessary, but you should justify the running time.

**Solution:** We begin by sorting the array  $P[1..n]$  and permuting the array  $Q[1..n]$  to maintain correspondence between endpoints, in  $O(n \log n)$  time. Then for any indices  $i < j$ , segments  $i$  and  $j$  intersect if and only if  $Q[i] > Q[j]$ . Thus, our goal is to compute the number of pairs of indices  $i < j$  such that  $Q[i] > Q[j]$ . Such a pair is called an *inversion*.

We count the number of inversions in  $Q$  using the following extension of mergesort; as a side effect, this algorithm also sorts  $Q$ . If  $n < 100$ , we use brute force in  $O(1)$  time. Otherwise:

- Color the elements in the Left half  $Q[1.. \lfloor n/2 \rfloor]$  bLue.
- Color the elements in the Right half  $Q[\lfloor n/2 \rfloor + 1..n]$  Red.
- Recursively count inversions in (and sort) the blue subarray  $Q[1.. \lfloor n/2 \rfloor]$ .
- Recursively count inversions in (and sort) the red subarray  $Q[\lfloor n/2 \rfloor + 1..n]$ .
- Count red/blue inversions as follows:
  - MERGE the sorted subarrays  $Q[1..n/2]$  and  $Q[n/2+1..n]$ , maintaining the element colors.
  - For each blue element  $Q[i]$  of the now-sorted array  $Q[1..n]$ , count the number of smaller red elements  $Q[j]$ .

The last substep can be performed in  $O(n)$  time using a simple for-loop:

```
COUNTREDBLUE( $A[1..n]$ ):
  count  $\leftarrow 0$ 
  total  $\leftarrow 0$ 
  for  $i \leftarrow 1$  to  $n$ 
    if  $A[i]$  is red
      count  $\leftarrow$  count + 1
    else
      total  $\leftarrow$  total + count
  return total
```

MERGE and COUNTREDBLUE each run in  $O(n)$  time. Thus, the running time of our inversion-counting algorithm obeys the mergesort recurrence  $T(n) = 2T(n/2) + O(n)$ . (We can safely ignore the floors and ceilings in the recursive arguments.) We conclude that the overall running time of our algorithm is  $O(n \log n)$ , as required.

**Rubric:** This is enough for full credit.

In fact, we can execute the third merge-and-count step directly by modifying the MERGE algorithm, without any need for “colors”. Here changes to the standard MERGE algorithm are indicated in red.

```
MERGEANDCOUNT( $A[1..n], m$ ):
   $i \leftarrow 1; j \leftarrow m + 1; count \leftarrow 0; total \leftarrow 0$ 
  for  $k \leftarrow 1$  to  $n$ 
    if  $j > n$ 
       $B[k] \leftarrow A[i]; i \leftarrow i + 1; total \leftarrow total + count$ 
    else if  $i > m$ 
       $B[k] \leftarrow A[j]; j \leftarrow j + 1; count \leftarrow count + 1$ 
    else if  $A[i] < A[j]$ 
       $B[k] \leftarrow A[i]; i \leftarrow i + 1; total \leftarrow total + count$ 
    else
       $B[k] \leftarrow A[j]; j \leftarrow j + 1; count \leftarrow count + 1$ 
  for  $k \leftarrow 1$  to  $n$ 
     $A[k] \leftarrow B[k]$ 
  return total
```

We can further optimize MERGEANDCOUNT by observing that  $count$  is always equal to  $j - m - 1$ , so we don’t need an additional variable. (Proof: Initially,  $j = m + 1$  and  $count = 0$ , and we always increment  $j$  and  $count$  together.)

```

MERGEANDCOUNT2( $A[1..n]$ ,  $m$ ):
     $i \leftarrow 1$ ;  $j \leftarrow m + 1$ ;  $total \leftarrow 0$ 
    for  $k \leftarrow 1$  to  $n$ 
        if  $j > n$ 
             $B[k] \leftarrow A[i]$ ;  $i \leftarrow i + 1$ ;  $total \leftarrow total + j - m - 1$ 
        else if  $i > m$ 
             $B[k] \leftarrow A[j]$ ;  $j \leftarrow j + 1$ 
        else if  $A[i] < A[j]$ 
             $B[k] \leftarrow A[i]$ ;  $i \leftarrow i + 1$ ;  $total \leftarrow total + j - m - 1$ 
        else
             $B[k] \leftarrow A[j]$ ;  $j \leftarrow j + 1$ 
    for  $k \leftarrow 1$  to  $n$ 
         $A[k] \leftarrow B[k]$ 
    return  $total$ 

```

MERGEANDCOUNT2 still runs in  $O(n)$  time, so the overall running time is still  $O(n \log n)$ , as required. ■

**Rubric:** 10 points = 2 for base case + 3 for divide (split and recurse) + 3 for conquer (merge and count) + 2 for time analysis. Max 3 points for a correct  $O(n^2)$ -time algorithm. This is neither the only way to correctly describe this algorithm nor the only correct  $O(n \log n)$ -time algorithm. No proof of correctness is required.

Notice that each boxed algorithm is preceded by an English description of the task that algorithm performs. **Omitting these descriptions is a Deadly Sin.**

## ∞ Homework 5 ∞

Due Tuesday, March 6, 2018 at 8pm

---

1. It's almost time to show off your flippin' sweet dancing skills! Tomorrow is the big dance contest you've been training for your entire life, except for that summer you spent with your uncle in Alaska hunting wolverines. You've obtained an advance copy of the list of  $n$  songs that the judges will play during the contest, in chronological order.

You know all the songs, all the judges, and your own dancing ability extremely well. For each integer  $k$ , you know that if you dance to the  $k$ th song on the schedule, you will be awarded exactly  $Score[k]$  points, but then you will be physically unable to dance for the next  $Wait[k]$  songs (that is, you cannot dance to songs  $k + 1$  through  $k + Wait[k]$ ). The dancer with the highest total score at the end of the night wins the contest, so you want your total score to be as high as possible.

Describe and analyze an efficient algorithm to compute the maximum total score you can achieve. The input to your sweet algorithm is the pair of arrays  $Score[1..n]$  and  $Wait[1..n]$ .

2. Suppose you are given a NFA  $M = (\{\text{0}, \text{1}\}, Q, s, A, \delta)$  and a binary string  $w \in \{\text{0}, \text{1}\}^*$ . Describe and analyze an efficient algorithm to determine whether  $M$  accepts  $w$ . Concretely, the input NFA  $M$  is represented as follows:

- $Q = \{1, 2, \dots, k\}$  for some integer  $k$ .
- The start state  $s$  is state 1.
- Accepting states are indicated by a boolean array  $A[1..k]$ , where  $A[q] = \text{TRUE}$  if and only if  $q \in A$
- The transition function  $\delta$  is represented by a boolean array  $inDelta[1..k, 0..1, 1..k]$ , where  $inDelta[p, a, q] = \text{TRUE}$  if and only if  $q \in \delta(p, a)$ .

Finally, the input string is given as an array  $w[1..n]$ . Your algorithm should return **TRUE** if  $M$  accepts  $w$ , and **FALSE** if  $M$  does not accept  $w$ . Report the running time of your algorithm as a function of  $k$  (the number of states in  $M$ ) and  $n$  (the length of  $w$ ). [Hint: *Do not convert  $M$  to a DFA!!*]

3. Recall that a *palindrome* is any string that is exactly the same as its reversal, like the empty string, or **I**, or **DEED**, or **RACECAR**, or **AMANAPLANACATACANALPANAMA**.

Any string can be decomposed into a sequence of palindromes. For example, the string **BUBBASEESABANANA** (“Bubba sees a banana.”) can be broken into non-empty palindromes in the following ways (and 65 others):

BUB • BASESAB • ANANA  
 B • U • BB • ASEESA • B • ANANA  
 BUB • B • A • SEES • ABA • N • ANA  
 B • U • BB • A • S • EE • S • A • B • A • NAN • A  
 B • U • B • B • A • S • E • E • S • A • B • A • N • A • N • A

- (a) Describe and analyze an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example:
- Given the string **PALINDROME**, your algorithm should return the integer 10.
  - Given the string **BUBBASEESABANANA**, your algorithm should return the integer 3.
  - Given the string **RACECAR**, your algorithm should return the integer 1.
- (b) A *metapalindrome* is a decomposition of a string into a sequence of non-empty palindromes, such that the sequence of palindrome lengths is itself a palindrome. For example, the decomposition

BUB • B • ALA • SEES • ABA • N • ANA

is a metapalindrome for the string **BUBBALASEESABANANA**, with the palindromic length sequence (3, 1, 3, 4, 3, 1, 3). Describe and analyze an efficient algorithm to find the length of the shortest metapalindrome for a given string. For example:

- Given the string **BUBBALASEESABANANA**, your algorithm should return the integer 7.
- Given the string **PALINDROME**, your algorithm should return the integer 10.
- Given the string **DEPOPED**, your algorithm should return the integer 1.

## Solved Problem

4. A *shuffle* of two strings  $X$  and  $Y$  is formed by interspersing the characters into a new string, keeping the characters of  $X$  and  $Y$  in the same order. For example, the string **BANANAANANAS** is a shuffle of the strings **BANANA** and **ANANAS** in several different ways.

**BANANAANANAS**      **BANANAANANAS**      **BANANAANANAS**

Similarly, the strings **PRODGYRNAMAMMIINC** and **DYPRONGARMAMMAGIC** are both shuffles of **DYNAMIC** and **PROGRAMMING**:

**PRODGYRNAMAMMIINC**      **DYPRONGARMAMMAGIC**

Given three strings  $A[1..m]$ ,  $B[1..n]$ , and  $C[1..m+n]$ , describe and analyze an algorithm to determine whether  $C$  is a shuffle of  $A$  and  $B$ .

**Solution:** We define a boolean function  $Shuf(i, j)$ , which is TRUE if and only if the prefix  $C[1..i+j]$  is a shuffle of the prefixes  $A[1..i]$  and  $B[1..j]$ . This function satisfies the following recurrence:

$$Shuf(i, j) = \begin{cases} \text{TRUE} & \text{if } i = j = 0 \\ Shuf(0, j-1) \wedge (B[j] = C[j]) & \text{if } i = 0 \text{ and } j > 0 \\ Shuf(i-1, 0) \wedge (A[i] = C[i]) & \text{if } i > 0 \text{ and } j = 0 \\ (Shuf(i-1, j) \wedge (A[i] = C[i+j])) \\ \vee (Shuf(i, j-1) \wedge (B[j] = C[i+j])) & \text{if } i > 0 \text{ and } j > 0 \end{cases}$$

We need to compute  $Shuf(m, n)$ .

We can memoize all function values into a two-dimensional array  $Shuf[0..m][0..n]$ . Each array entry  $Shuf[i, j]$  depends only on the entries immediately below and immediately to the right:  $Shuf[i-1, j]$  and  $Shuf[i, j-1]$ . Thus, we can fill the array in standard row-major order. The original recurrence gives us the following pseudocode:

```

SHUFFLE?(A[1..m], B[1..n], C[1..m+n]):
    Shuf[0,0] ← TRUE
    for j ← 1 to n
        Shuf[0,j] ← Shuf[0,j-1] ∧ (B[j] = C[j])
    for i ← 1 to n
        Shuf[i,0] ← Shuf[i-1,0] ∧ (A[i] = B[i])
        for j ← 1 to n
            Shuf[i,j] ← FALSE
            if A[i] = C[i+j]
                Shuf[i,j] ← Shuf[i,j] ∨ Shuf[i-1,j]
            if B[i] = C[i+j]
                Shuf[i,j] ← Shuf[i,j] ∨ Shuf[i,j-1]
    return Shuf[m,n]

```

The algorithm runs in  $O(mn)$  time. ■

**Rubric:** Max 10 points: Standard dynamic programming rubric. No proofs required. Max 7 points for a slower polynomial-time algorithm; scale partial credit accordingly.

**Standard dynamic programming rubric.** For problems worth 10 points:

- 6 points for a correct recurrence, described either using mathematical notation or as pseudocode for a recursive algorithm.
  - + 1 point for a clear English description of the function you are trying to evaluate. (Otherwise, we don't even know what you're *trying* to do.) **Deadly Sin: Automatic zero if the English description is missing.**
  - + 1 point for stating how to call your function to get the final answer.
  - + 1 point for base case(s).  $-\frac{1}{2}$  for one *minor* bug, like a typo or an off-by-one error.
  - + 3 points for recursive case(s).  $-1$  for each *minor* bug, like a typo or an off-by-one error. **No credit for the rest of the problem if the recursive case(s) are incorrect.**
- 4 points for details of the dynamic programming algorithm
  - + 1 point for describing the memoization data structure
  - + 2 points for describing a correct evaluation order; a clear picture is usually sufficient. If you use nested loops, be sure to specify the nesting order.
  - + 1 point for time analysis
- It is *not* necessary to state a space bound.
- For problems that ask for an algorithm that computes an optimal *structure*—such as a subset, partition, subsequence, or tree—an algorithm that computes only the *value* or *cost* of the optimal structure is sufficient for full credit, unless the problem says otherwise.
- Official solutions usually include pseudocode for the final iterative dynamic programming algorithm, **but iterative pseudocode is not required for full credit.** If your solution includes iterative pseudocode, you do not need to separately describe the recurrence, memoization structure, or evaluation order. (But you still need to describe the underlying recursive function in English.)
- Official solutions will provide target time bounds. Algorithms that are faster than this target are worth more points; slower algorithms are worth fewer points, typically by 2 or 3 points (out of 10) for each factor of  $n$ . Partial credit is scaled to the new maximum score, and all points above 10 are recorded as extra credit.

We rarely include these target time bounds in the actual questions, because when we have included them, significantly more students turned in algorithms that meet the target time bound but didn't work (earning 0/10) instead of correct algorithms that are slower than the target time bound (earning 8/10).

# CS/ECE 374 A ✦ Spring 2018

## ❖ Homework 6 ❖

Due Tuesday, March 13, 2018 at 8pm

- 
1. Suppose you are given an array  $A[1..n]$  of positive integers, each of which is colored either **red** or **blue**. An *increasing back-and-forth subsequence* is a sequence of indices  $I[1..\ell]$  with the following properties:

- $1 \leq I[j] \leq n$  for all  $j$ .
- $A[I[j]] < A[I[j+1]]$  for all  $j < \ell$ .
- If  $A[I[j]]$  is **red**, then  $I[j+1] > I[j]$ .
- If  $A[I[j]]$  is **blue**, then  $I[j+1] < I[j]$ .

Less formally, suppose we start with a token on some integer  $A[j]$ , and then repeatedly move the token Left (if it's on a **bLue** square) or Right (if it's on a **Red** square), always moving from a smaller number to a larger number. Then the sequence of token positions is an increasing back-and-forth subsequence.

Describe and analyze an efficient algorithm to compute the length of the longest increasing back-and-forth subsequence of a given array of  $n$  red and blue integers. For example, given the input array

1	1	0	2	5	9	6	6	4	5	8	9	7	7	3	2	3	8	4	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

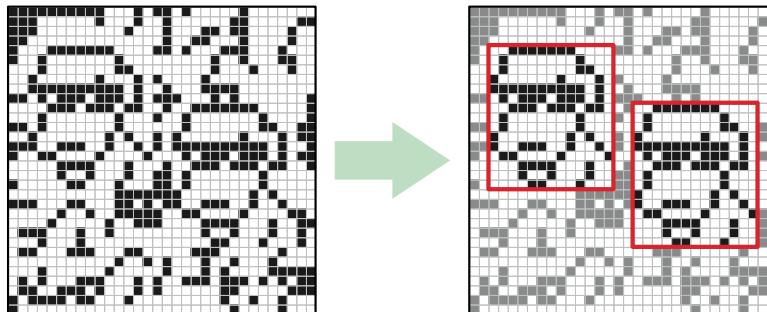
your algorithm should return the integer 9, which is the length of the following increasing back-and-forth subsequence:

0	1	2	3	4	6	7	8	9
20	1	16	17	9	8	13	11	12

(The small numbers are indices into the input array.)

2. Describe and analyze an algorithm that finds the largest rectangular pattern that appears more than once in a given bitmap. Your input is a two-dimensional array  $M[1..n, 1..n]$  of bits; your output is the area of the repeated pattern. (The two copies of the pattern might overlap, but must not actually coincide.)

For example, given the bitmap shown on the left in the figure below, your algorithm should return  $15 \times 13 = 195$ , because the same  $15 \times 13$  doggo appears twice, as shown on the right, and this is the largest such pattern.



3. *AVL trees* were the earliest self-balancing balanced binary search trees, first described in 1962 by Georgy Adelson-Velsky and Evgenii Landis. An AVL tree is a binary search tree where for every node  $v$ , the height of the left subtree of  $v$  and the height of the right subtree of  $v$  differ by at most 1.

Describe and analyze an efficient algorithm to construct an optimal AVL tree for a given set of keys and frequencies. Your input consists of a sorted array  $A[1..n]$  of search keys and an array  $f[1..n]$  of frequency counts, where  $f[i]$  is the number of searches for  $A[i]$ . Your task is to construct an AVL tree for the given keys such that the total cost of all searches is as small as possible. This is exactly the same cost function that we considered in Thursday's class; the only difference is that the output tree must satisfy the AVL balance constraint.

*[Hint: You do **not** need to know or use the insertion and deletion algorithms that keep the AVL tree balanced.]*

## Solved Problems

4. A string  $w$  of parentheses ( and ) and brackets [ and ] is *balanced* if and only if  $w$  is generated by the following context-free grammar:

$$S \rightarrow \varepsilon \mid (S) \mid [S] \mid SS$$

For example, the string  $w = (\textcolor{blue}{[\textcolor{red}{(}\textcolor{blue}{)}\textcolor{red}{]}\textcolor{blue}{(\textcolor{red}{)}\textcolor{blue}{)})\textcolor{red}{[}\textcolor{blue}{(\textcolor{red}{)}\textcolor{blue}{)}\textcolor{red}{]}$  is balanced, because  $w = xy$ , where

$$x = \textcolor{blue}{(\textcolor{red}{[\textcolor{blue}{(}\textcolor{red}{)}\textcolor{blue}{]}\textcolor{red}{]}\textcolor{blue}{(\textcolor{red}{)}\textcolor{blue}{)})} \quad \text{and} \quad y = \textcolor{red}{[}\textcolor{blue}{(\textcolor{red}{)}\textcolor{blue}{)}\textcolor{red}{]}.$$

Describe and analyze an algorithm to compute the length of a longest balanced subsequence of a given string of parentheses and brackets. Your input is an array  $A[1..n]$ , where  $A[i] \in \{\textcolor{blue}{(}, \textcolor{red}{)}, \textcolor{blue}{[}, \textcolor{red}{]}\}$  for every index  $i$ .

**Solution:** Suppose  $A[1..n]$  is the input string. For all indices  $i$  and  $k$ , let  $LBS(i, k)$  denote the length of the longest balanced subsequence of the substring  $A[i..k]$ . We need to compute  $LBS(1, n)$ . This function obeys the following recurrence:

$$LBS(i, j) = \begin{cases} 0 & \text{if } i \geq k \\ \max \left\{ \begin{array}{l} 2 + LBS(i+1, k-1) \\ \max_{j=1}^{k-1} (LBS(i, j) + LBS(j+1, k)) \end{array} \right\} & \text{if } A[i] \sim A[k] \\ \max_{j=1}^{k-1} (LBS(i, j) + LBS(j+1, k)) & \text{otherwise} \end{cases}$$

Here  $A[i] \sim A[k]$  indicates that  $A[i]$  and  $A[k]$  are matching delimiters: Either  $A[i] = \textcolor{blue}{(}$  and  $A[k] = \textcolor{red}{)}$  or  $A[i] = \textcolor{blue}{[}$  and  $A[k] = \textcolor{red}{]}$ .

We can memoize this function into a two-dimensional array  $LBS[1..n, 1..n]$ . Since every entry  $LBS[i, j]$  depends only on entries in later rows or earlier columns (or both), we can evaluate this array row-by-row from bottom up in the outer loop, scanning each row from left to right in the inner loop. The resulting algorithm runs in  $O(n^3)$  time.

```

LONGESTBALANCEDSUBSEQUENCE( $A[1..n]$ ):
  for  $i \leftarrow n$  down to 1
     $LBS[i, i] \leftarrow 0$ 
    for  $k \leftarrow i+1$  to  $n$ 
      if  $A[i] \sim A[k]$ 
         $LBS[i, k] \leftarrow LBS[i+1, k-1] + 2$ 
      else
         $LBS[i, k] \leftarrow 0$ 
      for  $j \leftarrow i$  to  $k-1$ 
         $LBS[i, k] \leftarrow \max \{LBS[i, k], LBS[i, j] + LBS[j+1, k]\}$ 
  return  $LBS[1, n]$ 
```

■

**Rubric:** 10 points, standard dynamic programming rubric

5. Oh, no! You've just been appointed as the new organizer of Giggle, Inc.'s annual mandatory holiday party! The employees at Giggle are organized into a strict hierarchy, that is, a tree with the company president at the root. The all-knowing oracles in Human Resources have assigned a real number to each employee measuring how "fun" the employee is. In order to keep things social, there is one restriction on the guest list: An employee cannot attend the party if their immediate supervisor is also present. On the other hand, the president of the company *must* attend the party, even though she has a negative fun rating; it's her company, after all.

Describe an algorithm that makes a guest list for the party that maximizes the sum of the "fun" ratings of the guests. The input to your algorithm is a rooted tree  $T$  describing the company hierarchy, where each node  $v$  has a field  $v.fun$  storing the "fun" rating of the corresponding employee.

**Solution (two functions):** We define two functions over the nodes of  $T$ .

- $\text{MaxFunYes}(v)$  is the maximum total "fun" of a legal party among the descendants of  $v$ , where  $v$  is definitely invited.
- $\text{MaxFunNo}(v)$  is the maximum total "fun" of a legal party among the descendants of  $v$ , where  $v$  is definitely not invited.

We need to compute  $\text{MaxFunYes}(\text{root})$ . These two functions obey the following mutual recurrences:

$$\begin{aligned}\text{MaxFunYes}(v) &= v.fun + \sum_{\text{children } w \text{ of } v} \text{MaxFunNo}(w) \\ \text{MaxFunNo}(v) &= \sum_{\text{children } w \text{ of } v} \max\{\text{MaxFunYes}(w), \text{MaxFunNo}(w)\}\end{aligned}$$

(These recurrences do not require separate base cases, because  $\sum \emptyset = 0$ .) We can memoize these functions by adding two additional fields  $v.yes$  and  $v.no$  to each node  $v$  in the tree. The values at each node depend only on the values at its children, so we can compute all  $2n$  values using a postorder traversal of  $T$ .

```
BESTPARTY( $T$ ):
    COMPUTEMAXFUN( $T.\text{root}$ )
    return  $T.\text{root}.yes$ 
```

```
COMPUTEMAXFUN( $v$ ):
     $v.yes \leftarrow v.fun$ 
     $v.no \leftarrow 0$ 
    for all children  $w$  of  $v$ 
        COMPUTEMAXFUN( $w$ )
         $v.yes \leftarrow v.yes + w.no$ 
         $v.no \leftarrow v.no + \max\{w.yes, w.no\}$ 
```

(Yes, this is still dynamic programming; we're only traversing the tree recursively because that's the most natural way to traverse trees!) The algorithm spends  $O(1)$  time at each node, and therefore runs in  $O(n)$  time altogether. ■

<sup>a</sup>A naïve recursive implementation would run in  $O(\phi^n)$  time in the worst case, where  $\phi = (1 + \sqrt{5})/2 \approx 1.618$  is the golden ratio. The worst-case tree is a path—every non-leaf node has exactly one child.

**Solution (one function):** For each node  $v$  in the input tree  $T$ , let  $\text{MaxFun}(v)$  denote the maximum total “fun” of a legal party among the descendants of  $v$ , where  $v$  may or may not be invited.

The president of the company must be invited, so none of the president’s “children” in  $T$  can be invited. Thus, the value we need to compute is

$$\text{root.fun} + \sum_{\text{grandchildren } w \text{ of root}} \text{MaxFun}(w).$$

The function  $\text{MaxFun}$  obeys the following recurrence:

$$\text{MaxFun}(v) = \max \left\{ \begin{array}{l} v.\text{fun} + \sum_{\text{grandchildren } x \text{ of } v} \text{MaxFun}(x) \\ \sum_{\text{children } w \text{ of } v} \text{MaxFun}(w) \end{array} \right\}$$

(This recurrence does not require a separate base case, because  $\sum \emptyset = 0$ .) We can memoize this function by adding an additional field  $v.\text{maxFun}$  to each node  $v$  in the tree. The value at each node depends only on the values at its children and grandchildren, so we can compute all values using a postorder traversal of  $T$ .

```
BESTPARTY( $T$ ):
    COMPUTEMAXFUN( $T.\text{root}$ )
     $party \leftarrow T.\text{root.fun}$ 
    for all children  $w$  of  $T.\text{root}$ 
        for all children  $x$  of  $w$ 
             $party \leftarrow party + x.\text{maxFun}$ 
    return  $party$ 
```

```
COMPUTEMAXFUN( $v$ ):
     $yes \leftarrow v.\text{fun}$ 
     $no \leftarrow 0$ 
    for all children  $w$  of  $v$ 
        COMPUTEMAXFUN( $w$ )
         $no \leftarrow no + w.\text{maxFun}$ 
        for all children  $x$  of  $w$ 
             $yes \leftarrow yes + x.\text{maxFun}$ 
     $v.\text{maxFun} \leftarrow \max\{yes, no\}$ 
```

(Yes, this is still dynamic programming; we’re only traversing the tree recursively because that’s the most natural way to traverse trees!<sup>a</sup>)

The algorithm spends  $O(1)$  time at each node (because each node has exactly one parent and one grandparent) and therefore runs in  $O(n)$  time altogether. ■

<sup>a</sup>Like the previous solution, a direct recursive implementation would run in  $O(\phi^n)$  time in the worst case, where  $\phi = (1 + \sqrt{5})/2 \approx 1.618$  is the golden ratio.

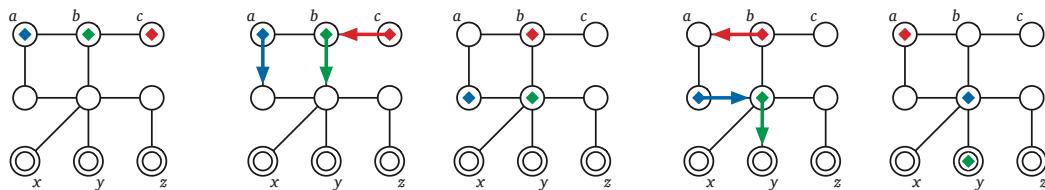
**Rubric:** 10 points: standard dynamic programming rubric. These are not the only correct solutions.

# CS/ECE 374 A ✦ Spring 2018

## ❖ Homework 7 ❖

Due Tuesday, March 27, 2018 at 8pm  
 (after Spring Break)

1. Consider the following solitaire game, played on a connected undirected graph  $G$ . Initially, tokens are placed on three start vertices  $a, b, c$ . In each turn, you *must* move *all three* tokens, by moving each token along an edge from its current vertex to an adjacent vertex. At the end of each turn, the three tokens *must* lie on three different vertices. Your goal is to move the tokens onto three goal vertices  $x, y, z$ ; it does not matter which token ends up on which goal vertex.



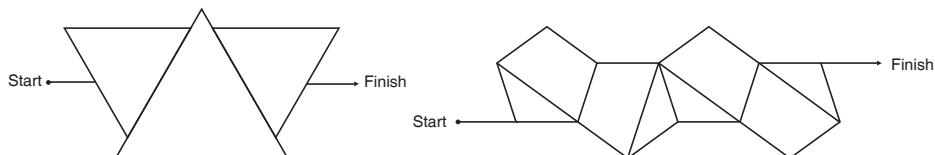
The initial configuration of the puzzle and the first two turns of a solution.

Describe and analyze an algorithm to determine whether this puzzle is solvable. Your input consists of the graph  $G$ , the start vertices  $a, b, c$ , and the goal vertices  $x, y, z$ . Your output is a single bit: TRUE or FALSE. [Hint: You've seen this sort of thing before.]

2. The following puzzles appear in my daughter's elementary-school math workbook.<sup>1</sup>

**PRACTICE**

Complete each angle maze below by tracing a path from start to finish that has only acute angles.



Describe and analyze an algorithm to solve arbitrary acute-angle mazes.

You are given a connected undirected graph  $G$ , whose vertices are points in the plane and whose edges are line segments. Edges do not intersect, except at their endpoints. For example, a drawing of the letter X would have five vertices and four edges; the first maze above has 13 vertices and 15 edges. You are also given two vertices Start and Finish.

Your algorithm should return TRUE if  $G$  contains a walk from Start to Finish that has only acute angles, and FALSE otherwise. Formally, a walk through  $G$  is valid if, for any two consecutive edges  $u \rightarrow v \rightarrow w$  in the walk, either  $\angle uvw = \pi$  or  $0 < \angle uvw < \pi/2$ . Assume you have a subroutine that can determine in  $O(1)$  time whether two segments with a common vertex define a straight, obtuse, right, or acute angle.

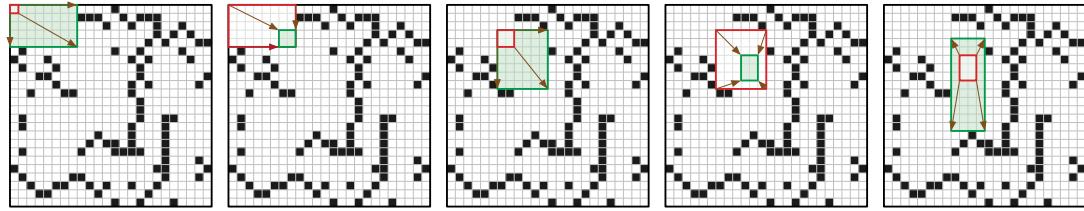
<sup>1</sup>Jason Batterson and Shannon Rogers, *Beast Academy Math: Practice 3A*, 2012. See <https://www.beastacademy.com/resources/printables.php> for more examples.

3. **Rectangle Walk** is a new abstract puzzle game, available for only 99¢ on Steam, iOS, Android, Xbox One, Playstation 5, Nintendo Wii U, Atari 2600, Palm Pilot, Commodore 64, TRS-80, Sinclair ZX-1, DEC PDP-8, ILLIAC V, Zuse Z3, Duramesc, Odhner Arithmometer, Analytical Engine, Jacquard Loom, Horologium mirabile Lundense, Leibniz Stepped Reckoner, Antikythera Mechanism, and Pile of Sticks.

The game is played on an  $n \times n$  grid of black and white squares. The player moves a rectangle through this grid, subject to the following conditions:

- The rectangle must be aligned with the grid; that is, the top, bottom, left, and right coordinates must be integers.
- The rectangle must fit within the  $n \times n$  grid, and it must contain at least one grid cell.
- The rectangle must not contain a black square.
- In a single move, the player can replace the current rectangle  $r$  with any rectangle  $r'$  that either contains  $r$  or is contained in  $r$ .

Initially, the player's rectangle is a  $1 \times 1$  square in the upper right corner. The player's goal is to reach a  $1 \times 1$  square in the bottom left corner using as few moves as possible.



The first five steps in a Rectangle Walk.

Describe and analyze an algorithm to compute the length of the shortest Rectangle Walk in a given bitmap. Your input is an array  $M[1..n, 1..n]$ , where  $M[i, j] = 1$  indicates a black square and  $M[i, j] = 0$  indicates a white square. You can assume that a valid rectangle walk exists; in particular,  $M[1, 1] = 0$  and  $M[n, n] = 0$ . For example, given the bitmap shown above, (I think) your algorithm should return the integer 18.

## Solved Problem

4. Professor McClane takes you out to a lake and hands you three empty jars. Each jar holds a positive integer number of gallons; the capacities of the three jars may or may not be different. The professor then demands that you put exactly  $k$  gallons of water into one of the jars (which one doesn't matter), for some integer  $k$ , using only the following operations:
- Fill a jar with water from the lake until the jar is full.
  - Empty a jar of water by pouring water into the lake.
  - Pour water from one jar to another, until either the first jar is empty or the second jar is full, whichever happens first.

For example, suppose your jars hold 6, 10, and 15 gallons. Then you can put 13 gallons of water into the third jar in six steps:

- Fill the third jar from the lake.
- Fill the first jar from the third jar. (Now the third jar holds 9 gallons.)
- Empty the first jar into the lake.
- Fill the second jar from the lake.
- Fill the first jar from the second jar. (Now the second jar holds 4 gallons.)
- Empty the second jar into the third jar.

Describe and analyze an efficient algorithm that either finds the smallest number of operations that leave exactly  $k$  gallons in any jar, or reports correctly that obtaining exactly  $k$  gallons of water is impossible. Your input consists of the capacities of the three jars and the positive integer  $k$ . For example, given the four numbers 6, 10, 15 and 13 as input, your algorithm should return the number 6 (for the sequence of operations listed above).

**Solution:** Let  $A, B, C$  denote the capacities of the three jars. We reduce the problem to breadth-first search in the following directed graph:

- $V = \{(a, b, c) \mid 0 \leq a \leq A \text{ and } 0 \leq b \leq B \text{ and } 0 \leq c \leq C\}$ . Each vertex corresponds to a possible **configuration** of water in the three jars. There are  $(A+1)(B+1)(C+1) = O(ABC)$  vertices altogether.
- The graph has a directed edge  $(a, b, c) \rightarrow (a', b', c')$  whenever it is possible to move from the first configuration to the second in one step. Specifically, there is an edge from  $(a, b, c)$  to each of the following vertices (except those already equal to  $(a, b, c)$ ):
  - $(0, b, c)$  and  $(a, 0, c)$  and  $(a, b, 0)$  — dumping a jar into the lake
  - $(A, b, c)$  and  $(a, B, c)$  and  $(a, b, C)$  — filling a jar from the lake
  - $\begin{cases} (0, a+b, c) & \text{if } a+b \leq B \\ (a+b-B, B, c) & \text{if } a+b \geq B \end{cases}$  — pouring from jar 1 into jar 2
  - $\begin{cases} (0, b, a+c) & \text{if } a+c \leq C \\ (a+c-C, b, C) & \text{if } a+c \geq C \end{cases}$  — pouring from jar 1 into jar 3

- $\begin{cases} (a+b, 0, c) & \text{if } a+b \leq A \\ (A, a+b-A, c) & \text{if } a+b \geq A \end{cases}$  — pouring from jar 2 into jar 1
- $\begin{cases} (a, 0, b+c) & \text{if } b+c \leq C \\ (a, b+c-C, C) & \text{if } b+c \geq C \end{cases}$  — pouring from jar 2 into jar 3
- $\begin{cases} (a+c, b, 0) & \text{if } a+c \leq A \\ (A, b, a+c-A) & \text{if } a+c \geq A \end{cases}$  — pouring from jar 3 into Jar 1
- $\begin{cases} (a, b+c, 0) & \text{if } b+c \leq B \\ (a, B, b+c-B) & \text{if } b+c \geq B \end{cases}$  — pouring from jar 3 into jar 2

Since each vertex has at most 12 outgoing edges, there are at most  $12(A+1) \times (B+1)(C+1) = O(ABC)$  edges altogether.

To solve the jars problem, we need to find the *shortest path* in  $G$  from the start vertex  $(0, 0, 0)$  to any target vertex of the form  $(k, \cdot, \cdot)$  or  $(\cdot, k, \cdot)$  or  $(\cdot, \cdot, k)$ . We can compute this shortest path by calling *breadth-first search* starting at  $(0, 0, 0)$ , and then examining every target vertex by brute force. If BFS does not visit any target vertex, we report that no legal sequence of moves exists. Otherwise, we find the target vertex closest to  $(0, 0, 0)$  and trace its parent pointers back to  $(0, 0, 0)$  to determine the shortest sequence of moves. The resulting algorithm runs in  $O(V + E) = O(ABC)$  time.

We can make this algorithm faster by observing that every move either leaves at least one jar empty or leaves at least one jar full. Thus, we only need vertices  $(a, b, c)$  where either  $a = 0$  or  $b = 0$  or  $c = 0$  or  $a = A$  or  $b = B$  or  $c = C$ ; no other vertices are reachable from  $(0, 0, 0)$ . The number of non-redundant vertices and edges is  $O(AB + BC + AC)$ . Thus, if we only construct and search the relevant portion of  $G$ , the algorithm runs in  $O(AB + BC + AC)$  time. ■

**Rubric:** 10 points: standard graph reduction rubric (see next page)

- Brute force construction is fine.
- 1 for calling Dijkstra instead of BFS
- max 8 points for  $O(ABC)$  time; scale partial credit.

**Standard rubric for graph reduction problems.** For problems out of 10 points:

- + 1 for correct vertices, *including English explanation for each vertex*
- + 1 for correct edges
  - $\frac{1}{2}$  for forgetting “directed” if the graph is directed
- + 1 for stating the correct problem (in this case, “shortest path”)
  - “Breadth-first search” is not a problem; it’s an algorithm!
- + 1 for correctly applying the correct algorithm (in this case, “breadth-first search from  $(0, 0, 0)$  and then examine every target vertex”)
- + 1 for time analysis in terms of the input parameters.
- + 5 for other details of the reduction
  - If your graph is constructed by naive brute force, you do not need to describe the construction algorithm; in this case, points for vertices, edges, problem, algorithm, and running time are all doubled.
  - Otherwise, apply the appropriate rubric, *including Deadly Sins*, to the construction algorithm. For example, for a solution that uses dynamic programming to build the graph quickly, apply the standard dynamic programming rubric.

# CS/ECE 374 A ✦ Spring 2018

## ❖ Homework 8 ❖

Due Tuesday, April 3, 2018 at 8pm

---

This is the last homework before Midterm 2.

---

1. After moving to a new city, you decide to choose a walking route from your home to your new office. To get a good daily workout, your route must consist of an uphill path (for exercise) followed by a downhill path (to cool down), or just an uphill path, or just a downhill path.<sup>1</sup> (You'll walk the same path home, so you'll get exercise one way or the other.) But you also want the *shortest* path that satisfies these conditions, so that you actually get to work on time.

Your input consists of an undirected graph  $G$ , whose vertices represent intersections and whose edges represent road segments, along with a start vertex  $s$  and a target vertex  $t$ . Every vertex  $v$  has a value  $h(v)$ , which is the height of that intersection above sea level, and each edge  $uv$  has a value  $\ell(uv)$ , which is the length of that road segment.

- (a) Describe and analyze an algorithm to find the shortest uphill–downhill walk from  $s$  to  $t$ . Assume all vertex heights are distinct.
  - (b) Suppose you discover that there is no path from  $s$  to  $t$  with the structure you want. Describe an algorithm to find a path from  $s$  to  $t$  that alternates between “uphill” and “downhill” subpaths as few times as possible, and has minimum length among all such paths. (There may be even shorter paths with more alternations, but you don’t care about them.) Again, assume all vertex heights are distinct.
2. Let  $G = (V, E)$  be a directed graph with weighted edges; edge weights could be positive, negative, or zero.
    - (a) How could we delete an arbitrary vertex  $v$  from this graph, without changing the shortest-path distance between any other pair of vertices? Describe an algorithm that constructs a directed graph  $G' = (V', E')$  with weighted edges, where  $V' = V \setminus \{v\}$ , and the shortest-path distance between any two nodes in  $G'$  is equal to the shortest-path distance between the same two nodes in  $G$ , in  $O(V^2)$  time.
    - (b) Now suppose we have already computed all shortest-path distances in  $G'$ . Describe an algorithm to compute the shortest-path distances in the original graph  $G$  from  $v$  to every other vertex, and from every other vertex to  $v$ , all in  $O(V^2)$  time.
    - (c) Combine parts (a) and (b) into another all-pairs shortest path algorithm that runs in  $O(V^3)$  time. (The resulting algorithm is *almost* the same as Floyd-Warshall!)

---

<sup>1</sup>A *hill* is an area of land that extends above the surrounding terrain, usually at a fairly gentle gradient. Like a building, but smoother and made of dirt and rock and trees instead of steel and concrete. It’s hard to explain.

3. The first morning after returning from a glorious spring break, Alice wakes to discover that her car won't start, so she has to get to her classes at Sham-Poobanana University by public transit. She has a complete transit schedule for Poobanana County. The bus routes are represented in the schedule by a directed graph  $G$ , whose vertices represent bus stops and whose edges represent bus routes between those stops. For each edge  $u \rightarrow v$ , the schedule records three positive real numbers:

- $\ell(u \rightarrow v)$  is the length of the bus ride from stop  $u$  to stop  $v$  (in minutes)
- $f(u \rightarrow v)$  is the first time (in minutes past 12am) that a bus leaves stop  $u$  for stop  $v$ .
- $\Delta(u \rightarrow v)$  is the time between successive departures from stop  $u$  to stop  $v$  (in minutes).

Thus, the first bus for this route leaves  $u$  at time  $f(u \rightarrow v)$  and arrives at  $v$  at time  $f(u \rightarrow v) + \ell(u \rightarrow v)$ , the second bus leaves  $u$  at time  $f(u \rightarrow v) + \Delta(u \rightarrow v)$  and arrives at  $v$  at time  $f(u \rightarrow v) + \Delta(u \rightarrow v) + \ell(u \rightarrow v)$ , the third bus leaves  $u$  at time  $f(u \rightarrow v) + 2 \cdot \Delta(u \rightarrow v)$  and arrives at  $v$  at time  $f(u \rightarrow v) + 2 \cdot \Delta(u \rightarrow v) + \ell(u \rightarrow v)$ , and so on.

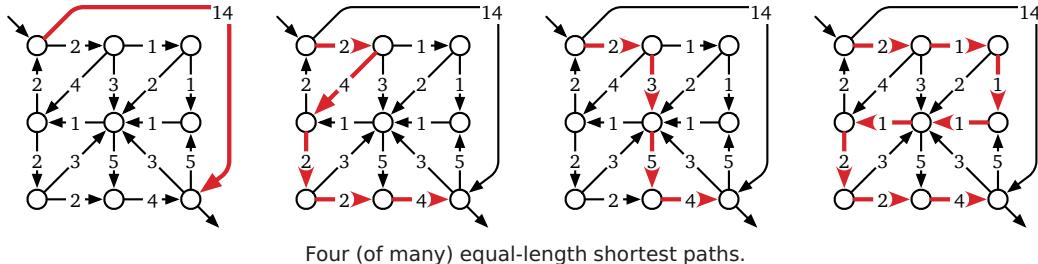
Alice wants to leave from stop  $s$  (her home) at a certain time and arrive at stop  $t$  (The See-Bull Center for Fake News Detection) as quickly as possible. If Alice arrives at a stop on one bus at the exact time that another bus is scheduled to leave, she can catch the second bus. Because she's a student at SPU, Alice can ride the bus for free, so she doesn't care how many times she has to change buses.

Describe and analyze an algorithm to find the earliest time Alice can reach her destination. Your input consists of the directed graph  $G = (V, E)$ , the vertices  $s$  and  $t$ , the values  $\ell(e), f(e), \Delta(e)$  for each edge  $e \in E$ , and Alice's starting time (in minutes past 12am).

*[Hint: In this rare instance, modifying the algorithm may be more efficient than modifying the input graph. Don't describe the algorithm from scratch; just describe your changes.]*

## Solved Problem

4. Although we typically speak of “the” shortest path from one vertex to another, a single graph could contain several minimum-length paths with the same endpoints.



Describe and analyze an algorithm to determine the *number* of shortest paths from a source vertex  $s$  to a target vertex  $t$  in an arbitrary directed graph  $G$  with weighted edges. You may assume that all edge weights are positive and that the necessary arithmetic operations can be performed in  $O(1)$  time each.

[Hint: Compute shortest path distances from  $s$  to every other vertex. Throw away all edges that cannot be part of a shortest path from  $s$  to another vertex. What's left?]

**Solution:** We start by computing shortest-path distances  $dist(v)$  from  $s$  to  $v$ , for every vertex  $v$ , using Dijkstra's algorithm. Call an edge  $u \rightarrow v$  **tight** if  $dist(u) + w(u \rightarrow v) = dist(v)$ . Every edge in a shortest path from  $s$  to  $t$  must be tight. Conversely, every path from  $s$  to  $t$  that uses only tight edges has total length  $dist(t)$  and is therefore a shortest path!

Let  $H$  be the subgraph of all tight edges in  $G$ . We can easily construct  $H$  in  $O(V + E)$  time. Because all edge weights are positive,  $H$  is a directed acyclic graph. It remains only to count the number of paths from  $s$  to  $t$  in  $H$ .

For any vertex  $v$ , let  $NumPaths(v)$  denote the number of paths in  $H$  from  $v$  to  $t$ ; we need to compute  $NumPaths(s)$ . This function satisfies the following simple recurrence:

$$NumPaths(v) = \begin{cases} 1 & \text{if } v = t \\ \sum_{v \rightarrow w} NumPaths(w) & \text{otherwise} \end{cases}$$

In particular, if  $v$  is a sink but  $v \neq t$  (and thus there are no paths from  $v$  to  $t$ ), this recurrence correctly gives us  $NumPaths(v) = \sum \emptyset = 0$ .

We can memoize this function into the graph itself, storing each value  $NumPaths(v)$  at the corresponding vertex  $v$ . Since each subproblem depends only on its successors in  $H$ , we can compute  $NumPaths(v)$  for all vertices  $v$  by considering the vertices in reverse topological order, or equivalently, by performing a depth-first search of  $H$  starting at  $s$ . The resulting algorithm runs in  $O(V + E)$  time.

The overall running time of the algorithm is dominated by Dijkstra's algorithm in the preprocessing phase, which runs in  $O(E \log V)$  time. ■

**Rubric:** 10 points = 5 points for reduction to counting paths in a dag (standard graph reduction rubric) + 5 points for the path-counting algorithm (standard dynamic programming rubric)

# CS/ECE 374 A ✦ Spring 2018

## ❖ Homework 9 ❖

Due Tuesday, April 17, 2018 at 8pm

---

1. For any integer  $k$ , the problem  $k$ SAT is defined as follows:
  - INPUT: A boolean formula  $\Phi$  in conjunctive normal form, with exactly  $k$  distinct literals in each clause.
  - OUTPUT: TRUE if  $\Phi$  has a satisfying assignment, and FALSE otherwise.
  - (a) Describe a polynomial-time reduction from 2SAT to 3SAT, and prove that your reduction is correct.
  - (b) Describe and analyze a polynomial-time algorithm for 2SAT. [Hint: This problem is strongly connected to topics covered earlier in the semester.]
  - (c) Why don't these results imply a polynomial-time algorithm for 3SAT?
2. This problem asks you to describe polynomial-time reductions between two closely related problems:
  - SUBSETSUM: Given a set  $S$  of positive integers and a target integer  $T$ , is there a subset of  $S$  whose sum is  $T$ ?
  - PARTITION: Given a set  $S$  of positive integers, is there a way to partition  $S$  into two subsets  $S_1$  and  $S_2$  that have the same sum?
  - (a) Describe a polynomial-time reduction from SUBSETSUM to PARTITION.
  - (b) Describe a polynomial-time reduction from PARTITION to SUBSETSUM.
3. *Pebbling* is a solitaire game played on an undirected graph  $G$ , where each vertex has zero or more *pebbles*. A single *pebbling move* removes two pebbles from some vertex  $v$  and adds one pebble to an arbitrary neighbor of  $v$ . (Obviously,  $v$  must have at least two pebbles before the move.) The PEBBLECLEARING problem asks, given a graph  $G = (V, E)$  and a pebble count  $p(v)$  for each vertex  $v$ , whether is there a sequence of pebbling moves that removes all but one pebble. Prove that PEBBLECLEARING is NP-hard.

Don't forget to prove that your reductions are correct.

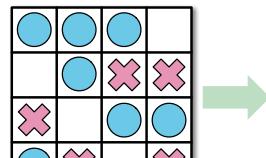
## Solved Problem

4. Consider the following solitaire game. The puzzle consists of an  $n \times m$  grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions:

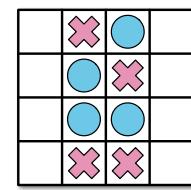
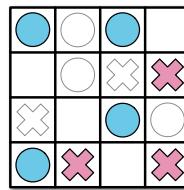
- (1) Every row contains at least one stone.
- (2) No column contains stones of both colors.

For some initial configurations of stones, reaching this goal is impossible; see the example below.

Prove that it is NP-hard to determine, given an initial configuration of red and blue stones, whether this puzzle can be solved.



A solvable puzzle and one of its many solutions.



An unsolvable puzzle.

**Solution:** We show that this puzzle is NP-hard by describing a reduction from 3SAT.

Let  $\Phi$  be a 3CNF boolean formula with  $m$  variables and  $n$  clauses. We transform this formula into a puzzle configuration in polynomial time as follows. The size of the board is  $n \times m$ . The stones are placed as follows, for all indices  $i$  and  $j$ :

- If the variable  $x_j$  appears in the  $i$ th clause of  $\Phi$ , we place a blue stone at  $(i, j)$ .
- If the negated variable  $\bar{x}_j$  appears in the  $i$ th clause of  $\Phi$ , we place a red stone at  $(i, j)$ .
- Otherwise, we leave cell  $(i, j)$  blank.

We claim that this puzzle has a solution if and only if  $\Phi$  is satisfiable. This claim immediately implies that solving the puzzle is NP-hard. We prove our claim as follows:

⇒ First, suppose  $\Phi$  is satisfiable; consider an arbitrary satisfying assignment. For each index  $j$ , remove stones from column  $j$  according to the value assigned to  $x_j$ :

- If  $x_j = \text{TRUE}$ , remove all red stones from column  $j$ .
- If  $x_j = \text{FALSE}$ , remove all blue stones from column  $j$ .

In other words, remove precisely the stones that correspond to FALSE literals. Because every variable appears in at least one clause, each column now contains stones of only one color (if any). On the other hand, each clause of  $\Phi$  must contain at least one TRUE literal, and thus each row still contains at least one stone. We conclude that the puzzle is satisfiable.

$\Leftarrow$  On the other hand, suppose the puzzle is solvable; consider an arbitrary solution. For each index  $j$ , assign a value to  $x_j$  depending on the colors of stones left in column  $j$ :

- If column  $j$  contains blue stones, set  $x_j = \text{TRUE}$ .
- If column  $j$  contains red stones, set  $x_j = \text{FALSE}$ .
- If column  $j$  is empty, set  $x_j$  arbitrarily.

In other words, assign values to the variables so that the literals corresponding to the remaining stones are all TRUE. Each row still has at least one stone, so each clause of  $\Phi$  contains at least one TRUE literal, so this assignment makes  $\Phi = \text{TRUE}$ . We conclude that  $\Phi$  is satisfiable.

This reduction clearly requires only polynomial time. ■

**Rubric (Standard polynomial-time reduction rubric): 10 points =**

- + 3 points for the reduction itself
  - For an NP-hardness proof, the reduction must be from a known NP-hard problem. You can use any of the NP-hard problems listed in the lecture notes (except the one you are trying to prove NP-hard, of course). **See the list on the next page.**
- + 3 points for the “if” proof of correctness
- + 3 points for the “only if” proof of correctness
- + 1 point for writing “polynomial time”
  - An incorrect polynomial-time reduction that still satisfies half of the correctness proof is worth at most 4/10.
  - A reduction in the wrong direction is worth 0/10.

**Some useful NP-hard problems.** You are welcome to use any of these in your own NP-hardness proofs, except of course for the specific problem you are trying to prove NP-hard.

**CIRCUITSAT:** Given a boolean circuit, are there any input values that make the circuit output TRUE?

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three distinct literals per clause, does the formula have a satisfying assignment?

**MAXINDEPENDENTSET:** Given an undirected graph  $G$ , what is the size of the largest subset of vertices in  $G$  that have no edges among them?

**MAXCLIQUE:** Given an undirected graph  $G$ , what is the size of the largest complete subgraph of  $G$ ?

**MINVERTEXCOVER:** Given an undirected graph  $G$ , what is the size of the smallest subset of vertices that touch every edge in  $G$ ?

**MINSETCOVER:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subcollection whose union is  $S$ ?

**MINHITTINGSET:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subset of  $S$  that intersects every subset  $S_i$ ?

**3COLOR:** Given an undirected graph  $G$ , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

**HAMILTONIANPATH:** Given graph  $G$  (either directed or undirected), is there a path in  $G$  that visits every vertex exactly once?

**HAMILTONIANCYCLE:** Given a graph  $G$  (either directed or undirected), is there a cycle in  $G$  that visits every vertex exactly once?

**TRAVELINGSALESMAN:** Given a graph  $G$  (either directed or undirected) with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in  $G$ ?

**LONGESTPATH:** Given a graph  $G$  (either directed or undirected, possibly with weighted edges), what is the length of the longest simple path in  $G$ ?

**STEINERTREE:** Given an undirected graph  $G$  with some of the vertices marked, what is the minimum number of edges in a subtree of  $G$  that contains every marked vertex?

**SUBSETSUM:** Given a set  $X$  of positive integers and an integer  $k$ , does  $X$  have a subset whose elements sum to  $k$ ?

**PARTITION:** Given a set  $X$  of positive integers, can  $X$  be partitioned into two subsets with the same sum?

**3PARTITION:** Given a set  $X$  of  $3n$  positive integers, can  $X$  be partitioned into  $n$  three-element subsets, all with the same sum?

**INTEGERLINEARPROGRAMMING:** Given a matrix  $A \in \mathbb{Z}^{n \times d}$  and two vectors  $b \in \mathbb{Z}^n$  and  $c \in \mathbb{Z}^d$ , compute  $\max\{c \cdot x \mid Ax \leq b, x \geq 0, x \in \mathbb{Z}^d\}$ .

**FEASIBLEILP:** Given a matrix  $A \in \mathbb{Z}^{n \times d}$  and a vector  $b \in \mathbb{Z}^n$ , determine whether the set of feasible integer points  $\max\{x \in \mathbb{Z}^d \mid Ax \leq b, x \geq 0\}$  is empty.

**DRAUGHTS:** Given an  $n \times n$  international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

**SUPERMARIOBROTHERS:** Given an  $n \times n$  Super Mario Brothers level, can Mario reach the castle?

**STEAMEDHAMS:** Aurora borealis? At this time of year, at this time of day, in this part of the country, localized entirely within your kitchen? May I see it?

# CS/ECE 374 A ✦ Spring 2018

## ❖ Homework 10 ❖

Due Tuesday, April 24, 2018 at 8pm

---

This is the last graded homework before the final exam.

---

1. (a) A subset  $S$  of vertices in an undirected graph  $G$  is *half-independent* if each vertex in  $S$  is adjacent to *at most one* other vertex in  $S$ . Prove that finding the size of the largest half-independent set of vertices in a given undirected graph is NP-hard.  
(b) A subset  $S$  of vertices in an undirected graph  $G$  is *sort-of-independent* if each vertex in  $S$  is adjacent to *at most 374* other vertices in  $S$ . Prove that finding the size of the largest sort-of-independent set of vertices in a given undirected graph is NP-hard.
  
2. Fix an alphabet  $\Sigma = \{\texttt{0}, \texttt{1}\}$ . Prove that the following problems are NP-hard.<sup>1</sup>
  - (a) Given a regular expression  $R$  over the alphabet  $\Sigma$ , is  $L(R) \neq \Sigma^*$ ?
  - (b) Given an NFA  $M$  over the alphabet  $\Sigma$ , is  $L(M) \neq \Sigma^*$ ?

[Hint: Encode all the **bad** choices for some problem into a regular expression  $R$ , so that if all choices are bad, then  $L(R) = \Sigma^*$ .]

3. Let  $\langle M \rangle$  denote the encoding of a Turing machine  $M$  (or if you prefer, the Python source code for the executable code  $M$ ). Recall that  $x \bullet y$  denotes the concatenation of strings  $x$  and  $y$ . Prove that the following language is undecidable.

$$\text{SELFSELFACCEPT} := \{\langle M \rangle \mid M \text{ accepts the string } \langle M \rangle \bullet \langle M \rangle\}$$

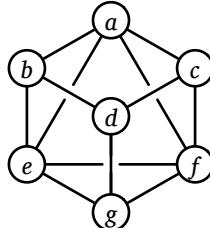
Note that Rice's theorem does *not* apply to this language.

---

<sup>1</sup>In fact, both of these problems are NP-hard even when  $|\Sigma| = 1$ , but proving that is much more difficult.

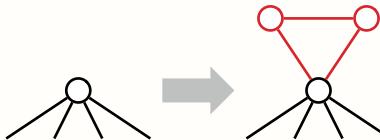
## Solved Problem

4. A *double-Hamiltonian tour* in an undirected graph  $G$  is a closed walk that visits every vertex in  $G$  exactly twice. Prove that it is NP-hard to decide whether a given graph  $G$  has a double-Hamiltonian tour.



This graph contains the double-Hamiltonian tour  $a \rightarrow b \rightarrow d \rightarrow g \rightarrow e \rightarrow b \rightarrow d \rightarrow c \rightarrow f \rightarrow a \rightarrow c \rightarrow f \rightarrow g \rightarrow e \rightarrow a$ .

**Solution:** We prove the problem is NP-hard with a reduction from the standard Hamiltonian cycle problem. Let  $G$  be an arbitrary undirected graph. We construct a new graph  $H$  by attaching a small gadget to every vertex of  $G$ . Specifically, for each vertex  $v$ , we add two vertices  $v^\sharp$  and  $v^\flat$ , along with three edges  $vv^\flat$ ,  $vv^\sharp$ , and  $v^\flat v^\sharp$ .



A vertex in  $G$ , and the corresponding vertex gadget in  $H$ .

I claim that  $G$  has a Hamiltonian cycle if and only if  $H$  has a double-Hamiltonian tour.

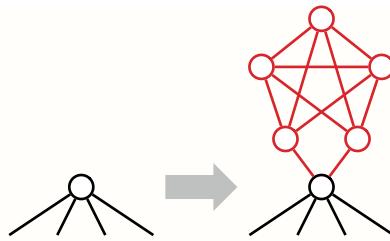
$\implies$  Suppose  $G$  has a Hamiltonian cycle  $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n \rightarrow v_1$ . We can construct a double-Hamiltonian tour of  $H$  by replacing each vertex  $v_i$  with the following walk:

$$\dots \rightarrow v_i \rightarrow v_i^\flat \rightarrow v_i^\sharp \rightarrow v_i^\flat \rightarrow v_i^\sharp \rightarrow v_i \rightarrow \dots$$

$\Leftarrow$  Conversely, suppose  $H$  has a double-Hamiltonian tour  $D$ . Consider any vertex  $v$  in the original graph  $G$ ; the tour  $D$  must visit  $v$  exactly twice. Those two visits split  $D$  into two closed walks, each of which visits  $v$  exactly once. Any walk from  $v^\flat$  or  $v^\sharp$  to any other vertex in  $H$  must pass through  $v$ . Thus, one of the two closed walks visits only the vertices  $v$ ,  $v^\flat$ , and  $v^\sharp$ . Thus, if we simply remove the vertices in  $H \setminus G$  from  $D$ , we obtain a closed walk in  $G$  that visits every vertex in  $G$  once.

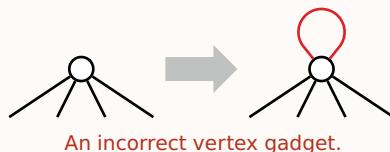
Given any graph  $G$ , we can clearly construct the corresponding graph  $H$  in polynomial time.

With more effort, we can construct a graph  $H$  that contains a double-Hamiltonian tour **that traverses each edge of  $H$  at most once** if and only if  $G$  contains a Hamiltonian cycle. For each vertex  $v$  in  $G$  we attach a more complex gadget containing five vertices and eleven edges, as shown on the next page.



■

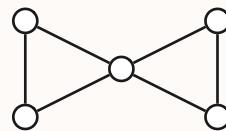
**Non-solution (self-loops):** We attempt to prove the problem is NP-hard with a reduction from the Hamiltonian cycle problem. Let  $G$  be an arbitrary undirected graph. We construct a new graph  $H$  by attaching a self-loop every vertex of  $G$ . Given any graph  $G$ , we can clearly construct the corresponding graph  $H$  in polynomial time.



Suppose  $G$  has a Hamiltonian cycle  $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n \rightarrow v_1$ . We can construct a double-Hamiltonian tour of  $H$  by alternating between edges of the Hamiltonian cycle and self-loops:

$$v_1 \rightarrow v_1 \rightarrow v_2 \rightarrow v_2 \rightarrow v_3 \rightarrow \dots \rightarrow v_n \rightarrow v_n \rightarrow v_1.$$

On the other hand, if  $H$  has a double-Hamiltonian tour, we *cannot* conclude that  $G$  has a Hamiltonian cycle, because we cannot guarantee that a double-Hamiltonian tour in  $H$  uses *any* self-loops. The graph  $G$  shown below is a counterexample; it has a double-Hamiltonian tour (even before adding self-loops!) but no Hamiltonian cycle.



This graph has a double-Hamiltonian tour.



**Rubric:** 10 points, standard polynomial-time reduction rubric

# CS/ECE 374 A ✦ Spring 2018

## ❖ “Homework” 11 ❖

“Due” Tuesday, May 1, 2018

---

This homework is optional. However, **similar undecidability questions may appear on the final exam**, so we still strongly recommend treating at least those questions as regular homework. Solutions will be released next Tuesday as usual.

---

1. Let  $M$  be a Turing machine, let  $w$  be an arbitrary input string, and let  $s$  be an integer. We say that  $M$  accepts  $w$  in space  $s$  if, given  $w$  as input,  $M$  accesses only the first  $s$  (or fewer) cells on its tape and eventually accepts.

\*(a) Sketch a Turing machine/algorithm that correctly decides the following language:

$$\text{SQUARESPACE} = \{ \langle M, w \rangle \mid M \text{ accepts } w \text{ in space } |w|^2 \}$$

(b) Prove that the following language is undecidable:

$$\text{SOMESENSESPACE} = \{ \langle M \rangle \mid M \text{ accepts at least one string } w \text{ in space } |w|^2 \}$$

2. Consider the following language:

$$\text{PICKY} = \left\{ \langle M \rangle \mid \begin{array}{l} M \text{ accepts at least one input string} \\ \text{and } M \text{ rejects at least one input string} \end{array} \right\}$$

- (a) Prove that PICKY is undecidable.
- (b) Sketch a Turing machine/algorithm that *accepts* PICKY.

The following problems ask you to prove some “obvious” claims about recursively-defined string functions. In each case, we want a self-contained, step-by-step induction proof that builds on formal definitions and prior results, *not* on intuition. In particular, your proofs must refer to the formal recursive definitions of string length and string concatenation:

$$|w| := \begin{cases} 0 & \text{if } w = \epsilon \\ 1 + |x| & \text{if } w = ax \text{ for some symbol } a \text{ and some string } x \end{cases}$$

$$w \bullet z := \begin{cases} z & \text{if } w = \epsilon \\ a \cdot (x \bullet z) & \text{if } w = ax \text{ for some symbol } a \text{ and some string } x \end{cases}$$

You may freely use the following results, which are proved in the lecture notes:

**Lemma 1:**  $w \bullet \epsilon = w$  for all strings  $w$ .

**Lemma 2:**  $|w \bullet x| = |w| + |x|$  for all strings  $w$  and  $x$ .

**Lemma 3:**  $(w \bullet x) \bullet y = w \bullet (x \bullet y)$  for all strings  $w$ ,  $x$ , and  $y$ .

The **reversal**  $w^R$  of a string  $w$  is defined recursively as follows:

$$w^R := \begin{cases} \epsilon & \text{if } w = \epsilon \\ x^R \bullet a & \text{if } w = ax \text{ for some symbol } a \text{ and some string } x \end{cases}$$

For example,  $\text{STRESSED}^R = \text{DESSERTS}$  and  $\text{WTF374}^R = \text{473FTW}$ .

1. Prove that  $|w| = |w^R|$  for every string  $w$ .
2. Prove that  $(w \bullet z)^R = z^R \bullet w^R$  for all strings  $w$  and  $z$ .
3. Prove that  $(w^R)^R = w$  for every string  $w$ .

[Hint: You need #2 to prove #3, but you may find it easier to solve #3 first.]

**To think about later:** Let  $\#(a, w)$  denote the number of times symbol  $a$  appears in string  $w$ . For example,  $\#(\text{X}, \text{WTF374}) = 0$  and  $\#(\text{0}, \text{000010101010010100}) = 12$ .

4. Give a formal recursive definition of  $\#(a, w)$ .
5. Prove that  $\#(a, w \bullet z) = \#(a, w) + \#(a, z)$  for all symbols  $a$  and all strings  $w$  and  $z$ .
6. Prove that  $\#(a, w^R) = \#(a, w)$  for all symbols  $a$  and all strings  $w$ .

Give regular expressions for each of the following languages over the alphabet  $\{0, 1\}$ .

1. All strings containing the substring **000**.
2. All strings *not* containing the substring **000**.
3. All strings in which every run of **0**s has length at least 3.
4. All strings in which all the **1**s appear before any substring **000**.
5. All strings containing at least three **0**s.
6. Every string except **000**. [*Hint: Don't try to be clever.*]

**Work on these later:**

7. All strings  $w$  such that *in every prefix of  $w$* , the number of **0**s and **1**s differ by at most 1.
- \*8. All strings containing at least two **0**s and at least one **1**.
- \*9. All strings  $w$  such that *in every prefix of  $w$* , the number of **0**s and **1**s differ by at most 2.
- ★10. All strings in which the substring **000** appears an even number of times.  
(For example, **0001000** and **0000** are in this language, but **00000** is not.)

Describe deterministic finite-state automata that accept each of the following languages over the alphabet  $\Sigma = \{0, 1\}$ . Describe briefly what each state in your DFAs *means*.

Either drawings or formal descriptions are acceptable, as long as the states  $Q$ , the start state  $s$ , the accept states  $A$ , and the transition function  $\delta$  are all be clear. Try to keep the number of states small.

1. All strings containing the substring **000**.
2. All strings *not* containing the substring **000**.
3. All strings in which every run of **0**s has length at least 3.
4. All strings in which all the **1**s appear before any substring **000**.
5. All strings containing at least three **0**s.
6. Every string except **000**. [*Hint: Don't try to be clever.*]

**Work on these later:**

7. All strings  $w$  such that *in every prefix of  $w$* , the number of **0**s and **1**s differ by at most 1.
  8. All strings containing at least two **0**s and at least one **1**.
  9. All strings  $w$  such that *in every prefix of  $w$* , the number of **0**s and **1**s differ by at most 2.
- \*10. All strings in which the substring **000** appears an even number of times.  
(For example, **0001000** and **0000** are in this language, but **00000** is not.)

Describe deterministic finite-state automata that accept each of the following languages over the alphabet  $\Sigma = \{\texttt{0}, \texttt{1}\}$ . You may find it easier to describe these DFAs formally than to draw pictures.

Either drawings or formal descriptions are acceptable, as long as the states  $Q$ , the start state  $s$ , the accept states  $A$ , and the transition function  $\delta$  are all clear. Try to keep the number of states small.

1. All strings in which the number of **0**s is even and the number of **1**s is *not* divisible by 3.
2. All strings that are **both** the binary representation of an integer divisible by 3 **and** the ternary (base-3) representation of an integer divisible by 4.

For example, the string **1100** is an element of this language, because it represents  $2^3 + 2^2 = 12$  in binary and  $3^3 + 3^2 = 36$  in ternary.

#### Work on these later:

3. All strings  $w$  such that  $\binom{|w|}{2} \bmod 6 = 4$ .  
*[Hint: Maintain both  $\binom{|w|}{2} \bmod 6$  and  $|w| \bmod 6$ .]*  
*[Hint:  $\binom{n+1}{2} = \binom{n}{2} + n$ .]*
- \*4. All strings  $w$  such that  $F_{\#(\texttt{10}, w)} \bmod 10 = 4$ , where  $\#(\texttt{10}, w)$  denotes the number of times **10** appears as a substring of  $w$ , and  $F_n$  is the  $n$ th Fibonacci number:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

Prove that each of the following languages is *not* regular.

1.  $\{\text{0}^{2^n} \mid n \geq 0\}$
2.  $\{\text{0}^{2n}\text{1}^n \mid n \geq 0\}$
3.  $\{\text{0}^m\text{1}^n \mid m \neq 2n\}$
4. Strings over  $\{\text{0}, \text{1}\}$  where the number of **0**s is exactly twice the number of **1**s.
5. Strings of properly nested parentheses **( )**, brackets **[ ]**, and braces **{ }**. For example, the string **( [ ] { } )** is in this language, but the string **( [ ] )** is not, because the left and right delimiters don't match.

**Work on these later:**

6. Strings of the form  $w_1\#w_2\#\cdots\#w_n$  for some  $n \geq 2$ , where each substring  $w_i$  is a string in  $\{\text{0}, \text{1}\}^*$ , and some pair of substrings  $w_i$  and  $w_j$  are equal.
7.  $\{\text{0}^{n^2} \mid n \geq 0\}$
8.  $\{w \in (\text{0} + \text{1})^* \mid w \text{ is the binary representation of a perfect square}\}$

1. (a) Convert the regular expression  $(\mathbf{0}^*\mathbf{1} + \mathbf{0}\mathbf{1}^*)^*$  into an NFA using Thompson's algorithm.  
(b) Convert the NFA you just constructed into a DFA using the incremental subset construction. Draw the resulting DFA. Your DFA should have four states, all reachable from the start state. (Some of these states are obviously equivalent, but keep them separate.)  
(c) **Think about later:** Convert the DFA you just constructed into a regular expression using Han and Wood's algorithm. You should *not* get the same regular expression you started with.  
(d) What is this language?
  
2. (a) Convert the regular expression  $(\varepsilon + (\mathbf{0} + \mathbf{1}\mathbf{1})^*\mathbf{0})\mathbf{1}(\mathbf{1}\mathbf{1})^*$  into an NFA using Thompson's algorithm.  
(b) Convert the NFA you just constructed into a DFA using the incremental subset construction. Draw the resulting DFA. Your DFA should have six states, all reachable from the start state. (Some of these states are obviously equivalent, but keep them separate.)  
(c) **Think about later:** Convert the DFA you just constructed into a regular expression using Han and Wood's algorithm. You should *not* get the same regular expression you started with.  
(d) What is this language?

Let  $L$  be an arbitrary regular language.

1. Prove that the language  $\text{insert1}(L) := \{x\textcolor{red}{1}y \mid xy \in L\}$  is regular.

Intuitively,  $\text{insert1}(L)$  is the set of all strings that can be obtained from strings in  $L$  by inserting exactly one  $\textcolor{red}{1}$ . For example, if  $L = \{\epsilon, \textcolor{red}{00K!}\}$ , then  $\text{insert1}(L) = \{\textcolor{red}{1}, \textcolor{red}{100K!}, \textcolor{red}{010K!}, \textcolor{red}{001K!}, \textcolor{red}{00K!1}, \textcolor{red}{00K!1}\}$ .

2. Prove that the language  $\text{delete1}(L) := \{xy \mid x\textcolor{red}{1}y \in L\}$  is regular.

Intuitively,  $\text{delete1}(L)$  is the set of all strings that can be obtained from strings in  $L$  by deleting exactly one  $\textcolor{red}{1}$ . For example, if  $L = \{\textcolor{red}{101101}, \textcolor{red}{00}, \epsilon\}$ , then  $\text{delete1}(L) = \{\textcolor{red}{01101}, \textcolor{red}{10101}, \textcolor{red}{10110}\}$ .

---

**Work on these later:** (In fact, these might be easier than problems 1 and 2.)

3. Consider the following recursively defined function on strings:

$$\text{stutter}(w) := \begin{cases} \epsilon & \text{if } w = \epsilon \\ aa \bullet \text{stutter}(x) & \text{if } w = ax \text{ for some symbol } a \text{ and some string } x \end{cases}$$

Intuitively,  $\text{stutter}(w)$  doubles every symbol in  $w$ . For example:

- $\text{stutter}(\text{PRESTO}) = \text{PPRREESSTTOO}$
- $\text{stutter}(\text{HOCUS}\diamond\text{POCUS}) = \text{HHOOCCUUSS}\diamond\text{PPOOCCUUSS}$

Let  $L$  be an arbitrary regular language.

- (a) Prove that the language  $\text{stutter}^{-1}(L) := \{w \mid \text{stutter}(w) \in L\}$  is regular.
- (b) Prove that the language  $\text{stutter}(L) := \{\text{stutter}(w) \mid w \in L\}$  is regular.

4. Consider the following recursively defined function on strings:

$$\text{evens}(w) := \begin{cases} \epsilon & \text{if } w = \epsilon \\ \epsilon & \text{if } w = a \text{ for some symbol } a \\ b \cdot \text{evens}(x) & \text{if } w = abx \text{ for some symbols } a \text{ and } b \text{ and some string } x \end{cases}$$

Intuitively,  $\text{evens}(w)$  skips over every other symbol in  $w$ . For example:

- $\text{evens}(\text{EXPPELLIARMUS}) = \text{XELAMS}$
- $\text{evens}(\text{AVADA}\diamond\text{KEDAVRA}) = \text{VD}\diamond\text{EAR.}$

Once again, let  $L$  be an arbitrary regular language.

- (a) Prove that the language  $\text{evens}^{-1}(L) := \{w \mid \text{evens}(w) \in L\}$  is regular.
- (b) Prove that the language  $\text{evens}(L) := \{\text{evens}(w) \mid w \in L\}$  is regular.

Let  $L$  be an arbitrary regular language over the alphabet  $\Sigma = \{\textcolor{red}{0}, \textcolor{red}{1}\}$ . Prove that the following languages are also regular. (You probably won't get to all of these.)

1.  $\text{FLIPODDS}(L) := \{flipOdds(w) \mid w \in L\}$ , where the function  $flipOdds$  inverts every odd-indexed bit in  $w$ . For example:

$$flipOdds(\textcolor{red}{0}00011101010101) = \textcolor{red}{1}01001011111111$$

**Solution:** Let  $M = (Q, s, A, \delta)$  be a DFA that accepts  $L$ . We construct a new DFA  $M' = (Q', s', A', \delta')$  that accepts  $\text{FLIPODDS}(L)$  as follows.

Intuitively,  $M'$  receives some string  $flipOdds(w)$  as input, restores every other bit to obtain  $w$ , and simulates  $M$  on the restored string  $w$ .

Each state  $(q, flip)$  of  $M'$  indicates that  $M$  is in state  $q$ , and we need to flip the next input bit if  $flip = \text{TRUE}$

$$Q' = Q \times \{\text{TRUE}, \text{FALSE}\}$$

$$s' = (s, \text{TRUE})$$

$$A' =$$

$$\delta'((q, flip), a) =$$

■

2.  $\text{UNFLIPODD1S}(L) := \{w \in \Sigma^* \mid flipOdd1s(w) \in L\}$ , where the function  $flipOdd1s$  inverts every other  $\textcolor{red}{1}$  bit of its input string, starting with the first  $\textcolor{red}{1}$ . For example:

$$flipOdd1s(\textcolor{red}{0}0001\underline{1}1101010101) = \textcolor{red}{0}000\underline{0}10100010001$$

**Solution:** Let  $M = (Q, s, A, \delta)$  be a DFA that accepts  $L$ . We construct a new DFA  $M' = (Q', s', A', \delta')$  that accepts  $\text{UNFLIPODD1S}(L)$  as follows.

Intuitively,  $M'$  receives some string  $w$  as input, flips every other  $\textcolor{red}{1}$  bit, and simulates  $M$  on the transformed string.

Each state  $(q, flip)$  of  $M'$  indicates that  $M$  is in state  $q$ , and we need to flip the next  $\textcolor{red}{1}$  bit of and only if  $flip = \text{TRUE}$ .

$$Q' = Q \times \{\text{TRUE}, \text{FALSE}\}$$

$$s' = (s, \text{TRUE})$$

$$A' =$$

$$\delta'((q, flip), a) =$$

■

3.  $\text{FLIPODD}\mathbf{1s}(L) := \{\text{flipOdd}\mathbf{1s}(w) \mid w \in L\}$ , where the function  $\text{flipOdd}\mathbf{1}$  is defined as in the previous problem.

**Solution:** Let  $M = (Q, s, A, \delta)$  be a DFA that accepts  $L$ . We construct a new NFA  $M' = (Q', s', A', \delta')$  that accepts  $\text{FLIPODD}\mathbf{1s}(L)$  as follows.

Intuitively,  $M'$  receives some string  $\text{flipOdd}\mathbf{1s}(w)$  as input, **guesses** which  $\mathbf{0}$  bits to restore to  $\mathbf{1s}$ , and simulates  $M$  on the restored string  $w$ . No string in  $\text{FLIPODD}\mathbf{1s}(L)$  has two  $\mathbf{1s}$  in a row, so if  $M'$  ever sees  $\mathbf{11}$ , it rejects.

Each state  $(q, \text{flip})$  of  $M'$  indicates that  $M$  is in state  $q$ , and we need to flip a  $\mathbf{0}$  bit before the next  $\mathbf{1}$  if  $\text{flip} = \text{TRUE}$ .

$$Q' = Q \times \{\text{TRUE}, \text{FALSE}\}$$

$$s' = (s, \text{TRUE})$$

$$A' =$$

$$\delta'((q, \text{flip}), a) =$$

■

4.  $\text{FARO}(L) := \{\text{faro}(w, x) \mid w, x \in L \text{ and } |w| = |x|\}$ , where the function  $\text{faro}$  is defined recursively as follows:

$$\text{faro}(w, x) := \begin{cases} x & \text{if } w = \epsilon \\ a \cdot \text{faro}(x, y) & \text{if } w = ay \text{ for some } a \in \Sigma \text{ and some } y \in \Sigma^* \end{cases}$$

For example,  $\text{faro}(\mathbf{0001101}, \mathbf{1111001}) = \mathbf{0101011100011}$ . (A "faro shuffle" splits a deck of cards into two equal piles and then perfectly interleaves them.)

**Solution:** Let  $M = (Q, s, A, \delta)$  be a DFA that accepts  $L$ . We construct a DFA  $M' = (Q', s', A', \delta')$  that accepts  $\text{FARO}(L)$  as follows.

Intuitively,  $M'$  reads the string  $\text{faro}(w, x)$  as input, splits the string into the subsequences  $w$  and  $x$ , and passes each of those strings to an independent copy of  $M$ .

Each state  $(q_1, q_2, \text{next})$  indicates that the copy of  $M$  that gets  $w$  is in state  $q_1$ , the copy of  $M$  that gets  $x$  is in state  $q_2$ , and  $\text{next}$  indicates which copy gets the next input bit.

$$Q' = Q \times Q \times \{1, 2\}$$

$$s' = (s, s, 1)$$

$$A' =$$

$$\delta'((q_1, q_2, \text{next}), a) =$$

■

Here are several problems that are easy to solve in  $O(n)$  time, essentially by brute force. Your task is to design algorithms for these problems that are significantly faster.

1. Suppose we are given an array  $A[1..n]$  of  $n$  distinct integers, which could be positive, negative, or zero, sorted in increasing order so that  $A[1] < A[2] < \dots < A[n]$ .
  - (a) Describe a fast algorithm that either computes an index  $i$  such that  $A[i] = i$  or correctly reports that no such index exists.
  - (b) Suppose we know in advance that  $A[1] > 0$ . Describe an even faster algorithm that either computes an index  $i$  such that  $A[i] = i$  or correctly reports that no such index exists. [Hint: This is **really** easy.]
2. Suppose we are given an array  $A[1..n]$  such that  $A[1] \geq A[2]$  and  $A[n-1] \leq A[n]$ . We say that an element  $A[x]$  is a **local minimum** if both  $A[x-1] \geq A[x]$  and  $A[x] \leq A[x+1]$ . For example, there are exactly six local minima in the following array:

9	7	7	2	1	3	7	5	4	7	3	3	4	8	6	9
	▲			▲				▲		▲					▲

Describe and analyze a fast algorithm that returns the index of one local minimum. For example, given the array above, your algorithm could return the integer 9, because  $A[9]$  is a local minimum. [Hint: With the given boundary conditions, any array **must** contain at least one local minimum. Why?]

3. Suppose you are given two sorted arrays  $A[1..n]$  and  $B[1..n]$  containing distinct integers. Describe a fast algorithm to find the median (meaning the  $n$ th smallest element) of the union  $A \cup B$ . For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \quad B[1..8] = [2, 4, 5, 8, 17, 19, 21, 23]$$

your algorithm should return the integer 9. [Hint: What can you learn by comparing one element of  $A$  with one element of  $B$ ?]

**To think about later:**

4. Now suppose you are given two sorted arrays  $A[1..m]$  and  $B[1..n]$  and an integer  $k$ . Describe a fast algorithm to find the  $k$ th smallest element in the union  $A \cup B$ . For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \quad B[1..5] = [2, 5, 7, 17, 19] \quad k = 6$$

your algorithm should return the integer 7.

In lecture, Jeff described an algorithm of Karatsuba that multiplies two  $n$ -digit integers using  $O(n^{\lg 3})$  single-digit additions, subtractions, and multiplications. In this lab we'll look at some extensions and applications of this algorithm.

1. Describe an algorithm to compute the product of an  $n$ -digit number and an  $m$ -digit number, where  $m < n$ , in  $O(m^{\lg 3 - 1} n)$  time.
2. Describe an algorithm to compute the decimal representation of  $2^n$  in  $O(n^{\lg 3})$  time.

*[Hint: Repeated squaring. The standard algorithm that computes one decimal digit at a time requires  $\Theta(n^2)$  time.]*

3. Describe a divide-and-conquer algorithm to compute the decimal representation of an arbitrary  $n$ -bit binary number in  $O(n^{\lg 3})$  time.

*[Hint: Let  $x = a \cdot 2^{n/2} + b$ . Watch out for an extra log factor in the running time.]*

**Think about later:**

4. Suppose we can multiply two  $n$ -digit numbers in  $O(M(n))$  time. Describe an algorithm to compute the decimal representation of an arbitrary  $n$ -bit binary number in  $O(M(n) \log n)$  time.

A **subsequence** of a sequence (for example, an array, linked list, or string), obtained by removing zero or more elements and keeping the rest in the same sequence order. A subsequence is called a **substring** if its elements are contiguous in the original sequence. For example:

- **SUBSEQUENCE**, **UBSEQU**, and the empty string  $\varepsilon$  are all substrings (and therefore subsequences) of the string **SUBSEQUENCE**;
- **SBSQNC**, **SQUEE**, and **EEE** are all subsequences of **SUBSEQUENCE** but not substrings;
- **QUEUE**, **EQUUS**, and **DIMAGGIO** are not subsequences (and therefore not substrings) of **SUBSEQUENCE**.

Describe recursive backtracking algorithms for the following problems. *Don't worry about running times.*

- Given an array  $A[1..n]$  of integers, compute the length of a **longest increasing subsequence**. A sequence  $B[1..\ell]$  is *increasing* if  $B[i] > B[i - 1]$  for every index  $i \geq 2$ .

For example, given the array

$$\langle 3, \underline{1}, \underline{4}, 1, \underline{5}, 9, 2, \underline{6}, 5, 3, 5, \underline{8}, \underline{9}, 7, 9, 3, 2, 3, 8, 4, 6, 2, 7 \rangle$$

your algorithm should return the integer 6, because  $\langle 1, 4, 5, 6, 8, 9 \rangle$  is a longest increasing subsequence (one of many).

- Given an array  $A[1..n]$  of integers, compute the length of a **longest decreasing subsequence**. A sequence  $B[1..\ell]$  is *decreasing* if  $B[i] < B[i - 1]$  for every index  $i \geq 2$ .

For example, given the array

$$\langle 3, 1, 4, 1, 5, \underline{9}, 2, \underline{6}, 5, 3, \underline{5}, 8, 9, 7, 9, 3, 2, 3, 8, \underline{4}, 6, \underline{2}, 7 \rangle$$

your algorithm should return the integer 5, because  $\langle 9, 6, 5, 4, 2 \rangle$  is a longest decreasing subsequence (one of many).

- Given an array  $A[1..n]$  of integers, compute the length of a **longest alternating subsequence**. A sequence  $B[1..\ell]$  is *alternating* if  $B[i] < B[i - 1]$  for every even index  $i \geq 2$ , and  $B[i] > B[i - 1]$  for every odd index  $i \geq 3$ .

For example, given the array

$$\langle \underline{3}, \underline{1}, \underline{4}, \underline{1}, \underline{5}, 9, \underline{2}, \underline{6}, \underline{5}, 3, 5, \underline{8}, 9, \underline{7}, \underline{9}, \underline{3}, 2, 3, \underline{8}, \underline{4}, \underline{6}, \underline{2}, \underline{7} \rangle$$

your algorithm should return the integer 17, because  $\langle 3, 1, 4, 1, 5, 2, 6, 5, 8, 7, 9, 3, 8, 4, 6, 2, 7 \rangle$  is a longest alternating subsequence (one of many).

**To think about later:**

4. Given an array  $A[1..n]$  of integers, compute the length of a longest **convex** subsequence of  $A$ . A sequence  $B[1..\ell]$  is *convex* if  $B[i] - B[i-1] > B[i-1] - B[i-2]$  for every index  $i \geq 3$ .

For example, given the array

$$\langle \underline{3}, \underline{1}, 4, \underline{1}, 5, 9, \underline{2}, 6, 5, 3, \underline{5}, 8, \underline{9}, 7, 9, 3, 2, 3, 8, 4, 6, 2, 7 \rangle$$

your algorithm should return the integer 6, because  $\langle 3, 1, 1, 2, 5, 9 \rangle$  is a longest convex subsequence (one of many).

5. Given an array  $A[1..n]$ , compute the length of a longest **palindrome** subsequence of  $A$ . Recall that a sequence  $B[1..\ell]$  is a *palindrome* if  $B[i] = B[\ell - i + 1]$  for every index  $i$ .

For example, given the array

$$\langle 3, 1, \underline{4}, 1, 5, \underline{9}, 2, 6, \underline{5}, \underline{3}, \underline{5}, 8, 9, 7, \underline{9}, 3, 2, 3, 8, \underline{4}, 6, 2, 7 \rangle$$

your algorithm should return the integer 7, because  $\langle 4, 9, 5, 3, 5, 9, 4 \rangle$  is a longest palindrome subsequence (one of many).

A *subsequence* of a sequence (for example, an array, a linked list, or a string), obtained by removing zero or more elements and keeping the rest in the same sequence order. A subsequence is called a *substring* if its elements are contiguous in the original sequence. For example:

- **SUBSEQUENCE, UBSEQU**, and the empty string  $\epsilon$  are all substrings of the string **SUBSEQUENCE**;
  - **SBSQNC, UEQUE**, and **EEE** are all subsequences of **SUBSEQUENCE** but not substrings;
  - **QUEUE, SSS**, and **FOOBAR** are not subsequences of **SUBSEQUENCE**.
- 

Describe and analyze *dynamic programming* algorithms for the following problems. For the first three, use the backtracking algorithms you developed on Wednesday.

1. Given an array  $A[1..n]$  of integers, compute the length of a longest *increasing* subsequence of  $A$ . A sequence  $B[1..\ell]$  is *increasing* if  $B[i] > B[i-1]$  for every index  $i \geq 2$ .
2. Given an array  $A[1..n]$  of integers, compute the length of a longest *decreasing* subsequence of  $A$ . A sequence  $B[1..\ell]$  is *decreasing* if  $B[i] < B[i-1]$  for every index  $i \geq 2$ .
3. Given an array  $A[1..n]$  of integers, compute the length of a longest *alternating* subsequence of  $A$ . A sequence  $B[1..\ell]$  is *alternating* if  $B[i] < B[i-1]$  for every even index  $i \geq 2$ , and  $B[i] > B[i-1]$  for every odd index  $i \geq 3$ .
4. Given an array  $A[1..n]$  of integers, compute the length of a longest *convex* subsequence of  $A$ . A sequence  $B[1..\ell]$  is *convex* if  $B[i] - B[i-1] > B[i-1] - B[i-2]$  for every index  $i \geq 3$ .
5. Given an array  $A[1..n]$ , compute the length of a longest *palindrome* subsequence of  $A$ . Recall that a sequence  $B[1..\ell]$  is a *palindrome* if  $B[i] = B[\ell-i+1]$  for every index  $i$ .

## Basic steps in developing a dynamic programming algorithm

1. **Formulate the problem recursively.** This is the hard part. There are two distinct but equally important things to include in your formulation.
  - (a) **Specification.** First, give a clear and precise English description of the problem you are claiming to solve. Not *how* to solve the problem, but *what* the problem actually is. Omitting this step in homeworks or exams is an automatic zero.
  - (b) **Solution.** Second, give a clear recursive formula or algorithm for the whole problem in terms of the answers to smaller instances of *exactly* the same problem. It generally helps to think in terms of a recursive definition of your inputs and outputs. If you discover that you need a solution to a *similar* problem, or a slightly *related* problem, you're attacking the wrong problem; go back to step 1.
2. **Build solutions to your recurrence from the bottom up.** Write an algorithm that starts with the base cases of your recurrence and works its way up to the final solution, by considering intermediate subproblems in the correct order. This stage can be broken down into several smaller, relatively mechanical steps:
  - (a) **Identify the subproblems.** What are all the different ways can your recursive algorithm call itself, starting with some initial input?
  - (b) **Analyze running time.** Add up the running times of all possible subproblems, *ignoring the recursive calls*.
  - (c) **Choose a memoization data structure.** For most problems, each recursive subproblem can be identified by a few integers, so you can use a multidimensional array. But some problems need a more complicated data structure.
  - (d) **Identify dependencies.** Except for the base cases, every recursive subproblem depends on other subproblems—which ones? Draw a picture of your data structure, pick a generic element, and draw arrows from each of the other elements it depends on. Then formalize your picture.
  - (e) **Find a good evaluation order.** Order the subproblems so that each subproblem comes *after* the subproblems it depends on. Typically, you should consider the base cases first, then the subproblems that depends only on base cases, and so on. *Be careful!*
  - (f) **Write down the algorithm.** You know what order to consider the subproblems, and you know how to solve each subproblem. So do that! If your data structure is an array, this usually means writing a few nested for-loops around your original recurrence.

Lenny Adve, the founding dean of the new Maximilian Q. Levchin College of Computer Science, has commissioned a series of snow ramps on the south slope of the Orchard Downs sledding hill<sup>1</sup> and challenged Bill Kudeki, head of the Department of Electrical and Computer Engineering, to a sledding contest. Bill and Lenny will both sled down the hill, each trying to maximize their air time. The winner gets to expand their department/college into Siebel Center, the new ECE Building, *and* the English Building; the loser has to move their entire department/college under the Boneyard bridge next to Everitt Lab (along with the English department).

Whenever Lenny or Bill reaches a ramp *while on the ground*, they can either use that ramp to jump through the air, possibly flying over one or more ramps, or sled past that ramp and stay on the ground. Obviously, if someone flies over a ramp, they cannot use that ramp to extend their jump.

1. Suppose you are given a pair of arrays  $Ramp[1..n]$  and  $Length[1..n]$ , where  $Ramp[i]$  is the distance from the top of the hill to the  $i$ th ramp, and  $Length[i]$  is the distance that any sledder who takes the  $i$ th ramp will travel through the air.

Describe and analyze an algorithm to determine the maximum *total* distance that Lenny and Bill can travel through the air. [*Hint: Do whatever you **feel** like you wanna do. Gosh!*]

2. Uh-oh. The university lawyers heard about Lenny and Bill's little bet and immediately objected. To protect the university from both lawsuits and sky-rocketing insurance rates, they impose an upper bound on the number of jumps that either sledder can take.

Describe and analyze an algorithm to determine the maximum total distance that Lenny or Bill can spend in the air *with at most  $k$  jumps*, given the original arrays  $Ramp[1..n]$  and  $Length[1..n]$  and the integer  $k$  as input.

3. **To think about later:** When the lawyers realized that imposing their restriction didn't immediately shut down the contest, they added a new restriction: No ramp can be used more than once! Disgusted by the legal interference, Lenny and Bill give up on their bet and decide to cooperate to put on a good show for the spectators.

Describe and analyze an algorithm to determine the maximum total distance that Lenny and Bill can spend in the air, each taking at most  $k$  jumps (so at most  $2k$  jumps total), and with each ramp used at most once.

---

<sup>1</sup>The north slope is faster, but too short for an interesting contest.

1. A **basic arithmetic expression** is composed of characters from the set  $\{1, +, \times\}$  and parentheses. Almost every integer can be represented by more than one basic arithmetic expression. For example, all of the following basic arithmetic expression represent the integer 14:

$$\begin{aligned} & 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 \\ & ((1 + 1) \times (1 + 1 + 1 + 1 + 1)) + ((1 + 1) \times (1 + 1)) \\ & (1 + 1) \times (1 + 1 + 1 + 1 + 1 + 1 + 1) \\ & (1 + 1) \times (((1 + 1 + 1) \times (1 + 1)) + 1) \end{aligned}$$

Describe and analyze an algorithm to compute, given an integer  $n$  as input, the minimum number of 1's in a basic arithmetic expression whose value is equal to  $n$ . The number of parentheses doesn't matter, just the number of 1's. For example, when  $n = 14$ , your algorithm should return 8, for the final expression above. The running time of your algorithm should be bounded by a small polynomial function of  $n$ .

#### Think about later:

2. Suppose you are given a sequence of integers separated by  $+$  and  $-$  signs; for example:

$$1 + 3 - 2 - 5 + 1 - 6 + 7$$

You can change the value of this expression by adding parentheses in different places. For example:

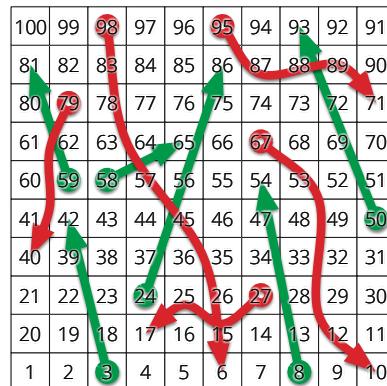
$$\begin{aligned} & 1 + 3 - 2 - 5 + 1 - 6 + 7 = -1 \\ & (1 + 3 - (2 - 5)) + (1 - 6) + 7 = 9 \\ & (1 + (3 - 2)) - (5 + 1) - (6 + 7) = -17 \end{aligned}$$

Describe and analyze an algorithm to compute, given a list of integers separated by  $+$  and  $-$  signs, the maximum possible value the expression can take by adding parentheses. Parentheses must be used only to group additions and subtractions; in particular, do not use them to create implicit multiplication as in  $1 + 3(-2)(-5) + 1 - 6 + 7 = 33$ .

For each of the problems below, transform the input into a graph and apply a standard graph algorithm that you've seen in class. Whenever you use a standard graph algorithm, you **must** provide the following information. (I recommend actually using a bulleted list.)

- What are the vertices? What does each vertex represent?
- What are the edges? Are they directed or undirected?
- If the vertices and/or edges have associated values, what are they?
- What problem do you need to solve on this graph?
- What standard algorithm are you using to solve that problem?
- What is the running time of your entire algorithm, *including* the time to build the graph, as a function of the original input parameters?

1. **Snakes and Ladders** is a classic board game, originating in India no later than the 16th century. The board consists of an  $n \times n$  grid of squares, numbered consecutively from 1 to  $n^2$ , starting in the bottom left corner and proceeding row by row from bottom to top, with rows alternating to the left and right. Certain pairs of squares, always in different rows, are connected by either “snakes” (leading down) or “ladders” (leading up). *Each square can be an endpoint of at most one snake or ladder.*



A typical Snakes and Ladders board.

Upward straight arrows are ladders; downward wavy arrows are snakes.

You start with a token in cell 1, in the bottom left corner. In each move, you advance your token up to  $k$  positions, for some fixed constant  $k$  (typically 6). Then if the token is at the *top* of a snake, you **must** slide the token down to the bottom of that snake, and if the token is at the *bottom* of a ladder, you **may** move the token up to the top of that ladder.

Describe and analyze an efficient algorithm to compute the smallest number of moves required for the token to reach the last square of the Snakes and Ladders board.

2. Let  $G$  be an undirected graph. Suppose we start with two coins on two arbitrarily chosen vertices of  $G$ . At every step, each coin **must** move to an adjacent vertex. Describe and analyze an efficient algorithm to compute the minimum number of steps to reach a configuration where both coins are on the same vertex, or to report correctly that no such configuration is reachable. The input to your algorithm consists of a graph  $G = (V, E)$  and two vertices  $u, v \in V$  (which may or may not be distinct).

Think about later:

3. Let  $G$  be an undirected graph. Suppose we start with 374 coins on 374 arbitrarily chosen vertices of  $G$ . At every step, each coin **must** move to an adjacent vertex. Describe and analyze an efficient algorithm to compute the minimum number of steps to reach a configuration where both coins are on the same vertex, or to report correctly that no such configuration is reachable. The input to your algorithm consists of a graph  $G = (V, E)$  and starting vertices  $s_1, s_2, \dots, s_{374}$  (which may or may not be distinct).

For each of the problems below, transform the input into a graph and apply a standard graph algorithm that you've seen in class. Whenever you use a standard graph algorithm, you *must* provide the following information. (I recommend actually using a bulleted list.)

- What are the vertices? What does each vertex represent?
  - What are the edges? Are they directed or undirected?
  - If the vertices and/or edges have associated values, what are they?
  - What problem do you need to solve on this graph?
  - What standard algorithm are you using to solve that problem?
  - What is the running time of your entire algorithm, *including* the time to build the graph, as a function of the original input parameters?
- 

1. Inspired by the previous lab, you decide to organize a Snakes and Ladders competition with  $n$  participants. In this competition, each game of Snakes and Ladders involves three players. After the game is finished, they are ranked first, second, and third. Each player may be involved in any (non-negative) number of games, and the number need not be equal among players.

At the end of the competition,  $m$  games have been played. You realize that you forgot to implement a proper rating system, and therefore decide to produce the overall ranking of all  $n$  players as you see fit. However, to avoid being too suspicious, if player  $A$  ranked better than player  $B$  in any game, then  $A$  must rank better than  $B$  in the overall ranking.

You are given the list of players and their ranking in each of the  $m$  games. Describe and analyze an algorithm that produces an overall ranking of the  $n$  players that is consistent with the individual game rankings, or correctly reports that no such ranking exists.

2. There are  $n$  galaxies connected by  $m$  intergalactic teleport-ways. Each teleport-way joins two galaxies and can be traversed in both directions. However, the company that runs the teleport-ways has established an extremely lucrative cost structure: Anyone can teleport *further* from their home galaxy at no cost whatsoever, but teleporting *toward* their home galaxy is prohibitively expensive.

Judy has decided to take a sabbatical tour of the universe by visiting as many galaxies as possible, starting at her home galaxy. To save on travel expenses, she wants to teleport away from her home galaxy at every step, except for the very last teleport home.

Describe and analyze an algorithm to compute the maximum number of galaxies that Judy can visit. Your input consists of an undirected graph  $G$  with  $n$  vertices and  $m$  edges describing the teleport-way network, an integer  $1 \leq s \leq n$  identifying Judy's home galaxy, and an array  $D[1..n]$  containing the distances of each galaxy from  $s$ .

#### To think about later:

3. Just before embarking on her universal tour, Judy wins the space lottery, giving her just enough money to afford two teleports toward her home galaxy. Describe and analyze a new algorithm to compute the maximum number of galaxies Judy can visit; if she visits the same galaxy twice, that counts as two visits. After all, argues the travel agent, who can see an entire galaxy in just one visit?

- \*4. Judy replies angrily to the travel agent that *she* can see an entire galaxy in just one visit, because 99% of every galaxy is exactly the same glowing balls of plasma and lifeless chunks of rock and McDonalds and Starbucks and prefab “Irish” pubs and overpriced souvenir shops and Peruvian street-corner musicians as every other galaxy.

Describe and analyze an algorithm to compute the maximum number of *distinct* galaxies Judy can visit. She is still *allowed* to visit the same galaxy more than once, but only the first visit counts toward her total.

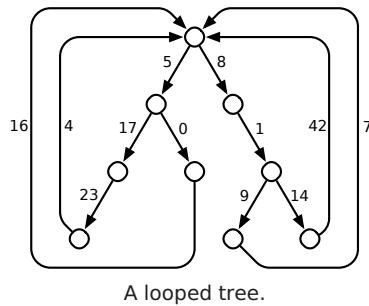
1. Describe and analyze an algorithm to compute the shortest path from vertex  $s$  to vertex  $t$  in a directed graph with weighted edges, where exactly *one* edge  $u \rightarrow v$  has negative weight. Assume the graph has no negative cycles. [Hint: Modify the input graph and run Dijkstra's algorithm. Alternatively, **don't** modify the input graph, but run Dijkstra's algorithm anyway.]
2. You just discovered your best friend from elementary school on Twitbook. You both want to meet as soon as possible, but you live in two different cities that are far apart. To minimize travel time, you agree to meet at an intermediate city, and then you simultaneously hop in your cars and start driving toward each other. But where *exactly* should you meet?

You are given a weighted graph  $G = (V, E)$ , where the vertices  $V$  represent cities and the edges  $E$  represent roads that directly connect cities. Each edge  $e$  has a weight  $w(e)$  equal to the time required to travel between the two cities. You are also given a vertex  $p$ , representing your starting location, and a vertex  $q$ , representing your friend's starting location.

Describe and analyze an algorithm to find the target vertex  $t$  that allows you and your friend to meet as soon as possible, assuming both of you leave home *right now*.

#### To think about later:

3. A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has a non-negative weight.



- (a) How much time would Dijkstra's algorithm require to compute the shortest path between two vertices  $u$  and  $v$  in a looped tree with  $n$  nodes?
- (b) Describe and analyze a faster algorithm.

- Suppose that you have just finished computing the array  $dist[1..V, 1..V]$  of shortest-path distances between *all* pairs of vertices in an edge-weighted directed graph  $G$ . Unfortunately, you discover that you incorrectly entered the weight of a single edge  $u \rightarrow v$ , so all that precious CPU time was wasted. Or was it? Maybe your distances are correct after all!

In each of the following problems, let  $w(u \rightarrow v)$  denote the weight that you used in your distance computation, and let  $w'(u \rightarrow v)$  denote the correct weight of  $u \rightarrow v$ .

- Suppose  $w(u \rightarrow v) > w'(u \rightarrow v)$ ; that is, the weight you used for  $u \rightarrow v$  was *larger* than its true weight. Describe an algorithm that repairs the distance array in  $O(V^2)$  time under this assumption. [Hint: For every pair of vertices  $x$  and  $y$ , either  $u \rightarrow v$  is on the shortest path from  $x$  to  $y$  or it isn't.]
  - Maybe even that was too much work. Describe an algorithm that determines whether your original distance array is actually correct in  $O(1)$  time, again assuming that  $w(u \rightarrow v) > w'(u \rightarrow v)$ . [Hint: Either  $u \rightarrow v$  is the shortest path from  $u$  to  $v$  or it isn't.]
  - To think about later:** Describe an algorithm that determines in  $O(VE)$  time whether your distance array is actually correct, even if  $w(u \rightarrow v) < w'(u \rightarrow v)$ .
  - To think about later:** Argue that when  $w(u \rightarrow v) < w'(u \rightarrow v)$ , repairing the distance array *requires* recomputing shortest paths from scratch, at least in the worst case.
- You—yes, *you*—can cause a major economic collapse with the power of graph algorithms!<sup>1</sup> The *arbitrage* business is a money-making scheme that takes advantage of differences in currency exchange. In particular, suppose that 1 US dollar buys 120 Japanese yen; 1 yen buys 0.01 euros; and 1 euro buys 1.2 US dollars. Then, a trader starting with \$1 can convert their money from dollars to yen, then from yen to euros, and finally from euros back to dollars, ending with \$1.44! The cycle of currencies  $\$ \rightarrow ¥ \rightarrow € \rightarrow \$$  is called an *arbitrage cycle*. Of course, finding and exploiting arbitrage cycles before the prices are corrected requires extremely fast algorithms.

Suppose  $n$  different currencies are traded in your currency market. You are given the matrix  $R[1..n]$  of exchange rates between every pair of currencies; for each  $i$  and  $j$ , one unit of currency  $i$  can be traded for  $R[i, j]$  units of currency  $j$ . (Do *not* assume that  $R[i, j] \cdot R[j, i] = 1$ .)

- Describe an algorithm that returns an array  $V[1..n]$ , where  $V[i]$  is the maximum amount of currency  $i$  that you can obtain by trading, starting with one unit of currency 1, assuming there are no arbitrage cycles.
- Describe an algorithm to determine whether the given matrix of currency exchange rates creates an arbitrage cycle.
- To think about later:** Modify your algorithm from part (b) to actually return an arbitrage cycle, if such a cycle exists.

---

<sup>1</sup>No, you can't.

1. *Flappy Bird* is a popular mobile game written by Nguyễn Hà Đông, originally released in May 2013. The game features a bird named “Faby”, who flies to the right at constant speed. Whenever the player taps the screen, Faby is given a fixed upward velocity; between taps, Faby falls due to gravity. Faby flies through a landscape of pipes until it touches either a pipe or the ground, at which point the game is over. Your task, should you choose to accept it, is to develop an algorithm to play Flappy Bird automatically.

Well, okay, not Flappy Bird exactly, but the following drastically simplified variant, which I will call *Flappy Pixel*. Instead of a bird, Faby is a single point, specified by three integers: horizontal position  $x$  (in pixels), vertical position  $y$  (in pixels), and vertical speed  $y'$  (in pixels per frame). Faby’s environment is described by two arrays  $Hi[1..n]$  and  $Lo[1..n]$ , where for each index  $i$ , we have  $0 < Lo[i] < Hi[i] < h$  for some fixed height value  $h$ . The game is described by the following piece of pseudocode:

```

FLAPPYPIXEL( $Hi[1..n]$ ,  $Lo[1..n]$ ):
     $y \leftarrow \lceil h/2 \rceil$ 
     $y' \leftarrow 0$ 
    for  $x \leftarrow 1$  to  $n$ 
        if the player taps the screen
             $y' \leftarrow 10$            $\langle\langle flap \rangle\rangle$ 
        else
             $y' \leftarrow y' - 1$        $\langle\langle fall \rangle\rangle$ 
         $y \leftarrow y + y'$ 
        if  $y < Lo[x]$  or  $y > Hi[x]$ 
            return FALSE            $\langle\langle player loses \rangle\rangle$ 
        return TRUE               $\langle\langle player wins \rangle\rangle$ 
```

Notice that in each iteration of the main loop, the player has the option of tapping the screen.

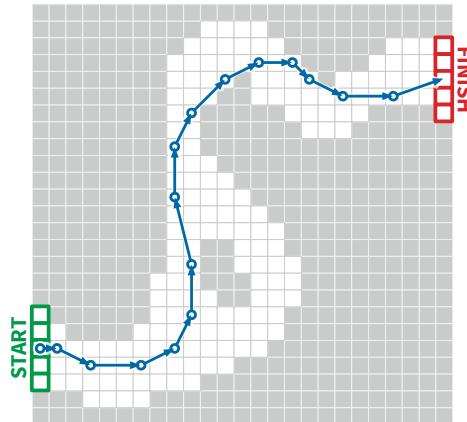
Describe and analyze an algorithm to determine the minimum number of times that the player must tap the screen to win Flappy Pixel, given the integer  $h$  and the arrays  $Hi[1..n]$  and  $Lo[1..n]$  as input. If the game cannot be won at all, your algorithm should return  $\infty$ . Describe the running time of your algorithm as a function of  $n$  and  $h$ .

*[Problem 2 is on the back.]*

2. **Racetrack** (also known as *Graph Racers* and *Vector Rally*) is a two-player paper-and-pencil racing game that Jeff played on the bus in 5th grade.<sup>1</sup> The game is played with a track drawn on a sheet of graph paper. The players alternately choose a sequence of grid points that represent the motion of a car around the track, subject to certain constraints explained below.

Each car has a *position* and a *velocity*, both with integer *x*- and *y*-coordinates. A subset of grid squares is marked as the *starting area*, and another subset is marked as the *finishing area*. The initial position of each car is chosen by the player somewhere in the starting area; the initial velocity of each car is always  $(0, 0)$ . At each step, the player optionally changes each component of the velocity by at most 1. The car's new position is then determined by adding the new velocity to the car's previous position. The new position must be inside the track; otherwise, the car crashes and that player loses the race.<sup>2</sup> The race ends when the first car reaches a position inside the finishing area.

velocity	position
$(0, 0)$	$(1, 5)$
$(1, 0)$	$(2, 5)$
$(2, -1)$	$(4, 4)$
$(3, 0)$	$(7, 4)$
$(2, 1)$	$(9, 5)$
$(1, 2)$	$(10, 7)$
$(0, 3)$	$(10, 10)$
$(-1, 4)$	$(9, 14)$
$(0, 3)$	$(9, 17)$
$(1, 2)$	$(10, 19)$
$(2, 2)$	$(12, 21)$
$(2, 1)$	$(14, 22)$
$(2, 0)$	$(16, 22)$
$(1, -1)$	$(17, 21)$
$(2, -1)$	$(19, 20)$
$(3, 0)$	$(22, 20)$
$(3, 1)$	$(25, 21)$



A 16-step Racetrack run, on a  $25 \times 25$  track. This is *not* the shortest run on this track.

Suppose the racetrack is represented by an  $n \times n$  array of bits, where each  $\textcolor{red}{0}$  bit represents a grid point inside the track, each  $\textcolor{red}{1}$  bit represents a grid point outside the track, the “starting line” consists of all  $\textcolor{red}{0}$  bits in column 1, and the “finishing line” consists of all  $\textcolor{red}{0}$  bits in column  $n$ .

Describe and analyze an algorithm to find the minimum number of steps required to move a car from the starting line to the finish line of a given racetrack.

[Hint: Your initial analysis can be improved.]

<sup>1</sup>The actual game is a bit more complicated than the version described here. See <http://harmmade.com/vectorracer/> for an excellent online version.

<sup>2</sup>However, it is not necessary for the entire line segment between the old position and the new position to lie inside the track. Sometimes Speed Racer has to push the A button.

### To think about later:

3. Consider the following variant of Flappy Pixel. The mechanics of the game are unchanged, but now the environment is specified by an array  $Points[1..n, 1..h]$  of integers, which could be positive, negative, or zero. If Faby falls off the top or bottom edge of the environment, the game immediately ends and the player gets nothing. Otherwise, at each frame, the player earns  $Points[x, y]$  points, where  $(x, y)$  is Faby's current position. The game ends when Faby reaches the right end of the environment.

```
FLAPPYPIXEL2( $Points[1..n]$ ):
    score  $\leftarrow 0$ 
     $y \leftarrow \lceil h/2 \rceil$ 
     $y' \leftarrow 0$ 
    for  $x \leftarrow 1$  to  $n$ 
        if the player taps the screen
             $y' \leftarrow 10$            ⟨⟨flap⟩⟩
        else
             $y' \leftarrow y' - 1$        ⟨⟨fall⟩⟩
         $y \leftarrow y + y'$ 
        if  $y < 1$  or  $y > h$ 
            return  $-\infty$           ⟨⟨fail⟩⟩
        score  $\leftarrow score + Points[x, y]$ 
    return score
```

Describe and analyze an algorithm to determine the maximum possible score that a player can earn in this game.

4. We can also consider a similar variant of Racetrack. Instead of bits, the “track” is described by an array  $Points[1..n, 1..n]$  of *numbers*, which could be positive, negative, or zero. Whenever the car lands on a grid cell  $(i, j)$ , the player receives  $Points[i, j]$  points. Forbidden grid cells are indicated by  $Points[i, j] = -\infty$ .

Describe and analyze an algorithm to find the largest possible score that a player can earn by moving a car from column 1 (the starting line) to column  $n$  (the finish line).

[Hint: Wait, what if all the point values are positive?]

1. Suppose you are given a magic black box that somehow answers the following decision problem in *polynomial time*:

- INPUT: A boolean circuit  $K$  with  $n$  inputs and one output.
- OUTPUT: TRUE if there are input values  $x_1, x_2, \dots, x_n \in \{\text{TRUE}, \text{FALSE}\}$  that make  $K$  output TRUE, and FALSE otherwise.

Using this black box as a subroutine, describe an algorithm that solves the following related search problem in *polynomial time*:

- INPUT: A boolean circuit  $K$  with  $n$  inputs and one output.
- OUTPUT: Input values  $x_1, x_2, \dots, x_n \in \{\text{TRUE}, \text{FALSE}\}$  that make  $K$  output TRUE, or NONE if there are no such inputs.

[*Hint: You can use the magic box more than once.*]

2. An **independent set** in a graph  $G$  is a subset  $S$  of the vertices of  $G$ , such that no two vertices in  $S$  are connected by an edge in  $G$ . Suppose you are given a magic black box that somehow answers the following decision problem in *polynomial time*:

- INPUT: An undirected graph  $G$  and an integer  $k$ .
- OUTPUT: TRUE if  $G$  has an independent set of size  $k$ , and FALSE otherwise.

- (a) Using this black box as a subroutine, describe algorithms that solves the following optimization problem in *polynomial time*:

- INPUT: An undirected graph  $G$ .
- OUTPUT: The size of the largest independent set in  $G$ .

[*Hint: You've seen this problem before.*]

- (b) Using this black box as a subroutine, describe algorithms that solves the following search problem in *polynomial time*:

- INPUT: An undirected graph  $G$ .
- OUTPUT: An independent set in  $G$  of maximum size.

**To think about later:**

3. Formally, a **proper coloring** of a graph  $G = (V, E)$  is a function  $c: V \rightarrow \{1, 2, \dots, k\}$ , for some integer  $k$ , such that  $c(u) \neq c(v)$  for all  $uv \in E$ . Less formally, a valid coloring assigns each vertex of  $G$  a color, such that every edge in  $G$  has endpoints with different colors. The **chromatic number** of a graph is the minimum number of colors in a proper coloring of  $G$ .

Suppose you are given a magic black box that somehow answers the following decision problem *in polynomial time*:

- INPUT: An undirected graph  $G$  and an integer  $k$ .
- OUTPUT: TRUE if  $G$  has a proper coloring with  $k$  colors, and FALSE otherwise.

Using this black box as a subroutine, describe an algorithm that solves the following **coloring problem** *in polynomial time*:

- INPUT: An undirected graph  $G$ .
- OUTPUT: A valid coloring of  $G$  using the minimum possible number of colors.

[*Hint: You can use the magic box more than once. The input to the magic box is a graph and **only** a graph, meaning **only** vertices and edges.*]

Proving that a problem  $X$  is NP-hard requires several steps:

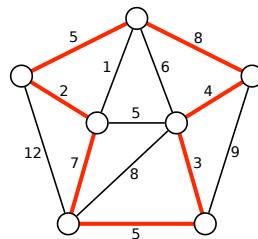
- Choose a problem  $Y$  that you already know is NP-hard (because we told you so in class).
  - Describe an algorithm to solve  $Y$ , using an algorithm for  $X$  as a subroutine. Typically this algorithm has the following form: Given an instance of  $Y$ , transform it into an instance of  $X$ , and then call the magic black-box algorithm for  $X$ .
  - **Prove** that your algorithm is correct. This always requires two separate steps, which are usually of the following form:
    - **Prove** that your algorithm transforms “good” instances of  $Y$  into “good” instances of  $X$ .
    - **Prove** that your algorithm transforms “bad” instances of  $Y$  into “bad” instances of  $X$ . Equivalently: Prove that if your transformation produces a “good” instance of  $X$ , then it was given a “good” instance of  $Y$ .
  - Argue that your algorithm for  $Y$  runs in polynomial time. (This is usually trivial.)
- 

1. Recall the following  $k$ COLOR problem: Given an undirected graph  $G$ , can its vertices be colored with  $k$  colors, so that every edge touches vertices with two different colors?
  - (a) Describe a direct polynomial-time reduction from 3COLOR to 4COLOR.
  - (b) Prove that  $k$ COLOR problem is NP-hard for any  $k \geq 3$ .
2. A *Hamiltonian cycle* in a graph  $G$  is a cycle that goes through every vertex of  $G$  exactly once. Deciding whether an arbitrary graph contains a Hamiltonian cycle is NP-hard.
 

A **tonian cycle** in a graph  $G$  is a cycle that goes through at least *half* of the vertices of  $G$ . Prove that deciding whether a graph contains a tonian cycle is NP-hard.

#### To think about later:

3. Let  $G$  be an undirected graph with weighted edges. A Hamiltonian cycle in  $G$  is **heavy** if the total weight of edges in the cycle is at least half of the total weight of all edges in  $G$ . Prove that deciding whether a graph contains a heavy Hamiltonian cycle is NP-hard.



A heavy Hamiltonian cycle. The cycle has total weight 34; the graph has total weight 67.

Prove that each of the following problems is NP-hard.

1. Given an undirected graph  $G$ , does  $G$  contain a simple path that visits all but 374 vertices?
2. Given an undirected graph  $G$ , does  $G$  have a spanning tree in which every node has degree at most 374?
3. Given an undirected graph  $G$ , does  $G$  have a spanning tree with at most 374 leaves?

Proving that a language  $L$  is undecidable by reduction requires several steps. (These are the essentially the same steps you already use to prove that a problem is NP-hard.)

- Choose a language  $L'$  that you already know is undecidable (because we told you so in class). The simplest choice is usually the standard halting language

$$\text{HALT} := \{\langle M, w \rangle \mid M \text{ halts on } w\}$$

- Describe an algorithm that decides  $L'$ , using an algorithm that decides  $L$  as a black box. Typically your reduction will have the following form:

Given an arbitrary string  $x$ , construct a special string  $y$ ,  
such that  $y \in L$  if and only if  $x \in L'$ .

In particular, if  $L = \text{HALT}$ , your reduction will have the following form:

Given the encoding  $\langle M, w \rangle$  of a Turing machine  $M$  and a string  $w$ ,  
construct a special string  $y$ , such that  
 $y \in L$  if and only if  $M$  halts on input  $w$ .

- Prove that your algorithm is correct. This proof almost always requires two separate steps:
  - Prove that if  $x \in L'$  then  $y \in L$ .
  - Prove that if  $x \notin L'$  then  $y \notin L$ .

**Very important:** Name every object in your proof, and *always* refer to objects by their names. Never refer to “the Turing machine” or “the algorithm” or “the code” or “the input string” or (gods forbid) “it” or “this”, even in casual conversation, even if you’re “just” explaining your intuition, even when you’re “just” *thinking* about the reduction to yourself.

---

Prove that the following languages are undecidable.

1. ACCEPTILLINI :=  $\{\langle M \rangle \mid M \text{ accepts the string ILLINI}\}$
2. ACCEPTTHREE :=  $\{\langle M \rangle \mid M \text{ accepts exactly three strings}\}$
3. ACCEPTPALINDROME :=  $\{\langle M \rangle \mid M \text{ accepts at least one palindrome}\}$
4. ACCEPTONLYPALINDROMES :=  $\{\langle M \rangle \mid \text{Every string accepted by } M \text{ is a palindrome}\}$

A solution for problem 1 appears on the next page; don’t look at it until you’ve thought a bit about the problem first.

**Solution (for problem 1):** For the sake of argument, suppose there is an algorithm DECIDEACCEPTILLINI that correctly decides the language ACCEPTILLINI. Then we can solve the halting problem as follows:

```

DECIDEHALT( $\langle M, w \rangle$ ):
  Encode the following Turing machine  $M'$ :
     $M'(x)$ :
      run  $M$  on input  $w$ 
      return TRUE
  if DECIDEACCEPTILLINI( $\langle M' \rangle$ )
    return TRUE
  else
    return FALSE

```

We prove this reduction correct as follows:

⇒ Suppose  $M$  halts on input  $w$ .

Then  $M'$  accepts every input string  $x$ .

In particular,  $M'$  accepts the string ILLINI.

So DECIDEACCEPTILLINI accepts the encoding  $\langle M' \rangle$ .

So DECIDEHALT correctly accepts the encoding  $\langle M, w \rangle$ .

⇐ Suppose  $M$  does not halt on input  $w$ .

Then  $M'$  diverges on every input string  $x$ .

In particular,  $M'$  does not accept the string ILLINI.

So DECIDEACCEPTILLINI rejects the encoding  $\langle M' \rangle$ .

So DECIDEHALT correctly rejects the encoding  $\langle M, w \rangle$ .

In both cases, DECIDEHALT is correct. But that's impossible, because HALT is undecidable. We conclude that the algorithm DECIDEACCEPTILLINI does not exist. ■

As usual for undecidability proofs, this proof invokes four distinct Turing machines:

- The hypothetical algorithm DECIDEACCEPTILLINI.
- The new algorithm DECIDEHALT that we construct in the solution.
- The arbitrary machine  $M$  whose encoding is part of the input to DECIDEHALT.
- The special machine  $M'$  whose encoding DECIDEHALT constructs (from the encoding of  $M$  and  $w$ ) and then passes to DECIDEACCEPTILLINI.

**Rice's Theorem.** Let  $\mathcal{L}$  be any set of languages that satisfies the following conditions:

- There is a Turing machine  $Y$  such that  $\text{ACCEPT}(Y) \in \mathcal{L}$ .
- There is a Turing machine  $N$  such that  $\text{ACCEPT}(N) \notin \mathcal{L}$ .

The language  $\text{ACCEPTIN}(\mathcal{L}) := \{\langle M \rangle \mid \text{ACCEPT}(M) \in \mathcal{L}\}$  is undecidable.

You may find the following Turing machines useful:

- $M_{\text{ACCEPT}}$  accepts every input.
- $M_{\text{REJECT}}$  rejects every input.
- $M_{\text{HANG}}$  infinite-loops on every input.

Prove that the following languages are undecidable *using Rice's Theorem*:

1.  $\text{ACCEPTREGULAR} := \{\langle M \rangle \mid \text{ACCEPT}(M) \text{ is regular}\}$
2.  $\text{ACCEPTILLINI} := \{\langle M \rangle \mid M \text{ accepts the string } \text{ILLINI}\}$
3.  $\text{ACCEPTPALINDROME} := \{\langle M \rangle \mid M \text{ accepts at least one palindrome}\}$
4.  $\text{ACCEPTTHREE} := \{\langle M \rangle \mid M \text{ accepts exactly three strings}\}$
5.  $\text{ACCEPTUNDECIDABLE} := \{\langle M \rangle \mid \text{ACCEPT}(M) \text{ is undecidable}\}$

**To think about later.** Which of the following languages are undecidable? How would you prove that? Remember that we know several ways to prove undecidability:

- Diagonalization: Assume the language is decidable, and derive an algorithm with self-contradictory behavior.
- Reduction: Assume the language is decidable, and derive an algorithm for a known undecidable language, like HALT or SELFREJECT or NEVERACCEPT.
- Rice's Theorem: Find an appropriate family of languages  $\mathcal{L}$ , a machine  $Y$  that accepts a language in  $\mathcal{L}$ , and a machine  $N$  that does not accept a language in  $\mathcal{L}$ .
- Closure: If two languages  $L$  and  $L'$  are decidable, then the languages  $L \cap L'$  and  $L \cup L'$  and  $L \setminus L'$  and  $L \oplus L'$  and  $L^*$  are all decidable, too.

6.  $\text{ACCEPT}\{\{\varepsilon\}\} := \{\langle M \rangle \mid M \text{ accepts only the string } \varepsilon; \text{ that is, } \text{ACCEPT}(M) = \{\varepsilon\}\}$
7.  $\text{ACCEPT}\{\emptyset\} := \{\langle M \rangle \mid M \text{ does not accept any strings; that is, } \text{ACCEPT}(M) = \emptyset\}$
8.  $\text{ACCEPT}\emptyset := \{\langle M \rangle \mid \text{ACCEPT}(M) \text{ is not an acceptable language}\}$
9.  $\text{ACCEPT}=\text{REJECT} := \{\langle M \rangle \mid \text{ACCEPT}(M) = \text{REJECT}(M)\}$
10.  $\text{ACCEPT}\neq\text{REJECT} := \{\langle M \rangle \mid \text{ACCEPT}(M) \neq \text{REJECT}(M)\}$
11.  $\text{ACCEPT}\cup\text{REJECT} := \{\langle M \rangle \mid \text{ACCEPT}(M) \cup \text{REJECT}(M) = \Sigma^*\}$

**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

1. For each statement below, check “Yes” if the statement is *always* true and “No” otherwise. Each correct answer is worth +1 point; each incorrect answer is worth  $-1/2$  point; checking “I don’t know” is worth  $+1/4$  point; and flipping a coin is (on average) worth  $+1/4$  point. You do **not** need to prove your answer is correct.

**Read each statement very carefully.** Some of these are deliberately subtle.

- (a) Every infinite language is regular.
- (b) If  $L$  is not regular, then for every string  $w \in L$ , there is a DFA that accepts  $w$ .
- (c) If  $L$  is context-free and  $L$  has a finite fooling set, then  $L$  is regular.
- (d) If  $L$  is regular and  $L' \cap L = \emptyset$ , then  $L'$  is regular.
- (e) The language  $\{\textcolor{red}{0}^i 1^j \textcolor{red}{0}^k \mid i + j + k \geq 374\}$  is not regular.
- (f) The language  $\{\textcolor{red}{0}^i 1^j \textcolor{red}{0}^k \mid i + j - k \geq 374\}$  is not regular.
- (g) Let  $M = (Q, \{\textcolor{red}{0}, \textcolor{red}{1}\}, s, A, \delta)$  be an arbitrary DFA, and let  $M' = (Q, \{\textcolor{red}{0}, \textcolor{red}{1}\}, s, A, \delta')$  be the DFA obtained from  $M$  by changing every  $\textcolor{red}{0}$ -transition into a  $\textcolor{red}{1}$ -transition and vice versa. More formally,  $M$  and  $M'$  have the same states, input alphabet, starting state, and accepting states, but  $\delta'(q, \textcolor{red}{0}) = \delta(q, \textcolor{red}{1})$  and  $\delta'(q, \textcolor{red}{1}) = \delta(q, \textcolor{red}{0})$ . Then  $L(M) \cap L(M') = \emptyset$ .
- (h) Let  $M = (Q, \Sigma, s, A, \delta)$  be an arbitrary NFA, and  $M' = (Q', \Sigma, s, A', \delta')$  be any NFA obtained from  $M$  by deleting some subset of the states. More formally, we have  $Q' \subseteq Q$ ,  $A' = A \cap Q'$ , and  $\delta'(q, a) = \delta(q, a) \cap Q'$  for all  $q \in Q'$ . Then  $L(M') \subseteq L(M)$ .
  - (i) For every regular language  $L$ , the language  $\{\textcolor{red}{0}^{|w|} \mid w \in L\}$  is also regular.
  - (j) For every context-free language  $L$ , the language  $\{\textcolor{red}{0}^{|w|} \mid w \in L\}$  is also context-free.
- 2. For any language  $L$ , define

$$\text{STRIPINIT}\textcolor{red}{0}s(L) = \{w \mid \textcolor{red}{0}^j w \in L \text{ for some } j \geq 0\}$$

Less formally,  $\text{STRIPINIT}\textcolor{red}{0}s(L)$  is the set of all strings obtained by stripping any number of initial  $\textcolor{red}{0}$ s from strings in  $L$ . For example, if  $L$  is the one-string language  $\{\textcolor{red}{00011010}\}$ , then

$$\text{STRIPINIT}\textcolor{red}{0}s(L) = \{\textcolor{red}{00011010}, \textcolor{red}{0011010}, \textcolor{red}{011010}, \textcolor{red}{11010}\}.$$

Prove that if  $L$  is a regular language, then  $\text{STRIPINIT}\textcolor{red}{0}s(L)$  is also a regular language.

3. For each of the following languages  $L$  over the alphabet  $\Sigma = \{\text{0}, \text{1}\}$ , give a regular expression that represents  $L$  **and** describe a DFA that recognizes  $L$ .

- (a)  $\{\text{0}^n w \text{1}^n \mid n > 1 \text{ and } w \in \Sigma^*\}$
- (b) All strings in  $\text{0}^* \text{1} \text{0}^*$  whose length is a multiple of 3.

4. The *parity* of a bit-string is **0** if the number of **1** bits is even, and **1** if the number of **1** bits is odd. For example:

$$\text{parity}(\varepsilon) = \text{0} \quad \text{parity}(\text{0010100}) = \text{0} \quad \text{parity}(\text{00101110100}) = \text{1}$$

- (a) Give a *self-contained*, formal, recursive definition of the *parity* function. In particular, do **not** refer to # or other functions defined in class.
- (b) Let  $L$  be an arbitrary regular language. Prove that the language  $\text{EvenParity}(L) := \{w \in L \mid \text{parity}(w) = \text{0}\}$  is also regular.
- (c) Let  $L$  be an arbitrary regular language. Prove that the language  $\text{AddParity}(L) := \{w \bullet \text{parity}(w) \mid w \in L\}$  is also regular. For example, if  $L$  contains the string **11100** and **11000**, then  $\text{AddParity}(L)$  contains the strings **111001** and **110000**.

5. Let  $L$  be the language  $\{\text{0}^i \text{1}^j \text{0}^k \mid i = j \text{ or } j = k\}$ .

- (a) **Prove** that  $L$  is not a regular language.
- (b) Describe a context-free grammar for  $L$ .

**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

1. For each statement below, check “Yes” if the statement is *always* true and “No” otherwise. Each correct answer is worth +1 point; each incorrect answer is worth  $-1/2$  point; checking “I don’t know” is worth  $+1/4$  point; and flipping a coin is (on average) worth  $+1/4$  point. You do **not** need to prove your answer is correct.

**Read each statement very carefully.** Some of these are deliberately subtle.

- (a) No infinite language is regular.
- (b) If  $L$  is regular, then for every string  $w \in L$ , there is a DFA that rejects  $w$ .
- (c) If  $L$  is context-free and  $L$  has a finite fooling set, then  $L$  is not regular.
- (d) If  $L$  is regular and  $L' \cap L = \emptyset$ , then  $L'$  is not regular.
- (e) The language  $\{\textcolor{red}{0}^i 1^j \textcolor{red}{0}^k \mid i + j + k \geq 374\}$  is regular.
- (f) The language  $\{\textcolor{red}{0}^i 1^j \textcolor{red}{0}^k \mid i + j - k \geq 374\}$  is regular.
- (g) Let  $M = (Q, \{\textcolor{red}{0}, \textcolor{red}{1}\}, s, A, \delta)$  be an arbitrary DFA, and let  $M' = (Q, \{\textcolor{red}{0}, \textcolor{red}{1}\}, s, A, \delta')$  be the DFA obtained from  $M$  by changing every  $\textcolor{red}{0}$ -transition into a  $\textcolor{red}{1}$ -transition and vice versa. More formally,  $M$  and  $M'$  have the same states, input alphabet, starting state, and accepting states, but  $\delta'(q, \textcolor{red}{0}) = \delta(q, \textcolor{red}{1})$  and  $\delta'(q, \textcolor{red}{1}) = \delta(q, \textcolor{red}{0})$ . Then  $L(M) \cup L(M') = \{\textcolor{red}{0}, \textcolor{red}{1}\}^*$ .
- (h) Let  $M = (Q, \Sigma, s, A, \delta)$  be an arbitrary NFA, and  $M' = (Q', \Sigma, s, A', \delta')$  be any NFA obtained from  $M$  by deleting some subset of the states. More formally, we have  $Q' \subseteq Q$ ,  $A' = A \cap Q'$ , and  $\delta'(q, a) = \delta(q, a) \cap Q'$  for all  $q \in Q'$ . Then  $L(M') \subseteq L(M)$ .
  - (i) For every non-regular language  $L$ , the language  $\{\textcolor{red}{0}^{|w|} \mid w \in L\}$  is also non-regular.
  - (j) For every context-free language  $L$ , the language  $\{\textcolor{red}{0}^{|w|} \mid w \in L\}$  is also context-free.
- 2. For any language  $L$ , define

$$\text{STRIPFINAL0s}(L) = \{w \mid w\textcolor{red}{0}^n \in L \text{ for some } n \geq 0\}$$

Less formally,  $\text{STRIPFINAL0s}(L)$  is the set of all strings obtained by stripping any number of final  $\textcolor{red}{0}$ s from strings in  $L$ . For example, if  $L$  is the one-string language  $\{\textcolor{red}{01101000}\}$ , then

$$\text{STRIPFINAL0s}(L) = \{\textcolor{red}{01101}, \textcolor{red}{011010}, \textcolor{red}{0110100}, \textcolor{red}{01101000}\}.$$

Prove that if  $L$  is a regular language, then  $\text{STRIPFINAL0s}(L)$  is also a regular language.

3. For each of the following languages  $L$  over the alphabet  $\Sigma = \{\text{0}, \text{1}\}$ , give a regular expression that represents  $L$  **and** describe a DFA that recognizes  $L$ .

- (a)  $\{\text{0}^n w \text{1}^n \mid n \geq 1 \text{ and } w \in \Sigma^+\}$
- (b) All strings in  $\text{0}^* \text{1}^* \text{0}^*$  whose length is even.

4. The *parity* of a bit-string is **0** if the number of **1** bits is even, and **1** if the number of **1** bits is odd. For example:

$$\text{parity}(\varepsilon) = \text{0} \quad \text{parity}(\text{0010100}) = \text{0} \quad \text{parity}(\text{00101110100}) = \text{1}$$

- (a) Give a *self-contained*, formal, recursive definition of the *parity* function. In particular, do **not** refer to # or other functions defined in class.
- (b) Let  $L$  be an arbitrary regular language. Prove that the language  $\text{OddParity}(L) := \{w \in L \mid \text{parity}(w) = \text{1}\}$  is also regular.
- (c) Let  $L$  be an arbitrary regular language. Prove that the language  $\text{AddParity}(L) := \{\text{parity}(w) \cdot w \mid w \in L\}$  is also regular. For example, if  $L$  contains the strings **01110** and **01100**, then  $\text{AddParity}(L)$  contains the strings **101110** and **001100**.

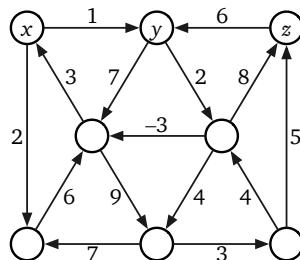
5. Let  $L$  be the language  $\{\text{0}^i \text{1}^j \text{0}^k \mid 2i = k \text{ or } i = 2k\}$ .

- (a) **Prove** that  $L$  is not a regular language.
- (b) Describe a context-free grammar for  $L$ .

**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

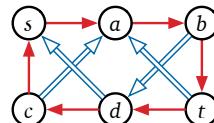
1. *Clearly* indicate the following structures in the directed graph below, or write NONE if the indicated structure does not exist. Don't be subtle; to indicate a collection of edges, draw a heavy black line along the entire length of each edge.



- (a) A depth-first tree rooted at  $x$ .
- (b) A breadth-first tree rooted at  $y$ .
- (c) A shortest-path tree rooted at  $z$ .
- (d) The shortest directed cycle.

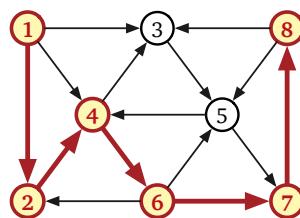
2. Suppose you are given a directed graph  $G$  where some edges are red and the remaining edges are blue. Describe an algorithm to find the shortest walk in  $G$  from one vertex  $s$  to another vertex  $t$  in which no three consecutive edges have the same color. That is, if the walk contains two red edges in a row, the next edge must be blue, and if the walk contains two blue edges in a row, the next edge must be red.

For example, if you are given the graph below (where single arrows are red and double arrows are blue), your algorithm should return the integer 7, because the shortest legal walk from  $s$  to  $t$  is  $s \rightarrow a \rightarrow b \Rightarrow d \rightarrow c \Rightarrow a \rightarrow b \rightarrow c$ .



3. Let  $G$  be an arbitrary (*not* necessarily acyclic) directed graph in which every vertex  $v$  has an integer label  $\ell(v)$ . Describe an algorithm to find the longest directed path in  $G$  whose vertex labels define an increasing sequence. Assume all labels are distinct.

For example, given the following graph as input, your algorithm should return the integer 5, which is the length of the increasing path  $1 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 7 \rightarrow 8$ .



4. Suppose you have an integer array  $A[1..n]$  that *used* to be sorted, but Swedish hackers have overwritten  $k$  entries of  $A$  with random numbers. Because you carefully monitor your system for intrusions, you know *how many* entries of  $A$  are corrupted, but not *which* entries or what the values are.

Describe an algorithm to determine whether your corrupted array  $A$  contains an integer  $x$ . Your input consists of the array  $A$ , the integer  $k$ , and the target integer  $x$ . For example, if  $A$  is the following array,  $k = 4$ , and  $x = 17$ , your algorithm should return TRUE. (The corrupted entries of the array are shaded.)

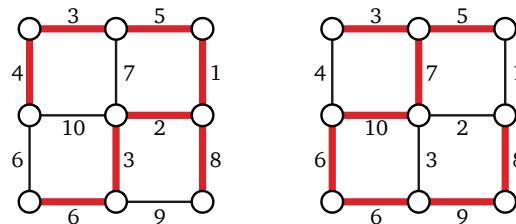
2	3	99	7	11	13	17	19	25	29	31	-5	41	43	47	53	8	61	67	71
---	---	----	---	----	----	----	----	----	----	----	----	----	----	----	----	---	----	----	----

Assume that  $x$  is not equal to any of the the corrupted values, and that all  $n$  array entries are distinct. Report the running time of your algorithm as a function of  $n$  and  $k$ . A solution only for the special case  $k = 1$  is worth 5 points; a complete solution for arbitrary  $k$  is worth 10 points. [Hint: First consider  $k = 0$ ; then consider  $k = 1$ .]

5. Suppose you give one of your interns at Twitbook an undirected graph  $G$  with weighted edges, and you ask them to compute a shortest-path tree rooted at a particular vertex. Two weeks later, your intern finally comes back with a spanning tree  $T$  of  $G$ . Unfortunately, the intern didn't record the shortest-path distances, the direction of the shortest-path edges, or even the source vertex (which you and the intern have both forgotten).

Describe and analyze an algorithm to determine, given a weighted undirected graph  $G$  and a spanning tree  $T$  of  $G$ , whether  $T$  is in fact a *shortest-path* tree in  $G$ . Assume all edge weights are non-negative.

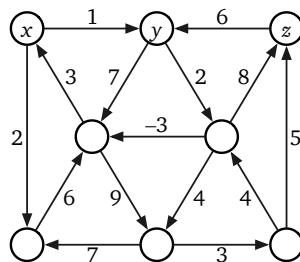
For example, given the inputs shown below, your algorithm should return TRUE for the example on the left, because  $T$  is a shortest-path tree rooted at the upper right vertex of  $G$ , but your algorithm should return FALSE for the example on the right.



**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

1. *Clearly* indicate the following structures in the directed graph below, or write NONE if the indicated structure does not exist. Don't be subtle; to indicate a collection of edges, draw a heavy black line along the entire length of each edge.

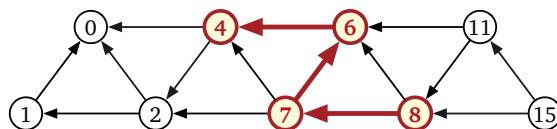


- (a) A depth-first tree rooted at  $x$ .
- (b) A breadth-first tree rooted at  $y$ .
- (c) A shortest-path tree rooted at  $z$ .
- (d) The shortest directed cycle.

2. Let  $G$  be a *directed* graph, where every vertex  $v$  has an associated height  $h(v)$ , and for every edge  $u \rightarrow v$  we have the inequality  $h(u) > h(v)$ . Assume all heights are distinct. The *span* of a path from  $u$  to  $v$  is the height difference  $h(u) - h(v)$ .

Describe and analyze an algorithm to find the *minimum span* of a path in  $G$  with *at least*  $k$  edges. Your input consists of the graph  $G$ , the vertex heights  $h(\cdot)$ , and the integer  $k$ . Report the running time of your algorithm as a function of  $V$ ,  $E$ , and  $k$ .

For example, given the following labeled graph and the integer  $k = 3$  as input, your algorithm should return the integer 4, which is the span of the path  $8 \rightarrow 7 \rightarrow 6 \rightarrow 4$ .



3. Suppose you have an integer array  $A[1..n]$  that used to be sorted, but Swedish hackers have overwritten  $k$  entries of  $A$  with random numbers. Because you carefully monitor your system for intrusions, you know *how many* entries of  $A$  are corrupted, but not *which* entries or what the values are.

Describe an algorithm to determine whether your corrupted array  $A$  contains an integer  $x$ . Your input consists of the array  $A$ , the integer  $k$ , and the target integer  $x$ . For example, if  $A$  is the following array,  $k = 4$ , and  $x = 17$ , your algorithm should return TRUE. (The corrupted entries of the array are shaded.)

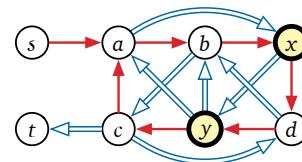
2	3	99	7	11	13	17	19	25	29	31	-5	41	43	47	53	8	61	67	71
---	---	----	---	----	----	----	----	----	----	----	----	----	----	----	----	---	----	----	----

Assume that  $x$  is not equal to any of the the corrupted values, and that all  $n$  array entries are distinct. Report the running time of your algorithm as a function of  $n$  and  $k$ . A solution only for the special case  $k = 1$  is worth 5 points; a complete solution for arbitrary  $k$  is worth 10 points. [Hint: First consider  $k = 0$ ; then consider  $k = 1$ .]

4. Suppose you are given a directed graph  $G$  in which every edge is either red or blue, and a subset of the vertices are marked as *special*. A walk in  $G$  is *legal* if color changes happen only at special vertices. That is, for any two consecutive edges  $u \rightarrow v \rightarrow w$  in a legal walk, if the edges  $u \rightarrow v$  and  $v \rightarrow w$  have different colors, the intermediate vertex  $v$  must be special.

Describe and analyze an algorithm that either returns the length of the shortest legal walk from vertex  $s$  to vertex  $t$ , or correctly reports that no such walk exists.<sup>1</sup>

For example, if you are given the following graph below as input (where single arrows are red, double arrows are blue), with special vertices  $x$  and  $y$ , your algorithm should return the integer 8, which is the length of the shortest legal walk  $s \rightarrow x \rightarrow a \rightarrow b \rightarrow x \Rightarrow y \Rightarrow b \Rightarrow c \Rightarrow t$ . The shorter walk  $s \rightarrow a \rightarrow b \Rightarrow c \Rightarrow t$  is not legal, because vertex  $b$  is not special.



5. Let  $G$  be a directed graph with weighted edges, in which every vertex is colored either red, green, or blue. Describe and analyze an algorithm to compute the length of the shortest walk in  $G$  that starts at a red vertex, then visits any number of vertices of any color, then visits a green vertex, then visits any number of vertices of any color, then visits a blue vertex, then visits any number of vertices of any color, and finally ends at a red vertex. Assume all edge weights are positive.

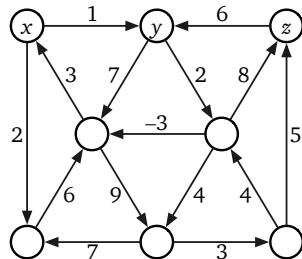
---

<sup>1</sup>If you've read China Miéville's excellent novel *The City & the City*, this problem should look familiar. If you haven't read *The City & the City*, I can't tell you why this problem should look familiar without spoiling the book.

**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

1. *Clearly* indicate the following structures in the directed graph below, or write NONE if the indicated structure does not exist. Don't be subtle; to indicate a collection of edges, draw a heavy black line along the entire length of each edge.



- (a) A depth-first tree rooted at  $x$ .
- (b) A breadth-first tree rooted at  $y$ .
- (c) A shortest-path tree rooted at  $z$ .
- (d) The shortest directed cycle.

2. After a few weeks of following your uphill-downhill walking path to work, your boss demands that you start showing up to work on time, so you decide to change your walking strategy. Your new goal is to walk to the highest altitude you can (to maximize exercise), while keeping the total length of your walk from home to work below some threshold (to make sure you get to work on time). Describe and analyze an algorithm to compute your new favorite route.

Your input consists of an *undirected* graph  $G$ , where each vertex  $v$  has a height  $h(v)$  and each edge  $e$  has a *positive* length  $\ell(e)$ , along with a start vertex  $s$ , a target vertex  $t$ , and a maximum length  $L$ . Your algorithm should return the *maximum height* reachable by a walk from  $s$  to  $t$  in  $G$ , whose total length is at most  $L$ .

[Hint: This is the same input as HW8 problem 1, but the problem is completely different. In particular, the number of uphill/downhill switches in your walk is irrelevant.]

3. Suppose you have an integer array  $A[1..n]$  that *used* to be sorted, but Swedish hackers have overwritten  $k$  entries of  $A$  with random numbers. Because you carefully monitor your system for intrusions, you know *how many* entries of  $A$  are corrupted, but not *which* entries or what the values are.

Describe an algorithm to determine whether your corrupted array  $A$  contains an integer  $x$ . Your input consists of the array  $A$ , the integer  $k$ , and the target integer  $x$ . For example, if  $A$  is the following array,  $k = 4$ , and  $x = 17$ , your algorithm should return TRUE. (The corrupted entries of the array are shaded.)

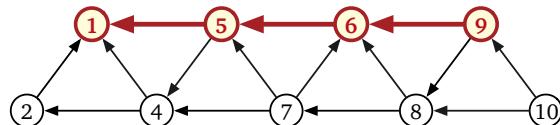
2	3	99	7	11	13	17	19	25	29	31	-5	41	43	47	53	8	61	67	71
---	---	----	---	----	----	----	----	----	----	----	----	----	----	----	----	---	----	----	----

Assume that  $x$  is not equal to any of the the corrupted values, and that all  $n$  array entries are distinct. Report the running time of your algorithm as a function of  $n$  and  $k$ . A solution only for the special case  $k = 1$  is worth 5 points; a complete solution for arbitrary  $k$  is worth 10 points. [Hint: First consider  $k = 0$ ; then consider  $k = 1$ .]

4. Let  $G$  be a **directed** graph, where every vertex  $v$  has an associated height  $h(v)$ , and for every edge  $u \rightarrow v$  we have the inequality  $h(u) > h(v)$ . Assume all heights are distinct. The *span* of a path from  $u$  to  $v$  is the height difference  $h(u) - h(v)$ .

Describe and analyze an algorithm to find the **maximum span** of a path in  $G$  with at most  $k$  edges. Your input consists of the graph  $G$ , the vertex heights  $h(\cdot)$ , and the integer  $k$ . Report the running time of your algorithm as a function of  $V$ ,  $E$ , and  $k$ .

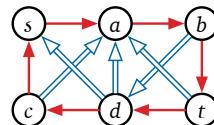
For example, given the following labeled graph and the integer  $k = 3$  as input, your algorithm should return the integer 8, which is the span of the downward path  $9 \rightarrow 6 \rightarrow 5 \rightarrow 1$ .



[Hint: This is a very different question from problem 2.]

5. Suppose you are given a directed graph  $G$  where some edges are red and the remaining edges are blue, along with two vertices  $s$  and  $t$ . Describe an algorithm to compute the length of the shortest walk in  $G$  from  $s$  to  $t$  that traverses an even number of red edges and an even number of blue edges. If the walk traverses the same edge multiple times, each traversal counts toward the total for that color.

For example, if you are given the graph below (where single arrows are red and double arrows are blue), your algorithm should return the integer 6, because the shortest legal walk from  $s$  to  $t$  is  $s \rightarrow a \rightarrow b \Rightarrow d \Rightarrow a \rightarrow b \rightarrow t$ .



CS/ECE 374 A ✦ Spring 2018

❖ Final Exam ♫ ❁

May 8, 2018

Real name:	
NetID:	

Gradescope name:	
Gradescope email:	

- 
- ***Don't panic!***
  - If you brought anything except your writing implements and your two double-sided 8½" × 11" cheat sheets, please put it away for the duration of the exam. In particular, please turn off and put away *all* medically unnecessary electronic devices.
  - Please clearly print your real name, your university NetID, your Gradescope name, and your Gradescope email address in the boxes above. **We will not scan this page into Gradescope.**
  - Please also print **only the name you are using on Gradescope** at the top of every page of the answer booklet, except this cover page. These are the pages we will scan into Gradescope.
  - Please do not write outside the black boxes on each page; these indicate the area of the page that the scanner can actually see.
  - **Please read the entire exam before writing anything.** Please ask for clarification if any question is unclear.
  - **The exam lasts 180 minutes.**
  - If you run out of space for an answer, continue on the back of the page, or on the blank pages at the end of this booklet, **but please tell us where to look.** Alternatively, feel free to tear out the blank pages and use them as scratch paper.
  - As usual, answering any (sub)problem with “I don’t know” (and nothing else) is worth 25% partial credit. **Yes, even for problem 1.** Correct, complete, but suboptimal solutions are *always* worth more than 25%. A blank answer is not the same as “I don’t know”.
  - **Please return your cheat sheets and all scratch paper with your answer booklet.**
  - **Good luck!** And thanks for a great semester!
-

Beware of the man who works hard to learn something,  
learns it, and finds himself no wiser than before.

He is full of murderous resentment of people who are ignorant  
without having come by their ignorance the hard way.

— Bokonon

## Final Exam ▷ Problem 1

For each of the following questions, indicate *every* correct answer by marking the “Yes” box, and indicate *every* incorrect answer by marking the “No” box. **Assume  $P \neq NP$ .** If there is any other ambiguity or uncertainty, mark the “No” box. For example:

<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No	$2 + 2 = 4$
<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No	$x + y = 5$
<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No	3SAT can be solved in polynomial time.
<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No	Jeff is not the Queen of England.
<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No	If $P = NP$ then Jeff is the Queen of England.

There are 40 yes/no choices altogether. Each correct choice is worth  $+\frac{1}{2}$  point; each incorrect choice is worth  $-\frac{1}{4}$  point. To indicate “I don’t know”, write IDK to the left of the Yes/No boxes; each IDK is worth  $+\frac{1}{8}$  point.

- (a) Which of the following statements is true for *every* language  $L \subseteq \{0, 1\}^*$ ?

<input type="checkbox"/> Yes	<input type="checkbox"/> No	$L$ is infinite.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	$L^*$ contains the empty string $\epsilon$ .
<input type="checkbox"/> Yes	<input type="checkbox"/> No	$L^*$ is decidable.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $L$ is regular then $(L^*)^*$ is regular.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $L$ is the intersection of two decidable languages, then $L$ is decidable.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $L$ is the intersection of two undecidable languages, then $L$ is undecidable.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $L$ is the complement of a regular language, then $L^*$ is regular.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $L$ has an infinite fooling set, then $L$ is undecidable.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	$L$ is decidable if and only if its complement $\overline{L}$ is undecidable.

(b) Which of the following statements is true for *every* directed graph  $G = (V, E)$ ?

Yes	No

$E \neq \emptyset$ .

Given the graph  $G$  as input, Floyd-Warshall runs in  $O(E^3)$  time.

If  $G$  has at least one source and at least one sink, then  $G$  is a dag.

We can compute a spanning tree of  $G$  using whatever-first search.

If the edges of  $G$  are weighted, we can compute the shortest path from any node  $s$  to any node  $t$  in  $O(E \log V)$  time using Dijkstra's algorithm.

(c) Which of the following languages over the alphabet  $\{\text{0}, \text{1}\}$  are *regular*?

Yes	No

$\{\text{0}^m \text{10}^n \mid m \leq n\}$

$\{\text{0}^m \text{10}^n \mid m + n \geq 374\}$

Binary representations of all perfect squares

$\{xy \mid yx \text{ is a palindrome}\}$

$\{\langle M \rangle \mid M \text{ accepts a finite number of non-palindromes}\}$

(d) Which of the following languages are *decidable*?

Yes	No

Binary representations of all perfect squares

$\{xy \in \{\text{0}, \text{1}\}^* \mid yx \text{ is a palindrome}\}$

$\{\langle M \rangle \mid M \text{ accepts the binary representation of every perfect square}\}$

$\{\langle M \rangle \mid M \text{ accepts a finite number of non-palindromes}\}$

The set of all regular expressions that represent the language  $\{\text{0}, \text{1}\}^*$ .  
(This is a language over the alphabet  $\{\text{0}, \text{E}, \text{0}, \text{1}, \text{*}, \text{+}, (\text{, })\}$ .)

(e) Which of the following languages can be proved undecidable *using Rice's Theorem*?

<input type="checkbox"/> Yes	<input type="checkbox"/> No	$\{\langle M \rangle \mid M \text{ accepts a finite number of strings}\}$
<input type="checkbox"/> Yes	<input type="checkbox"/> No	$\{\langle M \rangle \mid M \text{ accepts both } \langle M \rangle \text{ and } \langle M \rangle^R\}$
<input type="checkbox"/> Yes	<input type="checkbox"/> No	$\{\langle M \rangle \mid M \text{ accepts exactly 374 palindromes}\}$
<input type="checkbox"/> Yes	<input type="checkbox"/> No	$\{\langle M \rangle \mid M \text{ accepts some string } w \text{ after at most }  w ^2 \text{ steps}\}$

---

(f) Suppose we want to prove that the following language is undecidable.

$$\text{CHALMERS} := \{\langle M \rangle \mid M \text{ accepts both STEAMED and HAMS}\}$$

Professor Skinner suggests a reduction from the standard halting language

$$\text{HALT} := \{\langle M \rangle \# w \mid M \text{ halts on inputs } w\}.$$

Specifically, suppose there is a Turing machine  $Ch$  that decides CHALMERS. Professor Skinner claims that the following algorithm decides HALT.

```

DECIDEHALT( $\langle M \rangle \# w$ ):
  Encode the following Turing machine:
    AURORABOREALIS( $x$ ):
      if  $x = \text{STEAMED}$  or  $x = \text{HAMS}$  or  $x = \text{ALBANY}$ 
        run  $M$  on input  $w$ 
        return FALSE
      else
        return TRUE
  return  $Ch(\text{AURORABOREALIS})$ 

```

Which of the following statements is true for all inputs  $\langle M \rangle \# w$ ?

<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $M$ accepts $w$ , then AURORABOREALIS accepts CLAMS.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $M$ rejects $w$ , then AURORABOREALIS rejects UTICA.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $M$ rejects $w$ , then AURORABOREALIS halts on every input string.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $M$ accepts $w$ , then $Ch$ accepts (AURORABOREALIS).
<input type="checkbox"/> Yes	<input type="checkbox"/> No	DECIDEHALT decides the language HALT. (That is, Professor Skinner's reduction is actually correct.)
<input type="checkbox"/> Yes	<input type="checkbox"/> No	DECIDEHALT actually runs (or simulates) $M$ .
<input type="checkbox"/> Yes	<input type="checkbox"/> No	We could have proved CHALMERS is undecidable using Rice's theorem instead of this reduction.

---

(g) Consider the following pair of languages:

- $\text{3COLOR} := \{G \mid G \text{ is a 3-colorable undirected graph}\}$
- $\text{TREE} := \{G \mid G \text{ is a connected acyclic undirected graph}\}$

(For concreteness, assume that in both of these languages, graphs are represented by adjacency matrices.) Which of the following *must* be true, assuming  $P \neq NP$ ?

Yes	No
-----	----

TREE  $\cup$  3COLOR is NP-hard.

Yes	No
-----	----

TREE  $\cap$  3COLOR is NP-hard.

Yes	No
-----	----

3COLOR is undecidable.

Yes	No
-----	----

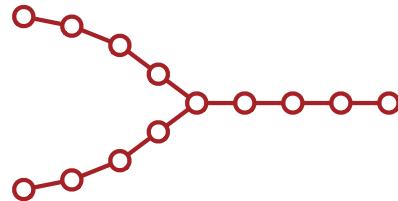
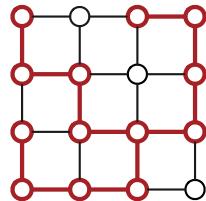
There is a polynomial-time reduction from 3COLOR to TREE.

Yes	No
-----	----

There is a polynomial-time reduction from TREE to 3COLOR.

---

A *wye* is an undirected graph that looks like the capital letter Y. More formally, a wye consists of three paths of equal length with one common endpoint, called the *hub*.



This grid graph contains a wye whose paths have length 4.

**Prove** that the following problem is NP-hard: Given an undirected graph  $G$ , what is the largest wye that is a subgraph of  $G$ ? The three paths of the wye must not share any vertices except the hub, and they must have exactly the same length.

---

Fix the alphabet  $\Sigma = \{0, 1\}$ . Recall that a *run* in a string  $w \in \Sigma^*$  is a maximal non-empty substring in which all symbols are equal. For example, the string  $00000100011111101$  consists of exactly six runs:  $0000010001111110 = 00000 \cdot 1 \cdot 000 \cdot 111111 \cdot 0 \cdot 1$ .

- (a) Let  $L$  be the set of all strings in  $\Sigma^*$  where every run has odd length. For example,  $L$  contains the string  $000100000$ , but  $L$  does not contain the string  $00011$ .

Describe both a regular expression for  $L$  and a DFA that accepts  $L$ .

- (b) Let  $L'$  be the set of all strings in  $\Sigma^*$  that have the same number of even-length runs and odd-length runs. For example,  $L'$  does not contain the string  $000011101$ , because it has three odd-length runs but only one even-length run, but  $L'$  does contain the string  $0000111011$ , because it has two runs of each parity.

*Prove* that  $L'$  is not regular.

Suppose we want to split an array  $A[1..n]$  of integers into  $k$  contiguous intervals that partition the sum of the values as evenly as possible. Specifically, define the *cost* of such a partition as the maximum, over all  $k$  intervals, of the sum of the values in that interval; our goal is to minimize this cost. Describe and analyze an algorithm to compute the minimum cost of a partition of  $A$  into  $k$  intervals, given the array  $A$  and the integer  $k$  as input.

For example, given the array  $A = [1, 6, -1, 8, 0, 3, 3, 9, 8, 8, 7, 4, 9, 8, 9, 4, 8, 4, 8, 2]$  and the integer  $k = 3$  as input, your algorithm should return the integer 37, which is the cost of the following partition:

$$[ \overbrace{1, 6, -1, 8, 0, 3, 3, 9, 8}^{37} | \overbrace{8, 7, 4, 9, 8}^{36} | \overbrace{9, 4, 8, 4, 8, 2}^{35} ]$$

The numbers above each interval show the sum of the values in that interval.

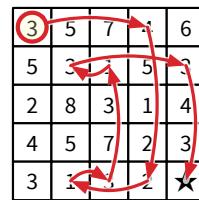
---

- (a) Fix the alphabet  $\Sigma = \{\textcolor{red}{0}, \textcolor{red}{1}\}$ . Describe and analyze an efficient algorithm for the following problem: Given an NFA  $M$  over  $\Sigma$ , does  $M$  accept at least one string? Equivalently, is  $L(M) \neq \emptyset$ ?
- (b) Recall from Homework 10 that deciding whether a given NFA accepts *every* string is NP-hard. Also recall that the complement of every regular language is regular; thus, for any NFA  $M$ , there is another NFA  $M'$  such that  $L(M') = \Sigma^* \setminus L(M)$ . So why doesn't your algorithm from part (a) imply that P=NP?
-

A **number maze** is an  $n \times n$  grid of positive integers. A token starts in the upper left corner; your goal is to move the token to the lower-right corner. On each turn, you are allowed to move the token up, down, left, or right; the distance you may move the token is determined by the number on its current square. For example, if the token is on a square labeled 3, then you may move the token three steps up, three steps down, three steps left, or three steps right. However, you are never allowed to move the token off the edge of the board.

Describe and analyze an efficient algorithm that either returns the minimum number of moves required to solve a given number maze, or correctly reports that the maze has no solution.

3	5	7	4	6
5	3	1	5	3
2	8	3	1	4
4	5	7	2	3
3	1	3	2	★



A  $5 \times 5$  number maze that can be solved in eight moves.

(scratch paper)

(scratch paper)

(scratch paper)

(scratch paper)

**Some useful NP-hard problems.** You are welcome to use any of these in your own NP-hardness proofs, except of course for the specific problem you are trying to prove NP-hard.

**CIRCUITSAT:** Given a boolean circuit, are there any input values that make the circuit output TRUE?

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three distinct literals per clause, does the formula have a satisfying assignment?

**MAXINDEPENDENTSET:** Given an undirected graph  $G$ , what is the size of the largest subset of vertices in  $G$  that have no edges among them?

**MAXCLIQUE:** Given an undirected graph  $G$ , what is the size of the largest complete subgraph of  $G$ ?

**MINVERTEXCOVER:** Given an undirected graph  $G$ , what is the size of the smallest subset of vertices that touch every edge in  $G$ ?

**MINSETCOVER:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subcollection whose union is  $S$ ?

**MINHITTINGSET:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subset of  $S$  that intersects every subset  $S_i$ ?

**3COLOR:** Given an undirected graph  $G$ , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

**HAMILTONIANPATH:** Given graph  $G$  (either directed or undirected), is there a path in  $G$  that visits every vertex exactly once?

**HAMILTONIANCYCLE:** Given a graph  $G$  (either directed or undirected), is there a cycle in  $G$  that visits every vertex exactly once?

**TRAVELINGSALESMAN:** Given a graph  $G$  (either directed or undirected) with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in  $G$ ?

**LONGESTPATH:** Given a graph  $G$  (either directed or undirected, possibly with weighted edges), what is the length of the longest simple path in  $G$ ?

**STEINERTREE:** Given an undirected graph  $G$  with some of the vertices marked, what is the minimum number of edges in a subtree of  $G$  that contains every marked vertex?

**SUBSETSUM:** Given a set  $X$  of positive integers and an integer  $k$ , does  $X$  have a subset whose elements sum to  $k$ ?

**PARTITION:** Given a set  $X$  of positive integers, can  $X$  be partitioned into two subsets with the same sum?

**3PARTITION:** Given a set  $X$  of  $3n$  positive integers, can  $X$  be partitioned into  $n$  three-element subsets, all with the same sum?

**INTEGERLINEARPROGRAMMING:** Given a matrix  $A \in \mathbb{Z}^{n \times d}$  and two vectors  $b \in \mathbb{Z}^n$  and  $c \in \mathbb{Z}^d$ , compute  $\max\{c \cdot x \mid Ax \leq b, x \geq 0, x \in \mathbb{Z}^d\}$ .

**FEASIBLEILP:** Given a matrix  $A \in \mathbb{Z}^{n \times d}$  and a vector  $b \in \mathbb{Z}^n$ , determine whether the set of feasible integer points  $\max\{x \in \mathbb{Z}^d \mid Ax \leq b, x \geq 0\}$  is empty.

**DRAUGHTS:** Given an  $n \times n$  international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

**SUPERMARIOBROTHERS:** Given an  $n \times n$  Super Mario Brothers level, can Mario reach the castle?

**STEAMEDHAMS:** Aurora borealis? At this time of year, at this time of day, in this part of the country, localized entirely within your kitchen? May I see it?

CS/ECE 374 A ✦ Spring 2018

❖ Final Exam ♫.♪ ~

May 8, 2018

Real name:	
NetID:	

Gradescope name:	
Gradescope email:	

- 
- ***Don't panic!***
  - If you brought anything except your writing implements and your two double-sided 8½" × 11" cheat sheets, please put it away for the duration of the exam. In particular, please turn off and put away *all* medically unnecessary electronic devices.
  - Please clearly print your real name, your university NetID, your Gradescope name, and your Gradescope email address in the boxes above. **We will not scan this page into Gradescope.**
  - Please also print **only the name you are using on Gradescope** at the top of every page of the answer booklet, except this cover page. These are the pages we will scan into Gradescope.
  - Please do not write outside the black boxes on each page; these indicate the area of the page that the scanner can actually see.
  - **Please read the entire exam before writing anything.** Please ask for clarification if any question is unclear.
  - **The exam lasts 180 minutes.**
  - If you run out of space for an answer, continue on the back of the page, or on the blank pages at the end of this booklet, **but please tell us where to look.** Alternatively, feel free to tear out the blank pages and use them as scratch paper.
  - As usual, answering any (sub)problem with “I don’t know” (and nothing else) is worth 25% partial credit. **Yes, even for problem 1.** Correct, complete, but suboptimal solutions are *always* worth more than 25%. A blank answer is not the same as “I don’t know”.
  - **Please return your cheat sheets and all scratch paper with your answer booklet.**
  - **Good luck!** And thanks for a great semester!
-

Beware of the man who works hard to learn something,  
learns it, and finds himself no wiser than before.

He is full of murderous resentment of people who are ignorant  
without having come by their ignorance the hard way.

— Bokonon

## Final Exam ▶ Problem 1

For each of the following questions, indicate *every* correct answer by marking the “Yes” box, and indicate *every* incorrect answer by marking the “No” box. **Assume  $P \neq NP$ .** If there is any other ambiguity or uncertainty, mark the “No” box. For example:

<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No	$2 + 2 = 4$
<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No	$x + y = 5$
<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No	3SAT can be solved in polynomial time.
<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No	Jeff is not the Queen of England.
<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No	If $P = NP$ then Jeff is the Queen of England.

There are 40 yes/no choices altogether. Each correct choice is worth  $+\frac{1}{2}$  point; each incorrect choice is worth  $-\frac{1}{4}$  point. To indicate “I don’t know”, write IDK to the left of the Yes/No boxes; each IDK is worth  $+\frac{1}{8}$  point.

- (a) Which of the following statements is true for *every* language  $L \subseteq \{0, 1\}^*$ ?

<input type="checkbox"/> Yes	<input type="checkbox"/> No	$L$ is finite.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	$L^*$ contains the empty string $\epsilon$ .
<input type="checkbox"/> Yes	<input type="checkbox"/> No	$L^*$ is decidable.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $L$ is regular then $\Sigma^* \setminus L^*$ is regular.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $L$ is the intersection of two decidable languages, then $L$ is decidable.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $L$ is the intersection of two undecidable languages, then $L$ is undecidable.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $L^*$ is the complement of a regular language, then $L$ is regular.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	If $L$ is undecidable, then every fooling set for $L$ is infinite.
<input type="checkbox"/> Yes	<input type="checkbox"/> No	$L$ is decidable if and only if its complement $\overline{L}$ is undecidable.

(b) Which of the following statements is true for *every* directed graph  $G = (V, E)$ ?

Yes	No

$E \neq \emptyset$ .

Given the graph  $G$  as input, Floyd-Warshall runs in  $O(E^3)$  time.

If  $G$  has at least one source and at least one sink, then  $G$  is a dag.

We can compute a spanning tree of  $G$  using whatever-first search.

If the edges of  $G$  are weighted, we can compute the shortest path from any node  $s$  to any node  $t$  in  $O(E \log V)$  time using Dijkstra's algorithm.

(c) Which of the following languages over the alphabet  $\{\text{0}, \text{1}\}$  are *regular*?

Yes	No

$\{\text{0}^m \text{10}^n \mid m \geq n\}$

$\{\text{0}^m \text{10}^n \mid m - n \geq 374\}$

Binary representations of all integers divisible by 374

$\{xy \mid yx \text{ is a palindrome}\}$

$\{\langle M \rangle \mid M \text{ accepts a finite number of non-palindromes}\}$

(d) Which of the following languages are *decidable*?

Yes	No

Binary representations of all integers divisible by 374

$\{xy \in \{\text{0}, \text{1}\}^* \mid yx \text{ is a palindrome}\}$

$\{\langle M \rangle \mid M \text{ accepts the binary representation of every integer divisible by 374}\}$

$\{\langle M \rangle \mid M \text{ accepts a finite number of non-palindromes}\}$

The set of all regular expressions that represent the language  $\{\text{0}, \text{1}\}^*$ .  
(This is a language over the alphabet  $\{\text{0}, \text{E}, \text{0}, \text{1}, \text{*}, +, (\text{, })\}$ .)

(e) Which of the following languages can be proved undecidable *using Rice's Theorem*?

Yes	No

- $\{\langle M \rangle \mid M \text{ accepts an infinite number of strings}\}$
- $\{\langle M \rangle \mid M \text{ accepts either } \langle M \rangle \text{ or } \langle M \rangle^R\}$
- $\{\langle M \rangle \mid M \text{ accepts } \textcolor{red}{001100} \text{ but rejects } \textcolor{red}{110011}\}$
- $\{\langle M \rangle \mid M \text{ accepts some string } w \text{ after at most } |w|^2 \text{ steps}\}$

(f) Suppose we want to prove that the following language is undecidable.

$$\text{CHALMERS} := \{\langle M \rangle \mid M \text{ accepts both STEAMED and HAMS}\}$$

Professor Skinner suggests a reduction from the standard halting language

$$\text{HALT} := \{\langle M \rangle \# w \mid M \text{ halts on inputs } w\}.$$

Specifically, suppose there is a Turing machine  $Ch$  that decides CHALMERS. Professor Skinner claims that the following algorithm decides HALT.

```

DECIDEHALT( $\langle M \rangle \# w$ ):
  Encode the following Turing machine:
    AURORABOREALIS(x):
      if  $x = \text{STEAMED}$  or  $x = \text{HAMS}$  or  $x = \text{ALBANY}$ 
        run  $M$  on input  $w$ 
        return TRUE
      else
        return FALSE
  return  $Ch(\text{AURORABOREALIS})$ 

```

Which of the following statements is true for all inputs  $\langle M \rangle \# w$ ?

Yes	No

- If  $M$  accepts  $w$ , then AURORABOREALIS accepts CLAMS.
- If  $M$  rejects  $w$ , then AURORABOREALIS rejects UTICA.
- If  $M$  hangs on  $w$ , then AURORABOREALIS accepts every input string.
- If  $M$  accepts  $w$ , then  $Ch$  accepts  $\langle \text{AURORABOREALIS} \rangle$ .
- DECIDEHALT decides the language HALT. (That is, Professor Skinner's reduction is actually correct.)
- DECIDEHALT actually runs (or simulates)  $M$ .
- We could have proved CHALMERS is undecidable using Rice's theorem instead of this reduction.

(g) Consider the following pair of languages:

- $\text{3COLOR} := \{G \mid G \text{ is a 3-colorable undirected graph}\}$
- $\text{TREE} := \{G \mid G \text{ is a connected acyclic undirected graph}\}$

(For concreteness, assume that in both of these languages, graphs are represented by adjacency matrices.) Which of the following *must* be true, assuming  $P \neq NP$ ?

Yes	No
-----	----

TREE  $\cup$  3COLOR is NP-hard.

Yes	No
-----	----

TREE  $\cap$  3COLOR is NP-hard.

Yes	No
-----	----

3COLOR is undecidable.

Yes	No
-----	----

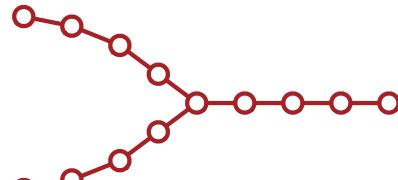
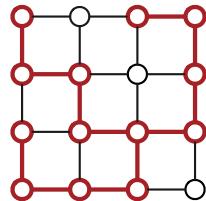
There is a polynomial-time reduction from 3COLOR to TREE.

Yes	No
-----	----

There is a polynomial-time reduction from TREE to 3COLOR.

---

A *wye* is an undirected graph that looks like the capital letter Y. More formally, a wye consists of three paths of equal length with one common endpoint, called the *hub*.



This grid graph contains a wye whose paths have length 4.

**Prove** that the following problem is NP-hard: Given an undirected graph  $G$ , what is the largest wye that is a subgraph of  $G$ ? The three paths of the wye must not share any vertices except the hub, and they must have exactly the same length.

---

## Final Exam □ Problem 3

Fix the alphabet  $\Sigma = \{0, 1\}$ . Recall that a *run* in a string  $w \in \Sigma^*$  is a maximal non-empty substring in which all symbols are equal. For example, the string  $00000100011111101$  consists of exactly six runs:  $0000010001111110 = 00000 \bullet 1 \bullet 000 \bullet 111111 \bullet 0 \bullet 1$ .

- (a) Let  $L$  be the set of all strings in  $\Sigma^*$  that contains at least one run whose length is divisible by 3. For example,  $L$  contains the string  $0011111000$ , but  $L$  does not contain the string  $1000011$ .

Describe both a regular expression for  $L$  and a DFA that accepts  $L$ .

- (b) Let  $L'$  be the set of all strings in  $\Sigma^*$  that have the same number of even-length runs and odd-length runs. For example,  $L'$  does not contain the string  $000011101$ , because it has three odd-length runs but only one even-length run, but  $L'$  does contain the string  $0000111011$ , because it has two runs of each parity.

*Prove* that  $L'$  is not regular.

Suppose we want to split an array  $A[1..n]$  of integers into  $k$  contiguous intervals that partition the sum of the values as evenly as possible. Specifically, define the *quality* of such a partition as the minimum, over all  $k$  intervals, of the sum of the values in that interval; our goal is to maximize quality. Describe and analyze an algorithm to compute the maximum quality of a partition of  $A$  into  $k$  intervals, given the array  $A$  and the integer  $k$  as input.

For example, given the array  $A = [1, 6, -1, 8, 0, 3, 3, 9, 8, 8, 7, 4, 9, 8, 9, 4, 8, 4, 8, 2]$  and the integer  $k = 3$  as input, your algorithm should return the integer 35, which is the quality of the following partition:

$$[ \overbrace{1, 6, -1, 8, 0, 3, 3, 9, 8}^{37} | \overbrace{8, 7, 4, 9, 8}^{36} | \overbrace{9, 4, 8, 4, 8, 2}^{35} ]$$

The numbers above each interval show the sum of the values in that interval.

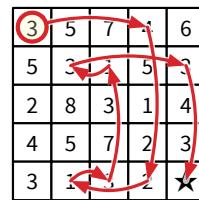
---

- (a) Fix the alphabet  $\Sigma = \{\textcolor{red}{0}, \textcolor{red}{1}\}$ . Describe and analyze an efficient algorithm for the following problem:  
Given a DFA  $M$  over  $\Sigma$ , does  $M$  reject *any* string? Equivalently, is  $L(M) \neq \Sigma^*$ ?
- (b) Recall from Homework 10 that the corresponding problem for NFAs is NP-hard. But any NFA can be transformed into equivalent DFA using the incremental subset construction. So why doesn't your algorithm from part (a) imply that P=NP?
-

A **number maze** is an  $n \times n$  grid of positive integers. A token starts in the upper left corner; your goal is to move the token to the lower-right corner. On each turn, you are allowed to move the token up, down, left, or right; the distance you may move the token is determined by the number on its current square. For example, if the token is on a square labeled 3, then you may move the token three steps up, three steps down, three steps left, or three steps right. However, you are never allowed to move the token off the edge of the board.

Describe and analyze an efficient algorithm that either returns the minimum number of moves required to solve a given number maze, or correctly reports that the maze has no solution.

3	5	7	4	6
5	3	1	5	3
2	8	3	1	4
4	5	7	2	3
3	1	3	2	★



A  $5 \times 5$  number maze that can be solved in eight moves.

(scratch paper)

(scratch paper)

(scratch paper)

(scratch paper)

**Some useful NP-hard problems.** You are welcome to use any of these in your own NP-hardness proofs, except of course for the specific problem you are trying to prove NP-hard.

**CIRCUITSAT:** Given a boolean circuit, are there any input values that make the circuit output TRUE?

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three distinct literals per clause, does the formula have a satisfying assignment?

**MAXINDEPENDENTSET:** Given an undirected graph  $G$ , what is the size of the largest subset of vertices in  $G$  that have no edges among them?

**MAXCLIQUE:** Given an undirected graph  $G$ , what is the size of the largest complete subgraph of  $G$ ?

**MINVERTEXCOVER:** Given an undirected graph  $G$ , what is the size of the smallest subset of vertices that touch every edge in  $G$ ?

**MINSETCOVER:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subcollection whose union is  $S$ ?

**MINHITTINGSET:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subset of  $S$  that intersects every subset  $S_i$ ?

**3COLOR:** Given an undirected graph  $G$ , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

**HAMILTONIANPATH:** Given graph  $G$  (either directed or undirected), is there a path in  $G$  that visits every vertex exactly once?

**HAMILTONIANCYCLE:** Given a graph  $G$  (either directed or undirected), is there a cycle in  $G$  that visits every vertex exactly once?

**TRAVELINGSALESMAN:** Given a graph  $G$  (either directed or undirected) with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in  $G$ ?

**LONGESTPATH:** Given a graph  $G$  (either directed or undirected, possibly with weighted edges), what is the length of the longest simple path in  $G$ ?

**STEINERTREE:** Given an undirected graph  $G$  with some of the vertices marked, what is the minimum number of edges in a subtree of  $G$  that contains every marked vertex?

**SUBSETSUM:** Given a set  $X$  of positive integers and an integer  $k$ , does  $X$  have a subset whose elements sum to  $k$ ?

**PARTITION:** Given a set  $X$  of positive integers, can  $X$  be partitioned into two subsets with the same sum?

**3PARTITION:** Given a set  $X$  of  $3n$  positive integers, can  $X$  be partitioned into  $n$  three-element subsets, all with the same sum?

**INTEGERLINEARPROGRAMMING:** Given a matrix  $A \in \mathbb{Z}^{n \times d}$  and two vectors  $b \in \mathbb{Z}^n$  and  $c \in \mathbb{Z}^d$ , compute  $\max\{c \cdot x \mid Ax \leq b, x \geq 0, x \in \mathbb{Z}^d\}$ .

**FEASIBLEILP:** Given a matrix  $A \in \mathbb{Z}^{n \times d}$  and a vector  $b \in \mathbb{Z}^n$ , determine whether the set of feasible integer points  $\max\{x \in \mathbb{Z}^d \mid Ax \leq b, x \geq 0\}$  is empty.

**DRAUGHTS:** Given an  $n \times n$  international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

**SUPERMARIOBROTHERS:** Given an  $n \times n$  Super Mario Brothers level, can Mario reach the castle?

**STEAMEDHAMS:** Aurora borealis? At this time of year, at this time of day, in this part of the country, localized entirely within your kitchen? May I see it?

# CS 373U: Combinatorial Algorithms, Spring 2004

## Homework 0

Due January 28, 2004 at noon

Name:	
Net ID:	Alias:



I understand the Homework Instructions and FAQ.

- 
- Neatly print your full name, your NetID, and an alias of your choice in the boxes above. Grades will be listed on the course web site by alias; for privacy reasons, your alias should not resemble your name or NetID. By providing an alias, you agree to let us list your grades; if you do not provide an alias, your grades will not be listed. *Never give us your Social Security number!*
  - Before you do anything else, read the Homework Instructions and FAQ on the course web page, and then check the box above. This web page gives instructions on how to write and submit homeworks—staple your solutions together in order, start each numbered problem on a new sheet of paper, write your name and NetID one every page, don't turn in source code, analyze and prove everything, use good English and good logic, and so on. See especially the policies regarding the magic phrases “I don’t know” and “and so on”. If you have *any* questions, post them to the course newsgroup or ask in lecture.
  - This homework tests your familiarity with prerequisite material—basic data structures, big-Oh notation, recurrences, discrete probability, and most importantly, induction—to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Chapters 1–10 of CLRS should be sufficient review, but you may also want consult your discrete mathematics and data structures textbooks.
  - Every homework will have five required problems and one extra-credit problem. Each numbered problem is worth 10 points.
- 

#	1	2	3	4	5	6*	Total
Score							
Grader							

1. Sort the functions in each box from asymptotically smallest to asymptotically largest, indicating ties if there are any. You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway, just for practice. Don't merge the lists together.

To simplify your answers, write  $f(n) \ll g(n)$  to mean  $f(n) = o(g(n))$ , and write  $f(n) \equiv g(n)$  to mean  $f(n) = \Theta(g(n))$ . For example, the functions  $n^2, n, \binom{n}{2}, n^3$  could be sorted either as  $n \ll n^2 \equiv \binom{n}{2} \ll n^3$  or as  $n \ll \binom{n}{2} \equiv n^2 \ll n^3$ .

(a)	$2^{\sqrt{\lg n}}$	$2^{\lg \sqrt{n}}$	$\sqrt{2^{\lg n}}$	$\sqrt{2^{\lg n}}$	$\lg 2^{\sqrt{n}}$	$\lg \sqrt{2^n}$	$\lg \sqrt{2^n}$	$\sqrt{\lg 2^n}$
	$\lg n^{\sqrt{2}}$	$\lg \sqrt{n^2}$	$\lg \sqrt{n^2}$	$\sqrt{\lg n^2}$	$\lg^2 \sqrt{n}$	$\lg^{\sqrt{2}} n$	$\sqrt{\lg^2 n}$	$\sqrt{\lg n^2}$

*(b)	$\lg(\sqrt{n}!)$	$\lg(\sqrt{n}!)$	$\sqrt{\lg(n!)}$	$(\lg \sqrt{n})!$	$(\sqrt{\lg n})!$	$\sqrt{(\lg n)!}$
------	------------------	------------------	------------------	-------------------	-------------------	-------------------

[Hint: Use Stirling's approximation for factorials:  $n! \approx n^{n+1/2}/e^n$ ]

2. Solve the following recurrences. State tight asymptotic bounds for each function in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ . Proofs are *not* required; just give us the list of answers. You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway, just for practice. Assume reasonable but nontrivial base cases. If your solution requires specific base cases, state them! Extra credit will be awarded for more exact solutions.

(a)  $A(n) = 9A(n/3) + n^2$

(b)  $B(n) = 2B(n/2) + n/\lg n$

(c)  $C(n) = \frac{2C(n-1)}{C(n-2)}$  [Hint: This is easy!]

(d)  $D(n) = D(n-1) + 1/n$

(e)  $E(n) = E(n/2) + D(n)$

(f)  $F(n) = 2F(\lfloor (n+3)/4 \rfloor - \sqrt{5n \lg n} + 6) + 7\sqrt{n+8} - \lg^9 \lg \lg n + 10^{\lg^* n} - 11/n^{12}$

(g)  $G(n) = 3G(n-1) - 3G(n-2) + G(n-3)$

\*(h)  $H(n) = 4H(n/2) - 4H(n/4) + 1$  [Hint: Careful!]

(i)  $I(n) = I(n/3) + I(n/4) + I(n/6) + I(n/8) + I(n/12) + I(n/24) + n$

★(j)  $J(n) = \sqrt{n} \cdot J(2\sqrt{n}) + n$

[Hint: First solve the secondary recurrence  $j(n) = 1 + j(2\sqrt{n})$ .]

3. Scientists have recently discovered a planet, tentatively named “Ygdrasil”, which is inhabited by a bizarre species called “nertices” (singular “nertex”). All nertices trace their ancestry back to a particular nertex named Rudy. Rudy is still quite alive, as is every one of his many descendants. Nertices reproduce asexually, like bees; each nertex has exactly one parent (except Rudy). There are three different types of nertices—red, green, and blue. The color of each nertex is correlated exactly with the number and color of its children, as follows:

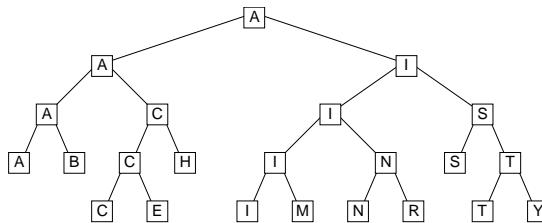
- Each red nertex has two children, exactly one of which is green.
- Each green nertex has exactly one child, which is not green.
- Blue nertices have no children.

In each of the following problems, let  $R$ ,  $G$ , and  $B$  respectively denote the number of red, green, and blue nertices on Ygdrasil.

- (a) Prove that  $B = R + 1$ .
- (b) Prove that either  $G = R$  or  $G = B$ .
- (c) Prove that  $G = B$  if and only if Rudy is green.

4. Algorithms and data structures were developed millions of years ago by the Martians, but not quite in the same way as the recent development here on Earth. Intelligent life evolved independently on Mars’ two moons, Phobos and Deimos.<sup>1</sup> When the two races finally met on the surface of Mars, after thousands of years of separate philosophical, cultural, religious, and scientific development, their disagreements over the proper structure of binary search trees led to a bloody (or more accurately, ichorous) war, ultimately leading to the destruction of all Martian life.

A *Phobian* binary search tree is a full binary tree that stores a set  $X$  of search keys. The root of the tree stores the *smallest* element in  $X$ . If  $X$  has more than one element, then the left subtree stores all the elements less than some pivot value  $p$ , and the right subtree stores everything else. Both subtrees are *nonempty* Phobian binary search trees. The actual pivot value  $p$  is *never* stored in the tree.



A Phobian binary search tree for the set  $\{M, A, R, T, I, N, B, Y, S, E, C, H\}$ .

- (a) Describe and analyze an algorithm  $\text{FIND}(x, T)$  that returns TRUE if  $x$  is stored in the Phobian binary search tree  $T$ , and FALSE otherwise.
- (b) A *Deimoid* binary search tree is almost exactly the same as its Phobian counterpart, except that the *largest* element is stored at the root, and both subtrees are Deimoid binary search trees. Describe and analyze an algorithm to transform an  $n$ -node Phobian binary search tree into a Deimoid binary search tree in  $O(n)$  time, using as little additional space as possible.

<sup>1</sup>Greek for “fear” and “panic”, respectively. Doesn’t that make you feel better?

5. Penn and Teller agree to play the following game. Penn shuffles a standard deck<sup>2</sup> of playing cards so that every permutation is equally likely. Then Teller draws cards from the deck, one at a time without replacement, until he draws the three of clubs ( $3\clubsuit$ ), at which point the remaining undrawn cards instantly burst into flames.

The first time Teller draws a card from the deck, he gives it to Penn. From then on, until the game ends, whenever Teller draws a card whose value is smaller than the last card he gave to Penn, he gives the new card to Penn.<sup>3</sup> To make the rules unambiguous, they agree beforehand that  $A = 1$ ,  $J = 11$ ,  $Q = 12$ , and  $K = 13$ .

- (a) What is the expected number of cards that Teller draws?
- (b) What is the expected *maximum* value among the cards Teller gives to Penn?
- (c) What is the expected *minimum* value among the cards Teller gives to Penn?
- (d) What is the expected number of cards that Teller gives to Penn?

Full credit will be given only for *exact* answers (with correct proofs, of course).

\*6. [Extra credit]<sup>4</sup>

*Lazy binary* is a variant of standard binary notation for representing natural numbers where we allow each “bit” to take on one of three values: 0, 1, or 2. Lazy binary notation is defined inductively as follows.

- The lazy binary representation of zero is 0.
- Given the lazy binary representation of any non-negative integer  $n$ , we can construct the lazy binary representation of  $n + 1$  as follows:
  - (a) increment the rightmost digit;
  - (b) if any digit is equal to 2, replace the rightmost 2 with 0 and increment the digit immediately to its left.

Here are the first several natural numbers in lazy binary notation:

0, 1, 10, 11, 20, 101, 110, 111, 120, 201, 210, 1011, 1020, 1101, 1110, 1111, 1120, 1201, 1210, 2011, 2020, 2101, 2110, 10111, 10120, 10201, 10210, 11011, 11020, 11101, 11110, 11111, 11120, 11201, 11210, 12011, 12020, 12101, 12110, 20111, 20120, 20201, 20210, 21011, 21020, 21101, 21110, 101111, 101120, 101201, 101210, 102011, 102020, 102101, 102110, ...

- (a) Prove that in any lazy binary number, between any two 2s there is at least one 0, and between two 0s there is at least one 2.
- (b) Prove that for any natural number  $N$ , the sum of the digits of the lazy binary representation of  $N$  is exactly  $\lfloor \lg(N + 1) \rfloor$ .

---

<sup>2</sup>In a standard deck of 52 cards, each card has a *suit* in the set  $\{\spadesuit, \heartsuit, \clubsuit, \diamondsuit\}$  and a *value* in the set  $\{A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K\}$ , and every possible suit-value pair appears in the deck exactly once. Actually, to make the game more interesting, Penn and Teller normally use razor-sharp ninja throwing cards.

<sup>3</sup>Specifically, he hurls them from the opposite side of the stage directly into the back of Penn’s right hand.

<sup>4</sup>The “I don’t know” rule does not apply to extra credit problems. There is no such thing as “partial extra credit”.

# CS 373U: Combinatorial Algorithms, Spring 2004

## Homework 1

Due Monday, February 9, 2004 at noon

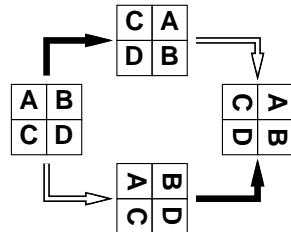
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

- 
- For this and all following homeworks, groups of up to three people can turn in a single solution. Please write *all* your names and NetIDs on *every* page you turn in.
- 

#	1	2	3	4	5	6*	Total
Score							
Grader							

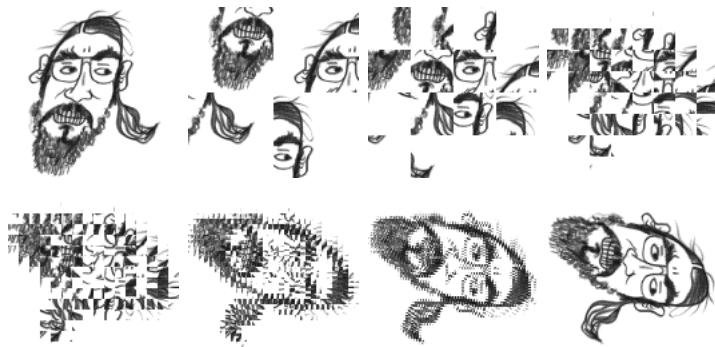
1. Some graphics hardware includes support for an operation called *blit*, or **block transfer**, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

Suppose we want to rotate an  $n \times n$  pixelmap 90° clockwise. One way to do this is to split the pixelmap into four  $n/2 \times n/2$  blocks, move each block to its proper position using a sequence of five blits, and then recursively rotate each block. Alternately, we can first recursively rotate the blocks and blit them into place afterwards.



Two algorithms for rotating a pixelmap.  
Black arrows indicate blitting the blocks into place.  
White arrows indicate recursively rotating the blocks.

The following sequence of pictures shows the first algorithm (blit then recurse) in action.



In the following questions, assume  $n$  is a power of two.

- Prove that both versions of the algorithm are correct. [Hint: If you exploit all the available symmetries, your proof will only be a half of a page long.]
- Exactly* how many blits does the algorithm perform?
- What is the algorithm's running time if each  $k \times k$  blit takes  $O(k^2)$  time?
- What if each  $k \times k$  blit takes only  $O(k)$  time?

2. The traditional Devonian/Cornish drinking song “The Barley Mow” has the following pseudolyrics<sup>1</sup>, where  $\text{container}[i]$  is the name of a container that holds  $2^i$  ounces of beer.<sup>2</sup>

```
BARLEYMOW( $n$ ):
    "Here's a health to the barley-mow, my brave boys,"
    "Here's a health to the barley-mow!"

    "We'll drink it out of the jolly brown bowl,"
    "Here's a health to the barley-mow!"
    "Here's a health to the barley-mow, my brave boys,"
    "Here's a health to the barley-mow!"

for  $i \leftarrow 1$  to  $n$ 
    "We'll drink it out of the container[ $i$ ], boys,"
    "Here's a health to the barley-mow!"
    for  $j \leftarrow i$  downto 1
        "The container[j],"
        "And the jolly brown bowl!"
        "Here's a health to the barley-mow!"
        "Here's a health to the barley-mow, my brave boys,"
        "Here's a health to the barley-mow!"
```

- (a) Suppose each container name  $\text{container}[i]$  is a single word, and you can sing four words a second. How long would it take you to sing  $\text{BARLEYMOW}(n)$ ? (Give a tight asymptotic bound.)
- (b) If you want to sing this song for  $n > 20$ , you’ll have to make up your own container names, and to avoid repetition, these names will get progressively longer as  $n$  increases<sup>3</sup>. Suppose  $\text{container}[n]$  has  $\Theta(\log n)$  syllables, and you can sing six syllables per second. Now how long would it take you to sing  $\text{BARLEYMOW}(n)$ ? (Give a tight asymptotic bound.)
- (c) Suppose each time you mention the name of a container, you drink the corresponding amount of beer: one ounce for the jolly brown bowl, and  $2^i$  ounces for each  $\text{container}[i]$ . Assuming for purposes of this problem that you are at least 21 years old, *exactly* how many ounces of beer would you drink if you sang  $\text{BARLEYMOW}(n)$ ? (Give an *exact* answer, not just an asymptotic bound.)

---

<sup>1</sup>Pseudolyrics are to lyrics as pseudocode is to code.

<sup>2</sup>One version of the song uses the following containers: nipperkin, gill pot, half-pint, pint, quart, pottle, gallon, half-anker, anker, firkin, half-barrel, barrel, hogshead, pipe, well, river, and ocean. Every container in this list is twice as big as its predecessor, except that a firkin is actually 2.25 ankers, and the last three units are just silly.

<sup>3</sup>“We'll drink it out of the hemisemidemiyottapint, boys!”

3. In each of the problems below, you are given a ‘magic box’ that can solve one problem quickly, and you are asked to construct an algorithm that uses the magic box to solve a different problem.
- (a) **3-Coloring:** A graph is *3-colorable* if it is possible to color each vertex red, green, or blue, so that for every edge, its two vertices have two different colors. Suppose you have a magic box that can tell you whether a given graph is 3-colorable in constant time. Describe an algorithm that constructs a 3-coloring of a given graph (if one exists) as quickly as possible.
  - (b) **3SUM:** The 3SUM problem asks, given a set of integers, whether any three elements sum to zero. Suppose you have a magic box that can solve the 3SUM problem in constant time. Describe an algorithm that actually finds, given a set of integers, three elements that sum to zero (if they exist) as quickly as possible.
  - (c) **Traveling Salesman:** A *Hamiltonian cycle* in a graph is a cycle that visits every vertex exactly once. Given a complete graph where every edge has a weight, the *traveling salesman* cycle is the Hamiltonian cycle with minimum total weight; that is, the sum of the weight of the edges is smaller than for any other Hamiltonian cycle. Suppose you have a magic box that can tell you the weight of the traveling salesman cycle of a weighted graph in constant time. Describe an algorithm that actually constructs the traveling salesman cycle of a given weighted graph as quickly as possible.
4. (a) Describe and analyze an algorithm to sort an array  $A[1..n]$  by calling a subroutine SQRTSORT( $k$ ), which sorts the subarray  $A[k+1..k+\lceil\sqrt{n}\rceil]$  in place, given an arbitrary integer  $k$  between 0 and  $n - \lceil\sqrt{n}\rceil$  as input. Your algorithm is *only* allowed to inspect or modify the input array by calling SQRTSORT; in particular, your algorithm must not directly compare, move, or copy array elements. How many times does your algorithm call SQRTSORT in the worst case?
- (b) Prove that your algorithm from part (a) is optimal up to constant factors. In other words, if  $f(n)$  is the number of times your algorithm calls SQRTSORT, prove that no algorithm can sort using  $o(f(n))$  calls to SQRTSORT.
- (c) Now suppose SQRTSORT is implemented recursively, by calling your sorting algorithm from part (a). For example, at the second level of recursion, the algorithm is sorting arrays roughly of size  $n^{1/4}$ . What is the worst-case running time of the resulting sorting algorithm? (To simplify the analysis, assume that the array size  $n$  has the form  $2^{2^k}$ , so that repeated square roots are always integers.)

5. In a previous incarnation, you worked as a cashier in the lost Antarctic colony of Nadira, spending the better part of your day giving change to your customers. Because paper is a very rare and valuable resource on Antarctica, cashiers were required by law to use the fewest bills possible whenever they gave change. Thanks to the numerological predilections of one of its founders, the currency of Nadira, called Dream Dollars, was available in the following denominations: \$1, \$4, \$7, \$13, \$28, \$52, \$91, \$365.<sup>4</sup>

- (a) The greedy change algorithm repeatedly takes the largest bill that does not exceed the target amount. For example, to make \$122 using the greedy algorithm, we first take a \$91 bill, then a \$28 bill, and finally three \$1 bills. Give an example where this greedy algorithm uses more Dream Dollar bills than the minimum possible.
- (b) Describe and analyze a recursive algorithm that computes, given an integer  $k$ , the minimum number of bills needed to make  $k$  Dream Dollars. (Don't worry about making your algorithm fast; just make sure it's correct.)
- (c) Describe a dynamic programming algorithm that computes, given an integer  $k$ , the minimum number of bills needed to make  $k$  Dream Dollars. (This one needs to be fast.)

\*6. [Extra Credit] A popular puzzle called “Lights Out!”, made by Tiger Electronics, has the following description. The game consists of a  $5 \times 5$  array of lighted buttons. By pushing any button, you toggle (on to off, off to on) that light and its four (or fewer) immediate neighbors. The goal of the game is to have every light off at the same time.

We generalize this puzzle to a graph problem. We are given an arbitrary graph with a lighted button at every vertex. Pushing the button at a vertex toggles its light and the lights at all of its neighbors in the graph. A *light configuration* is just a description of which lights are on and which are off. We say that a light configuration is *solvable* if it is possible to get from that configuration to the everything-off configuration by pushing buttons. Some (but clearly not all) light configurations are unsolvable.

- (a) Suppose the graph is just a cycle of length  $n$ . Give a simple and complete characterization of the solvable light configurations in this case. (What we're really looking for here is a *fast* algorithm to decide whether a given configuration is solvable or not.) [Hint: For which cycle lengths is **every** configuration solvable?]
- \*(b) Characterize the set of solvable light configurations when the graph is an arbitrary tree.
- \*(c) A *grid graph* is a graph whose vertices are a regular  $h \times w$  grid of integer points, with edges between immediate vertical or horizontal neighbors. Characterize the set of solvable light configurations for an arbitrary grid graph. (For example, the original Lights Out puzzle can be modeled as a  $5 \times 5$  grid graph.)

---

<sup>4</sup>For more details on the history and culture of Nadira, including images of the various denominations of Dream Dollars, see <http://www.dream-dollars.com>.

# CS 373U: Combinatorial Algorithms, Spring 2004

## Homework 2

Due Friday, February 20, 2004 at noon  
(so you have the whole weekend to study for the midterm)

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

- 
- Starting with this homework, we are changing the way we want you to submit solutions. For each numbered problem, if you use more than one page, staple all those pages together. **Please do *not* staple your entire homework together.** This will allow us to more easily distribute the problems to the graders. Remember to print the name and NetID of every member of your group, as well as the assignment and problem numbers, on every page you submit. You do not need to turn in this cover page.
  - Unless specifically stated otherwise, you can use the fact that the following problems are NP-hard to prove that other problems are NP-hard: Circuit-SAT, 3SAT, Vertex Cover, Maximum Clique, Maximum Independent Set, Hamiltonian Path, Hamiltonian Cycle,  $k$ -Colorability for any  $k \geq 3$ , Traveling Salesman Path, Travelling Salesman Cycle, Subset Sum, Partition, 3Partition, Hitting Set, Minimum Steiner Tree, Minesweeper, Tetris, or any other NP-hard problem described in the lecture notes.
  - This homework is a little harder than the last one. You might want to start early.
- 

#	1	2	3	4	5	6*	Total
Score							
Grader							

1. In lecture on February 5, Jeff presented the following algorithm to compute the length of the longest increasing subsequence of an  $n$ -element array  $A[1..n]$  in  $O(n^2)$  time.

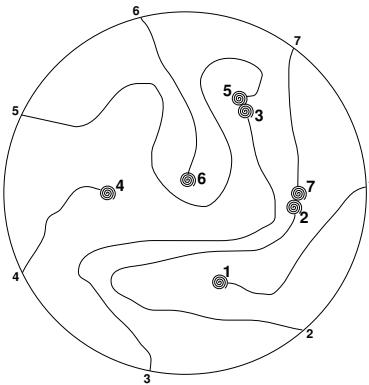
```

LENGTHOFLIS( $A[1..n]$ ):
   $A[n+1] = \infty$ 
  for  $i \leftarrow 1$  to  $n+1$ 
     $L[i] \leftarrow 1$ 
    for  $j \leftarrow 1$  to  $i-1$ 
      if  $A[j] < A[i]$  and  $1 + L[j] < L[i]$ 
         $L[i] \leftarrow 1 + L[j]$ 
  return  $L[n+1] - 1$ 

```

Describe another algorithm for this problem that runs in  $O(n \log n)$  time. [Hint: Use a data structure to replace the inner loop with something faster.]

2. Every year, as part of its annual meeting, the Antarctic Snail Lovers of Union Glacier hold a Round Table Mating Race. A large number of high-quality breeding snails are placed at the edge of a round table. The snails are numbered in order around the table from 1 to  $n$ . The snails wander around the table, each snail leaving a trail of slime behind it. The snails have been specially trained never to fall off the edge of the table or to cross a slime trail (even their own). When any two snails meet, they are declared a breeding pair, removed from the table, and whisked away to a romantic hole in the ground to make little baby snails. Note that some snails may never find a mate, even if  $n$  is even and the race goes on forever.



The end of an Antarctic SLUG race. Snails 1, 4, and 6 never find a mate.  
The organizers must pay  $M[3, 5] + M[2, 7]$ .

For every pair of snails, the Antarctic SLUG race organizers have posted a monetary reward, to be paid to the owners if that pair of snails meets during the Mating Race. Specifically, there is a two-dimensional array  $M[1..n, 1..n]$  posted on the wall behind the Round Table, where  $M[i, j] = M[j, i]$  is the reward if snails  $i$  and  $j$  meet.

Describe and analyze an algorithm to compute the maximum total reward that the organizers could be forced to pay, given the  $n \times n$  array  $M$  as input.

3. Describe and analyze a *polynomial-time* algorithm to determine whether a boolean formula in conjunctive normal form, with exactly *two* literals in each clause, is satisfiable.
4. This problem asks you to prove that four different variants of the minimum spanning tree problem are NP-hard. In each case, the input is a connected undirected graph  $G$  with weighted edges. Each problem considers a certain subset of the possible spanning trees of  $G$ , and asks you to compute the spanning tree with minimum total weight in that subset.
- Prove that finding the minimum-weight *depth first search* tree is NP-hard. (To remind yourself what depth first search is, and why it computes a spanning tree, see Jeff's introductory notes on graphs or Chapter 22 in CLRS.)
  - Suppose a subset  $S$  of the nodes in the input graph are marked. Prove that it is NP-hard to compute the minimum spanning tree whose leaves are all in  $S$ . [Hint: First consider the case  $|S| = 2$ .]
  - Prove that for any integer  $\ell \geq 2$ , it is NP-hard to compute the minimum spanning tree with exactly  $\ell$  leaves. [Hint: First consider the case  $\ell = 2$ .]
  - Prove that for any integer  $d \geq 2$ , it is NP-hard to compute the minimum spanning tree with maximum degree  $d$ . [Hint: First consider the case  $d = 2$ . By now this should start to look familiar.]

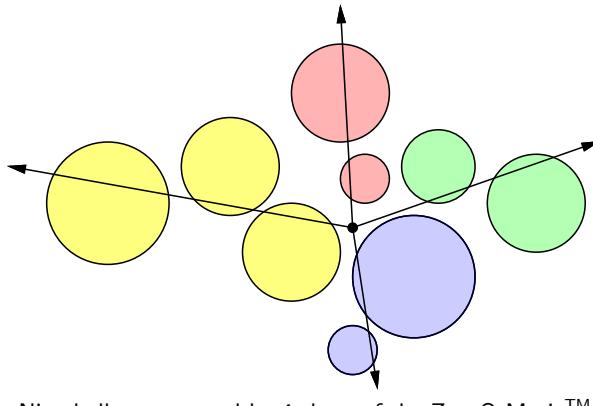
You're welcome to use reductions among these four problems. For example, even if you can't solve part (d), if you can prove that (d) implies (b), you will get full credit for (b). Just don't argue circularly.

5. Consider a machine with a row of  $n$  processors numbered 1 through  $n$ . A *job* is some computational task that occupies a contiguous set of processors for some amount of time. Each processor can work on only one job at a time. Each job is represented by a pair  $J_i = (n_i, t_i)$ , where  $n_i$  is the number of processors required and  $t_i$  is the amount of processing time required to perform the job. A *schedule* for a set of jobs  $\{J_1, \dots, J_m\}$  assigns each job  $J_i$  to some set of  $n_i$  contiguous processors for an interval of  $t_i$  seconds, so that no processor works on more than one job at any time. The *make-span* of a schedule is the time from the start to the finish of all jobs.

The *parallel scheduling problem* asks, given a set of jobs as input, to compute a schedule for those jobs with the smallest possible make-span.

- Prove that the parallel scheduling problem is NP-hard.
- Give an algorithm that computes a 3-approximation of the minimum make-span of a set of jobs in  $O(m \log m)$  time. That is, if the minimum make-span is  $M$ , your algorithm should compute a schedule with make-span at most  $3M$ . You can assume that  $n$  is a power of 2.

- \*6. **[Extra credit]** Suppose you are standing in a field surrounded by several large balloons. You want to use your brand new Acme Brand Zap-O-Matic™ to pop all the balloons, without moving from your current location. The Zap-O-Matic™ shoots a high-powered laser beam, which pops all the balloons it hits. Since each shot requires enough energy to power a small country for a year, you want to fire as few shots as possible.



The *minimum zap* problem can be stated more formally as follows. Given a set  $C$  of  $n$  circles in the plane, each specified by its radius and the  $(x, y)$  coordinates of its center, compute the minimum number of rays from the origin that intersect every circle in  $C$ . Your goal is to find an efficient algorithm for this problem.

- (a) Describe and analyze a greedy algorithm whose output is within 1 of optimal. That is, if  $m$  is the minimum number of rays required to hit every circle in the input, then your greedy algorithm must output either  $m$  or  $m + 1$ . (Of course, you must prove this fact.)
- (b) Describe an algorithm that solves the minimum zap problem in  $O(n^2)$  time.
- \*(c) Describe an algorithm that solves the minimum zap problem in  $O(n \log n)$  time.

Assume you have a subroutine  $\text{INTERSECTS}(r, c)$  that determines, in  $O(1)$  time, whether a ray  $r$  intersects a circle  $c$ . It's not that hard to write this subroutine, but it's not the interesting part of the problem.

# CS 373U: Combinatorial Algorithms, Spring 2004

## Homework 3

Due Friday, March 12, 2004 at noon

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

- 
- For each numbered problem, if you use more than one page, staple all those pages together. **Please do *not* staple your entire homework together.** This will allow us to more easily distribute the problems to the graders. Remember to print the name and NetID of every member of your group, as well as the assignment and problem numbers, on every page you submit. You do not need to turn in this cover page.
  - This homework is challenging. You might want to start early.
- 

#	1	2	3	4	5	6*	Total
Score							
Grader							

1. Let  $S$  be a set of  $n$  points in the plane. A point  $p$  in  $S$  is called *Pareto-optimal* if no other point in  $S$  is both above and to the right of  $p$ .
  - (a) Describe and analyze a deterministic algorithm that computes the Pareto-optimal points in  $S$  in  $O(n \log n)$  time.
  - (b) Suppose each point in  $S$  is chosen independently and uniformly at random from the unit square  $[0, 1] \times [0, 1]$ . What is the *exact* expected number of Pareto-optimal points in  $S$ ?
  
  
  
  
2. Suppose we have an oracle  $\text{RANDOM}(k)$  that returns an integer chosen independently and uniformly at random from the set  $\{1, \dots, k\}$ , where  $k$  is the input parameter;  $\text{RANDOM}$  is our only source of random bits. We wish to write an efficient function  $\text{RANDOMPERMUTATION}(n)$  that returns a permutation of the integers  $\langle 1, \dots, n \rangle$  chosen uniformly at random.
  - (a) Consider the following implementation of  $\text{RANDOMPERMUTATION}$ .

```
RANDOMPERMUTATION( $n$ ):
  for  $i = 1$  to  $n$ 
     $\pi[i] \leftarrow \text{NULL}$ 
  for  $i = 1$  to  $n$ 
     $j \leftarrow \text{RANDOM}(n)$ 
    while ( $\pi[j] \neq \text{NULL}$ )
       $j \leftarrow \text{RANDOM}(n)$ 
     $\pi[j] \leftarrow i$ 
  return  $\pi$ 
```

Prove that this algorithm is correct. Analyze its expected runtime.

- (b) Consider the following partial implementation of  $\text{RANDOMPERMUTATION}$ .

```
RANDOMPERMUTATION( $n$ ):
  for  $i = 1$  to  $n$ 
     $A[i] \leftarrow \text{RANDOM}(n)$ 
   $\pi \leftarrow \text{SOMEFUNCTION}(A)$ 
  return  $\pi$ 
```

Prove that if the subroutine  $\text{SOMEFUNCTION}$  is deterministic, then this algorithm cannot be correct. [Hint: There is a one-line proof.]

- (c) Consider a correct implementation of  $\text{RANDOMPERMUTATION}(n)$  with the following property: whenever it calls  $\text{RANDOM}(k)$ , the argument  $k$  is at most  $m$ . Prove that this algorithm **always** calls  $\text{RANDOM}$  at least  $\Omega(\frac{n \log n}{\log m})$  times.
- (d) Describe and analyze an implementation of  $\text{RANDOMPERMUTATION}$  that runs in expected worst-case time  $O(n)$ .

3. A *meldable priority queue* stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:

- **MAKEQUEUE**: Return a new priority queue containing the empty set.
- **FINDMIN( $Q$ )**: Return the smallest element of  $Q$  (if any).
- **DELETEMIN( $Q$ )**: Remove the smallest element in  $Q$  (if any).
- **INSERT( $Q, x$ )**: Insert element  $x$  into  $Q$ , if it is not already there.
- **DECREASEKEY( $Q, x, y$ )**: Replace an element  $x \in Q$  with a smaller key  $y$ . (If  $y > x$ , the operation fails.) The input is a pointer directly to the node in  $Q$  containing  $x$ .
- **DELETE( $Q, x$ )**: Delete the element  $x \in Q$ . The input is a pointer directly to the node in  $Q$  containing  $x$ .
- **MELD( $Q_1, Q_2$ )**: Return a new priority queue containing all the elements of  $Q_1$  and  $Q_2$ ; this operation destroys  $Q_1$  and  $Q_2$ .

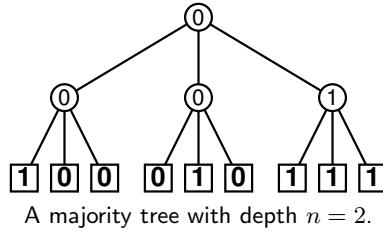
A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm:

```

MELD( $Q_1, Q_2$ ):
    if  $Q_1$  is empty return  $Q_2$ 
    if  $Q_2$  is empty return  $Q_1$ 
    if  $key(Q_1) > key(Q_2)$ 
        swap  $Q_1 \leftrightarrow Q_2$ 
    with probability  $1/2$ 
         $left(Q_1) \leftarrow \text{MELD}(left(Q_1), Q_2)$ 
    else
         $right(Q_1) \leftarrow \text{MELD}(right(Q_1), Q_2)$ 
    return  $Q_1$ 
```

- (a) Prove that for *any* heap-ordered binary trees  $Q_1$  and  $Q_2$  (not just those constructed by the operations listed above), the expected running time of  $\text{MELD}(Q_1, Q_2)$  is  $O(\log n)$ , where  $n = |Q_1| + |Q_2|$ . [Hint: How long is a random root-to-leaf path in an  $n$ -node binary tree if each left/right choice is made with equal probability?]
- (b) [Extra credit] Prove that  $\text{MELD}(Q_1, Q_2)$  runs in  $O(\log n)$  time with high probability.
- (c) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and  $O(1)$  additional time. (This implies that every operation takes  $O(\log n)$  time with high probability.)

4. A *majority tree* is a complete binary tree with depth  $n$ , where every leaf is labeled either 0 or 1. The *value* of a leaf is its label; the *value* of any internal node is the majority of the values of its three children. Consider the problem of computing the value of the root of a majority tree, given the sequence of  $3^n$  leaf labels as input. For example, if  $n = 2$  and the leaves are labeled 1, 0, 0, 0, 1, 0, 1, 1, 1, the root has value 0.



- (a) Prove that *any* deterministic algorithm that computes the value of the root of a majority tree *must* examine every leaf. [Hint: Consider the special case  $n = 1$ . Recurse.]
- (b) Describe and analyze a randomized algorithm that computes the value of the root in worst-case expected time  $O(c^n)$  for some constant  $c < 3$ . [Hint: Consider the special case  $n = 1$ . Recurse.]
5. Suppose  $n$  lights labeled  $0, \dots, n - 1$  are placed clockwise around a circle. Initially, each light is set to the off position. Consider the following random process.

```

LIGHTTHECIRCLE( $n$ ):
   $k \leftarrow 0$ 
  turn on light 0
  while at least one light is off
    with probability  $1/2$ 
       $k \leftarrow (k + 1) \bmod n$ 
    else
       $k \leftarrow (k - 1) \bmod n$ 
    if light  $k$  is off, turn it on
  
```

Let  $p(i, n)$  be the probability that light  $i$  is the last to be turned on by  $\text{LIGHTTHECIRCLE}(n, 0)$ . For example,  $p(0, 2) = 0$  and  $p(1, 2) = 1$ . Find an exact closed-form expression for  $p(i, n)$  in terms of  $n$  and  $i$ . Prove your answer is correct.

6. **[Extra Credit]** Let  $G$  be a *bipartite* graph on  $n$  vertices. Each vertex  $v$  has an associated set  $C(v)$  of  $\lg 2n$  colors with which  $v$  is compatible. We wish to find a coloring of the vertices in  $G$  so that every vertex  $v$  is assigned a color from its set  $C(v)$  and no edge has the same color at both ends. Describe and analyze a randomized algorithm that computes such a coloring in expected worst-case time  $O(n \log^2 n)$ . [Hint: For any events  $A$  and  $B$ ,  $\Pr[A \cup B] \leq \Pr[A] + \Pr[B]$ .]

# CS 373U: Combinatorial Algorithms, Spring 2004

## Homework 4

Due Friday, April 2, 2004 at noon

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

- 
- For each numbered problem, if you use more than one page, staple all those pages together. **Please do *not* staple your entire homework together.** This will allow us to more easily distribute the problems to the graders. Remember to print the name and NetID of every member of your group, as well as the assignment and problem numbers, on every page you submit. You do not need to turn in this cover page.
  - As with previous homeworks, we strongly encourage you to begin early.
- 

#	1	2	3	4	5	6*	Total
Score							
Grader							

1. Suppose we can insert or delete an element into a hash table in constant time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:

- After an insertion, if the table is more than  $3/4$  full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
- After a deletion, if the table is less than  $1/4$  full, we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do *not* use the potential method (like CLRS does); there is a much easier solution.

2. Remember the difference between stacks and queues? Good.

- (a) Describe how to implement a queue using two stacks and  $O(1)$  additional memory, so that the amortized time for any enqueue or dequeue operation is  $O(1)$ . The *only* access you have to the stacks is through the standard subroutines PUSH and POP.
- (b) A *quack* is a data structure combining properties of both stacks and queues. It can be viewed as a list of elements written left to right such that three operations are possible:
  - **Push:** add a new item to the left end of the list;
  - **Pop:** remove the item on the left end of the list;
  - **Pull:** remove the item on the right end of the list.

Implement a quack using *three* stacks and  $O(1)$  additional memory, so that the amortized time for any push, pop, or pull operation is  $O(1)$ . Again, you are *only* allowed to access the stacks through the standard functions PUSH and POP.

3. Some applications of binary search trees attach a *secondary data structure* to each node in the tree, to allow for more complicated searches. Maintaining these secondary structures usually complicates algorithms for keeping the top-level search tree balanced.

Suppose we have a binary search tree  $T$  where every node  $v$  stores a secondary structure of size  $O(|v|)$ , where  $|v|$  denotes the number of descendants of  $v$  in  $T$ . Performing a rotation at a node  $v$  in  $T$  now requires  $O(|v|)$  time, because we have to rebuild one of the secondary structures.

- (a) [1 pt] Overall, how much space does this data structure use in the worst case?
- (b) [1 pt] How much space does this structure use if the top-level search tree  $T$  is balanced?
- (c) [2 pt] Suppose  $T$  is a splay tree. Prove that the *amortized* cost of a splay (and therefore of a search, insertion, or deletion) is  $\Omega(n)$ . [*Hint: This is easy!*]
- (d) [3 pts] Now suppose  $T$  is a scapegoat tree, and that rebuilding the subtree rooted at  $v$  requires  $\Theta(|v| \log |v|)$  time (because we also have to rebuild all the secondary structures). What is the *amortized* cost of inserting a new element into  $T$ ?
- (e) [3 pts] Finally, suppose  $T$  is a treap. What's the worst-case *expected* time for inserting a new element into  $T$ ?

4. In a *dirty* binary search tree, each node is labeled either *clean* or *dirty*. The lazy deletion scheme used for scapegoat trees requires us to *purge* the search tree, keeping all the clean nodes and deleting all the dirty nodes, as soon as half the nodes become dirty. In addition, the purged tree should be perfectly balanced.

Describe an algorithm to purge an arbitrary  $n$ -node dirty binary search tree in  $O(n)$  time, using only  $O(\log n)$  additional memory. For 5 points extra credit, reduce the additional memory requirement to  $O(1)$  *without repeating an old CS373 homework solution*.<sup>1</sup>

5. This problem considers a variant of the lazy binary notation introduced in the extra credit problem from Homework 0. In a *doubly lazy binary number*, each bit can take one of *four* values:  $-1$ ,  $0$ ,  $1$ , or  $2$ . The only legal representation for zero is  $0$ . To increment, we add  $1$  to the least significant bit, then carry the rightmost  $2$  (if any). To decrement, we subtract  $1$  from the least significant bit, and then borrow the rightmost  $-1$  (if any).

LAZYINCREMENT( $B[0..n]$ ):

```

 $B[0] \leftarrow B[0] + 1$ 
for  $i \leftarrow 1$  to  $n - 1$ 
  if  $B[i] = 2$ 
     $B[i] \leftarrow 0$ 
     $B[i + 1] \leftarrow B[i + 1] + 1$ 
return
  
```

LAZYDECREMENT( $B[0..n]$ ):

```

 $B[0] \leftarrow B[0] - 1$ 
for  $i \leftarrow 1$  to  $n - 1$ 
  if  $B[i] = -1$ 
     $B[i] \leftarrow 1$ 
     $B[i + 1] \leftarrow B[i + 1] - 1$ 
return
  
```

For example, here is a doubly lazy binary count from zero up to twenty and then back down to zero. The bits are written with the least significant bit (*i.e.*,  $B[0]$ ) on the right. For succinctness, we write  $\pm$  instead of  $-1$  and omit any leading  $0$ 's.

```

0  $\xrightarrow{++}$  1  $\xrightarrow{++}$  10  $\xrightarrow{++}$  11  $\xrightarrow{++}$  20  $\xrightarrow{++}$  101  $\xrightarrow{++}$  110  $\xrightarrow{++}$  111  $\xrightarrow{++}$  120  $\xrightarrow{++}$  201  $\xrightarrow{++}$  210
 $\xrightarrow{++}$  1011  $\xrightarrow{++}$  1020  $\xrightarrow{++}$  1101  $\xrightarrow{++}$  1110  $\xrightarrow{++}$  1111  $\xrightarrow{++}$  1120  $\xrightarrow{++}$  1201  $\xrightarrow{++}$  1210  $\xrightarrow{++}$  2011  $\xrightarrow{++}$  2020
 $\xrightarrow{--}$  2011  $\xrightarrow{--}$  2010  $\xrightarrow{--}$  2001  $\xrightarrow{--}$  2000  $\xrightarrow{--}$  20 $\pm$ 1  $\xrightarrow{--}$  2 $\pm$ 10  $\xrightarrow{--}$  2 $\pm$ 01  $\xrightarrow{--}$  1100  $\xrightarrow{--}$  11 $\pm$ 1  $\xrightarrow{--}$  1010
 $\xrightarrow{--}$  1001  $\xrightarrow{--}$  1000  $\xrightarrow{--}$  10 $\pm$ 1  $\xrightarrow{--}$  1 $\pm$ 10  $\xrightarrow{--}$  1 $\pm$ 01  $\xrightarrow{--}$  100  $\xrightarrow{--}$  1 $\pm$ 1  $\xrightarrow{--}$  10  $\xrightarrow{--}$  1  $\xrightarrow{--}$  0
  
```

Prove that for any intermixed sequence of increments and decrements of a doubly lazy binary number, starting with  $0$ , the amortized time for each operation is  $O(1)$ . Do *not* assume, as in the example above, that all the increments come before all the decrements.

---

<sup>1</sup>That was for a slightly different problem anyway.

6. [Extra credit] My wife is teaching a class<sup>2</sup> where students work on homeworks in groups of exactly three people, subject to the following rule: *No two students may work together on more than one homework.* At the beginning of the semester, it was easy to find homework groups, but as the course progresses, it is becoming harder and harder to find a legal grouping. Finally, in despair, she decides to ask a computer scientist to write a program to find the groups for her.
- (a) We can formalize this homework-group-assignment problem as follows. The input is a graph, where the vertices are the  $n$  students, and two students are joined by an edge if they have not yet worked together. Every node in this graph has the same degree; specifically, if there have been  $k$  homeworks so far, each student is connected to exactly  $n - 1 - 2k$  other students. The goal is to find  $n/3$  disjoint triangles in the graph, or conclude that no such triangles exist. Prove (or disprove!) that this problem is NP-hard.
  - (b) Suppose my wife had planned ahead and assigned groups for every homework at the beginning of the semester. How many homeworks can she assign, as a function of  $n$ , without violating the no-one-works-together-twice rule? Prove the best upper and lower bounds you can. To prove the upper bound, describe an algorithm that actually assigns the groups for each homework.

---

<sup>2</sup>Math 302: Non-Euclidean Geometry. Problem 1 from last week's homework assignment: "Invert Mr. Happy."

# CS 373U: Combinatorial Algorithms, Spring 2004

## Homework 5

Due Wednesday, April 28, 2004 at noon

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

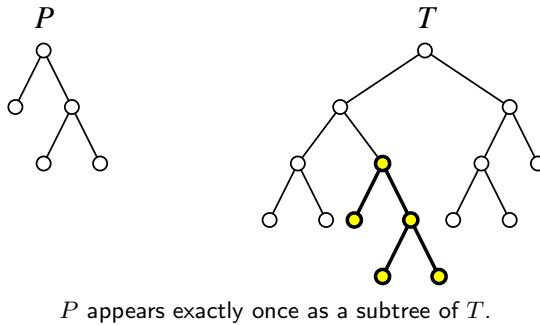
- 
- For each numbered problem, if you use more than one page, staple all those pages together. **Please do *not* staple your entire homework together.** This will allow us to more easily distribute the problems to the graders. Remember to print the name and NetID of every member of your group, as well as the assignment and problem numbers, on every page you submit. You do not need to turn in this cover page.
  - As with previous homeworks, we strongly encourage you to begin early.
  - This will be the last graded homework.
- 

#	1	2	3	4	5	Total
Score						
Grader						

1. (a) Prove that every graph with the same number of vertices and edges has a cycle.  
(b) Prove that every graph with exactly two fewer edges than vertices is disconnected.

Both proofs should be entirely self-contained. In particular, they should not use the word “tree” or any properties of trees that you saw in CS 225 or CS 273.

2. A *palindrome* is a string of characters that is exactly the same as its reversal, like X, FOOF, RADAR, or AMANAPLANACATACANALPANAMA.
  - (a) Describe and analyze an algorithm to compute the longest *prefix* of a given string that is a palindrome. For example, the longest palindrome prefix of RADARDETECTAR is RADAR, and the longest palindrome prefix of ALGORITHMSHOMWORK is the single letter A.
  - (b) Describe and analyze an algorithm to compute a longest *subsequence* of a given string that is a palindrome. For example, the longest palindrome subsequence of RADARDETECTAR is RAETEAR (or RADADAR or RADRDAR or RATETAR or RATCTAR), and the longest palindrome subsequence of ALGORITHMSHOMWORK is OMOMO (or RMHMR or OHSHO or...).
3. Describe and analyze an algorithm that decides, given two binary trees  $P$  and  $T$ , whether  $T$  is a subtree of  $P$ . There is no actual *data* stored in the nodes—these are not binary *search* trees or binary *heaps*. You are only trying to match the *shape* of the trees.

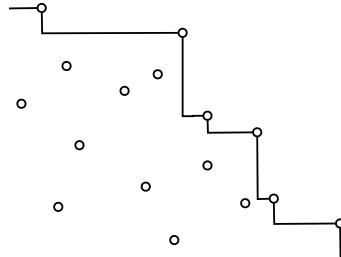


4. Describe and analyze an algorithm that computes the *second* smallest spanning tree of a given connected, undirected, edge-weighted graph.
5. Show that if the input graph is allowed to have negative edges (but no negative cycles), Dijkstra's algorithm<sup>1</sup> runs in exponential time in the worst case. Specifically, describe how to construct, for every integer  $n$ , a weighted directed graph  $G_n$  without negative cycles that forces Dijkstra's algorithm to perform  $\Omega(2^n)$  relaxation steps. Give your description in the form of an algorithm! [Hint: Towers of Hanoi.]

---

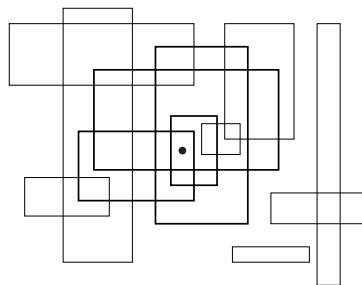
<sup>1</sup>This refers to the version of Dijkstra's algorithm described in Jeff's lecture notes. The version in CLRS is always fast, but sometimes gives incorrect results for graphs with negative edges.

1. Let  $P$  be a set of  $n$  points in the plane. Recall that a point  $p \in P$  is *Pareto-optimal* if no other point is both above and to the right of  $p$ . Intuitively, the sorted sequence of Pareto-optimal points describes a *staircase* with all the points in  $P$  below and to the left. Your task is to describe some algorithms that compute this staircase.



The staircase of a set of points

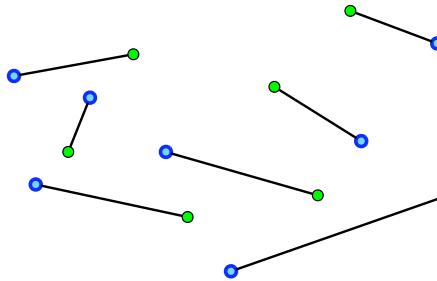
- (a) Describe an algorithm to compute the staircase of  $P$  in  $O(nh)$  time, where  $h$  is the number of Pareto-optimal points.
  - (b) Describe a divide-and-conquer algorithm to compute the staircase of  $P$  in  $O(n \log n)$  time. [Hint: I know of at least two different ways to do this.]
  - \*(c) Describe an algorithm to compute the staircase of  $P$  in  $O(n \log h)$  time, where  $h$  is the number of Pareto-optimal points. [Hint: I know of at least two different ways to do this.]
  - (d) Finally, suppose the points in  $P$  are already given in sorted order from left to right. Describe an algorithm to compute the staircase of  $P$  in  $O(n)$  time. [Hint: I know of at least two different ways to do this.]
2. Let  $R$  be a set of  $n$  rectangles in the plane.
- (a) Describe and analyze a plane sweep algorithm to decide, in  $O(n \log n)$  time, whether any two rectangles in  $R$  intersect.
  - \*(b) The *depth* of a point is the number of rectangles in  $R$  that contain that point. The *maximum depth* of  $R$  is the maximum, over all points  $p$  in the plane, of the depth of  $p$ . Describe a plane sweep algorithm to compute the maximum depth of  $R$  in  $O(n \log n)$  time.



A point with depth 4 in a set of rectangles.

- (c) Describe and analyze a polynomial-time reduction from the maximum depth problem in part (b) to MAXCLIQUE: Given a graph  $G$ , how large is the largest clique in  $G$ ?
- (d) MAXCLIQUE is NP-hard. So does your reduction imply that P=NP? Why or why not?

3. Let  $G$  be a set of  $n$  green points, called “Ghosts”, and let  $B$  be a set of  $n$  blue points, called “ghostBusters”, so that no three points lie on a common line. Each Ghostbuster has a gun that shoots a stream of particles in a straight line until it hits a ghost. The Ghostbusters want to kill all of the ghosts at once, by having each Ghostbuster shoot a different ghost. It is **very important** that the streams do not cross.



A non-crossing matching between 7 ghosts and 7 Ghostbusters

- (a) Prove that the Ghostbusters can succeed. More formally, prove that there is a collection of  $n$  non-intersecting line segments, each joining one point in  $G$  to one point in  $B$ . [Hint: Think about the set of joining segments with minimum total length.]
- (b) Describe and analyze an algorithm to find a line  $\ell$  that passes through one ghost and one Ghostbuster, so that same number of ghosts as Ghostbusters are above  $\ell$ .
- \*(c) Describe and analyze an algorithm to find a line  $\ell$  such that exactly half the ghosts and exactly half the Ghostbusters are above  $\ell$ . (Assume  $n$  is even.)
- (d) Using your algorithm for part (b) or part (c) as a subroutine, describe and analyze an algorithm to find the line segments described in part (a). (Assume  $n$  is a power of two if necessary.)

**Spengler:** *Don't cross the streams.*

**Venkman:** *Why?*

**Spengler:** *It would be bad.*

**Venkman:** *I'm fuzzy on the whole good/bad thing. What do you mean "bad"?*

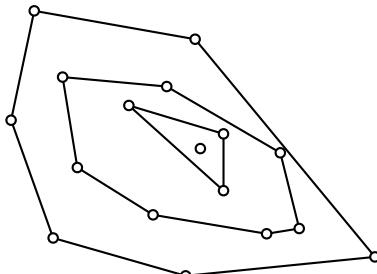
**Spengler:** *Try to imagine all life as you know it stopping instantaneously and every molecule in your body exploding at the speed of light.*

**Stantz:** *Total protonic reversal!*

**Venkman:** *That's bad. Okay. Alright, important safety tip, thanks Egon.*

— Dr. Egon Spengler (Harold Ramis), Dr. Peter Venkman (Bill Murray),  
and Dr. Raymond Stanz (Dan Aykroyd), *Ghostbusters*, 1984

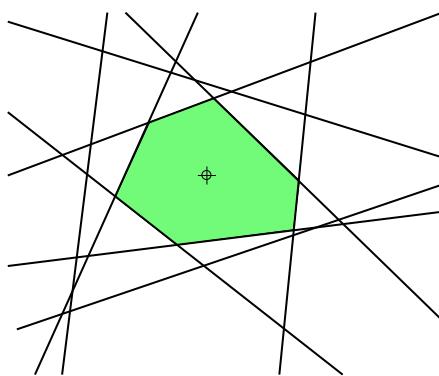
4. The *convex layers* of a point set  $P$  consist of a series of nested convex polygons. The convex layers of the empty set are empty. Otherwise, the first layer is just the convex hull of  $P$ , and the remaining layers are the convex layers of the points that are not on the convex hull of  $P$ .



The convex layers of a set of points.

Describe and analyze an efficient algorithm to compute the convex layers of a given  $n$ -point set. For full credit, your algorithm should run in  $O(n^2)$  time.

5. Suppose we are given a set of  $n$  lines in the plane, where none of the lines passes through the origin  $(0, 0)$  and at most two lines intersect at any point. These lines divide the plane into several convex polygonal regions, or *cells*. Describe and analyze an efficient algorithm to compute the cell containing the origin. The output should be a doubly-linked list of the cell's vertices. [Hint: There are literally dozens of solutions. One solution is to reduce this problem to the convex hull problem. Every other solution looks like a convex hull algorithm.]



The cell containing the origin in an arrangement of lines.

Write your answers in the separate answer booklet.

1. **Multiple Choice:** Each question below has one of the following answers.

A:  $\Theta(1)$     B:  $\Theta(\log n)$     C:  $\Theta(n)$     D:  $\Theta(n \log n)$     E:  $\Theta(n^2)$     X: I don't know.

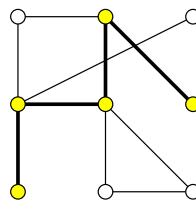
For each question, write the letter that corresponds to your answer. You do not need to justify your answers. Each correct answer earns you 1 point. Each X earns you  $\frac{1}{4}$  point. **Each incorrect answer costs you  $\frac{1}{2}$  point.** Your total score will be rounded **down** to an integer. Negative scores will be rounded up to zero.

- (a) What is  $\sum_{i=1}^n \lg i$ ?
- (b) What is  $\sum_{i=1}^{\lg n} i 2^i$ ?
- (c) How many decimal digits are required to write the  $n$ th Fibonacci number?
- (d) What is the solution of the recurrence  $T(n) = 4T(n/8) + n \log n$ ?
- (e) What is the solution of the recurrence  $T(n) = T(n - 3) + \frac{5}{n}$ ?
- (f) What is the solution of the recurrence  $T(n) = 5T\left(\lceil \frac{n+13}{3} \rceil + \lfloor \sqrt{n} \rfloor\right) + (10n - 7)^2 - \frac{\lg^3 n}{\lg \lg n}$ ?
- (g) How long does it take to construct a Huffman code, given an array of  $n$  character frequencies as input?
- (h) How long does it take to sort an array of size  $n$  using quicksort?
- (i) Given an unsorted array  $A[1..n]$ , how long does it take to construct a binary search tree for the elements of  $A$ ?
- (j) A train leaves Chicago at 8:00pm and travels south at 75 miles per hour. Another train leaves New Orleans at 1:00pm and travels north at 60 miles per hour. The conductors of both trains are playing a game of chess over the phone. After each player moves, the other player must move before his train has traveled five miles. How many moves do the two players make before their trains pass each other (somewhere near Memphis)?

2. Describe and analyze efficient algorithms to solve the following problems:

- (a) Given a set of  $n$  integers, does it contain a pair of elements  $a, b$  such that  $a + b = 0$ ?
- (b) Given a set of  $n$  integers, does it contain three elements  $a, b, c$  such that  $a + b = c$ ?

3. A *tonian path* in a graph  $G$  is a simple path in  $G$  that visits more than half of the vertices of  $G$ . (Intuitively, a tonian path is “most of a Hamiltonian path”.) Prove that it is NP-hard to determine whether or not a given graph contains a tonian path.



A tonian path in a 9-vertex graph.

4. *Vankin's Mile* is a solitaire game played on an  $n \times n$  square grid. The player starts by placing a token on any square of the grid. Then on each turn, the player moves the token either one square to the right or one square down. The game ends when player moves the token off the edge of the board. Each square of the grid has a numerical value, which could be positive, negative, or zero. The player starts with a score of zero; whenever the token lands on a square, the player adds its value to his score. The object of the game is to score as many points as possible.

For example, given the grid below, the player can score  $8 - 6 + 7 - 3 + 4 = 10$  points by placing the initial token on the 8 in the second row, and then moving down, down, right, down, down. (This is *not* the best possible score for these values.)

-1	7	-8	10	-5
-4	-9	<b>8</b>	-6	0
5	-2	<b>-6</b>	-6	7
-7	4	<b>7</b> $\Rightarrow$ <b>-3</b>	-3	-3
7	1	-6	<b>4</b>	-9

Describe and analyze an algorithm to compute the maximum possible score for a game of Vankin's Mile, given the  $n \times n$  array of values as input.

5. Suppose you are given two sorted arrays  $A[1..m]$  and  $B[1..n]$  and an integer  $k$ . Describe an algorithm to find the  $k$ th smallest element in the union of  $A$  and  $B$  in  $\Theta(\log(m+n))$  time. For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \quad B[1..5] = [2, 5, 8, 17, 19] \quad k = 6$$

your algorithm should return 8. You can assume that the arrays contain no duplicates. [Hint: What can you learn from comparing one element of  $A$  to one element of  $B$ ?]

Write your answers in the separate answer booklet.

1. **Multiple Choice:** Each question below has one of the following answers.

A:  $\Theta(1)$     B:  $\Theta(\log n)$     C:  $\Theta(n)$     D:  $\Theta(n \log n)$     E:  $\Theta(n^2)$     X: I don't know.

For each question, write the letter that corresponds to your answer. You do not need to justify your answers. Each correct answer earns you 1 point. Each X earns you  $\frac{1}{4}$  point. **Each incorrect answer costs you  $\frac{1}{2}$  point.** Your total score will be rounded **down** to an integer. Negative scores will be rounded up to zero.

- What is  $\sum_{i=1}^n \frac{n}{i}$ ?
- What is  $\sum_{i=1}^{\lg n} 4^i$ ?
- How many bits are required to write  $n!$  in binary?
- What is the solution of the recurrence  $T(n) = 4T(n/2) + n \log n$ ?
- What is the solution of the recurrence  $T(n) = T(n - 3) + \frac{5}{n}$ ?
- What is the solution of the recurrence  $T(n) = 9T\left(\lceil \frac{n+13}{3} \rceil\right) + 10n - 7\sqrt{n} - \frac{\lg^3 n}{\lg \lg n}$ ?
- How long does it search for a value in an  $n$ -node binary search tree?
- Given a sorted array  $A[1..n]$ , how long does it take to construct a binary search tree for the elements of  $A$ ?
- How long does it take to construct a Huffman code, given an array of  $n$  character frequencies as input?
- A train leaves Chicago at 8:00pm and travels south at 75 miles per hour. Another train leaves New Orleans at 1:00pm and travels north at 60 miles per hour. The conductors of both trains are playing a game of chess over the phone. After each player moves, the other player must move before his train has traveled five miles. How many moves do the two players make before their trains pass each other (somewhere near Memphis)?

2. Describe and analyze an algorithm to find the length of the longest substring that appears both forward and backward in an input string  $T[1..n]$ . The forward and backward substrings must not overlap. Here are several examples:

- Given the input string ALGORITHM, your algorithm should return 0.
- Given the input string RECURSION, your algorithm should return 1, for the substring R.
- Given the input string REDIVIDE, your algorithm should return 3, for the substring EDI. (The forward and backward substrings must not overlap!)
- Given the input string DYNAMICPROGRAMMINGMANYTIMES, your algorithm should return 4, for the substring YNAM.

For full credit, your algorithm should run in  $O(n^2)$  time.

3. The *median* of a set of size  $n$  is its  $\lceil n/2 \rceil$ th largest element, that is, the element that is as close as possible to the middle of the set in sorted order. It's quite easy to find the median of a set in  $O(n \log n)$  time—just sort the set and look in the middle—but you (correctly!) think that you can do better.

During your lifelong quest for a faster median-finding algorithm, you meet and befriend the Near-Middle Fairy. Given any set  $X$ , the Near-Middle Fairy can find an element  $m \in X$  that is *near* the middle of  $X$  in  $O(1)$  time. Specifically, at least a third of the elements of  $X$  are smaller than  $m$ , and at least a third of the elements of  $X$  are larger than  $m$ .

Describe and analyze an algorithm to find the median of a set in  $O(n)$  time if you are allowed to ask the Near-Middle Fairy for help. [Hint: You may need the PARTITION subroutine from QUICKSORT.]

4. SUBSETSUM and PARTITION are two closely related NP-hard problems.

- SUBSETSUM: Given a set  $X$  of integers and an integer  $k$ , does  $X$  have a subset whose elements sum up to  $k$ ?
- PARTITION: Given a set  $X$  of integers and an integer  $k$ , can  $X$  be partitioned into two subsets whose sums are equal?
  - (a) Describe and analyze a polynomial-time reduction from SUBSETSUM to PARTITION.
  - (b) Describe and analyze a polynomial-time reduction from PARTITION to SUBSETSUM.

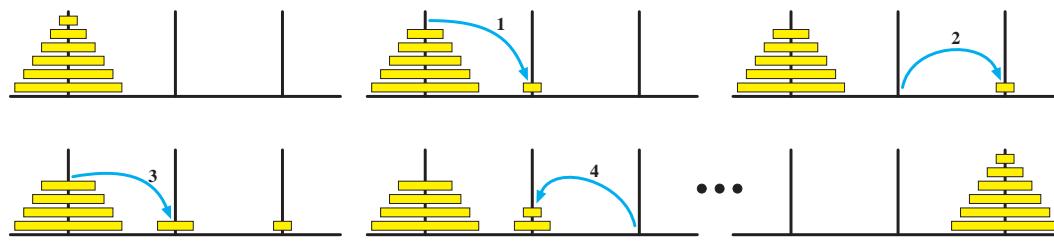
5. Describe and analyze efficient algorithms to solve the following problems:

- (a) Given a set of  $n$  integers, does it contain a pair of elements  $a, b$  such that  $a + b = 0$ ?
- (b) Given a set of  $n$  integers, does it contain three elements  $a, b, c$  such that  $a + b + c = 0$ ?

**Write your answers in the separate answer booklet.**

1. In the well-known *Tower of Hanoi* problem, we have three spikes, one of which has a tower of  $n$  disks of different sizes, stacked with smaller disks on top of larger ones. In a single step, we are allowed to take the top disk on any spike and move it to the top of another spike. We are never allowed to place a larger disk on top of a smaller one. Our goal is to move all the disks from one spike to another.

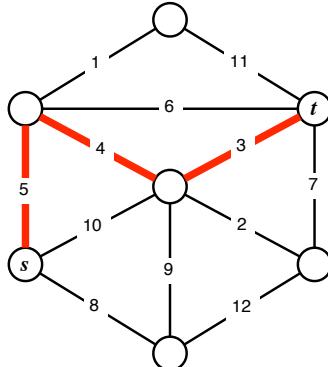
Hmmm.... You've probably known how to solve this problem since CS 125, so make it more interesting, let's add another constraint: The three spikes are arranged in a row, and we are also forbidden to move a disk directly from the left spike to the right spike or vice versa. In other words, we *must* move a disk either to or from the middle spike at *every* step.



The first four steps required to move the disks from the left spike to the right spike.

- (a) [4 pts] Describe an algorithm that moves the stack of  $n$  disks from the left needle to the right needle in as few steps as possible.
- (b) [6 pts] Exactly how many steps does your algorithm take to move all  $n$  disks? A correct  $\Theta$ -bound is worth 3 points. [Hint: Set up and solve a recurrence.]
- 
2. Consider a random walk on a path with vertices numbered  $1, 2, \dots, n$  from left to right. At each step, we flip a coin to decide which direction to walk, moving one step left or one step right with equal probability. The random walk ends when we fall off one end of the path, either by moving left from vertex 1 or by moving right from vertex  $n$ . In Midterm 2, you were asked to prove that if we start at vertex 1, the probability that the walk ends by falling off the *left* end of the path is exactly  $n/(n+1)$ .
- (a) [6 pts] Prove that if we start at vertex 1, the expected number of steps before the random walk ends is exactly  $n$ . [Hint: Set up and solve a recurrence. Use the result from Midterm 2.]
- (b) [4 pts] Suppose we start at vertex  $n/2$  instead. State a tight  $\Theta$ -bound on the expected length of the random walk in this case. **No proof is required.** [Hint: Set up and solve a recurrence. Use part (a), even if you can't prove it.]
- 
3. Prove that any connected acyclic graph with  $n$  vertices has exactly  $n - 1$  edges. Do not use the word "tree" or any well-known properties of trees; your proof should follow entirely from the definitions.

4. Consider a path between two vertices  $s$  and  $t$  in an undirected weighted graph  $G$ . The *bottleneck length* of this path is the maximum weight of any edge in the path. The *bottleneck distance* between  $s$  and  $t$  is the minimum bottleneck length of any path from  $s$  to  $t$ . (If there are no paths from  $s$  to  $t$ , the bottleneck distance between  $s$  and  $t$  is  $\infty$ .)



The bottleneck distance between  $s$  and  $t$  is 5.

Describe and analyze an algorithm to compute the bottleneck distance between every pair of vertices in an arbitrary undirected weighted graph. Assume that no two edges have the same weight.

5. The 5COLOR asks, given a graph  $G$ , whether the vertices of a graph  $G$  can be colored with five colors so that no edge has two endpoints with the same color. You already know from class that this problem is NP-complete.

Now consider the related problem 5COLOR $\pm 1$ : Given a graph  $G$ , can we color each vertex with an integer from the set  $\{0, 1, 2, 3, 4\}$ , so that for every edge, the colors of the two endpoints differ by exactly 1 modulo 5? (For example, a vertex with color 4 can only be adjacent to vertices colored 0 or 3.) We would like to show that 5COLOR $\pm 1$  is NP-complete as well.

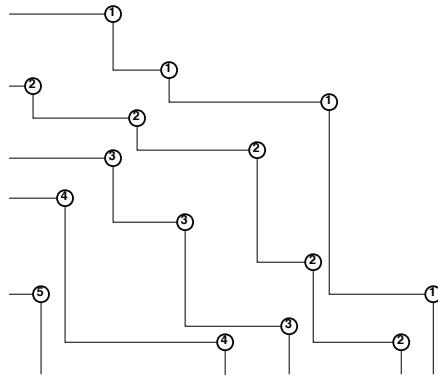
- (a) [2 pts] Show that 5COLOR $\pm 1$  is in NP.
- (b) [1 pt] To prove that 5COLOR $\pm 1$  is NP-hard (and therefore NP-complete), we must describe a polynomial time algorithm for *one* of the following problems. Which one?
  - Given an arbitrary graph  $G$ , compute a graph  $H$  such that 5COLOR( $G$ ) is true if and only if 5COLOR $\pm 1$ ( $H$ ) is true.
  - Given an arbitrary graph  $G$ , compute a graph  $H$  such that 5COLOR $\pm 1$ ( $G$ ) is true if and only if 5COLOR( $H$ ) is true.

- (c) [1 pt] Explain briefly why the following argument is not correct.

For any graph  $G$ , if 5COLOR $\pm 1$ ( $G$ ) is true, then 5COLOR( $G$ ) is true (using the same coloring). Therefore if we could solve 5COLOR $\pm 1$  quickly, we could also solve 5COLOR quickly. In other words, 5COLOR $\pm 1$  is at least as hard as 5COLOR. We know that 5COLOR is NP-hard, so 5COLOR $\pm 1$  must also be NP-hard!

- (d) [6 pts] Prove that 5COLOR $\pm 1$  is NP-hard. [Hint: Look at some small examples. Replace the edges of  $G$  with a simple gadget, so that the resulting graph  $H$  has the desired property from part (b).]

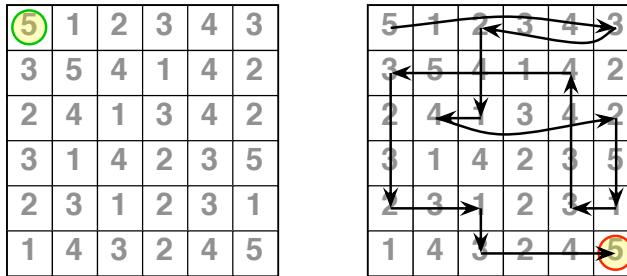
6. Let  $P$  be a set of points in the plane. Recall that a point  $p \in P$  is *Pareto-optimal* if no other points in  $P$  are both above and to the right of  $p$ . Intuitively, the sequence of Pareto-optimal points forms a *staircase* with all the other points in  $P$  below and to the left. The *staircase layers* of  $P$  are defined recursively as follows. The empty set has no staircase layers. Otherwise, the first staircase layer contains all the Pareto-optimal points in  $P$ , and the remaining layers are the staircase layers of  $P$  minus the first layer.



A set of points with 5 staircase layers

Describe and analyze an algorithm to compute the number of staircase layers of a point set  $P$  as quickly as possible. For example, given the points illustrated above, your algorithm would return the number 5.

7. Consider the following puzzle played on an  $n \times n$  square grid, where each square is labeled with a positive integer. A token is placed on one of the squares. At each turn, you may move the token left, right, up, or down; the distance you move the token must be equal to the number on the current square. For example, if the token is on a square labeled "3", you are allowed more the token three squares down, three square left, three squares up, or three squares right. You are never allowed to move the token off the board.



A sequence of legal moves from the top left corner to the bottom right corner.

- (a) [4 pts] Describe and analyze an algorithm to determine, given an  $n \times n$  array of labels and two squares  $s$  and  $t$ , whether there is a sequence of legal moves that takes the token from  $s$  to  $t$ .
- (b) [6 pts] Suppose you are only given the  $n \times n$  array of labels. Describe how to preprocess these values, so that afterwards, given any two squares  $s$  and  $t$ , you can determine in  $O(1)$  time whether there is a sequence of legal moves from  $s$  to  $t$ .

Answer four of these seven problems; the lowest three scores will be dropped.

1. Suppose we are given an array  $A[1..n]$  with the special property that  $A[1] \geq A[2]$  and  $A[n-1] \leq A[n]$ . We say that an element  $A[x]$  is a *local minimum* if it is less than or equal to both its neighbors, or more formally, if  $A[x-1] \geq A[x]$  and  $A[x] \leq A[x+1]$ . For example, there are five local minima in the following array:

9	7	7	2	1	3	7	5	4	7	3	3	4	8	6	9
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

We can obviously find a local minimum in  $O(n)$  time by scanning through the array. Describe and analyze an algorithm that finds a local minimum in  $O(\log n)$  time. [Hint: With the given boundary conditions, the array **must** have at least one local minimum. Why?]

2. Consider a random walk on a path with vertices numbered  $1, 2, \dots, n$  from left to right. At each step, we flip a coin to decide which direction to walk, moving one step left or one step right with equal probability. The random walk ends when we fall off one end of the path, either by moving left from vertex 1 or by moving right from vertex  $n$ . In Midterm 2, you were asked to prove that if we start at vertex 1, the probability that the walk ends by falling off the *left* end of the path is exactly  $n/(n+1)$ .
- (a) [6 pts] **Prove** that if we start at vertex 1, the expected number of steps before the random walk ends is exactly  $n$ . [Hint: Set up and solve a recurrence. Use the result from Midterm 2.]
  - (b) [4 pts] Suppose we start at vertex  $n/2$  instead. State and **prove** a tight  $\Theta$ -bound on the expected length of the random walk in this case. [Hint: Set up and solve a recurrence. Use part (a), even if you can't prove it.]
3. **Prove** that any connected acyclic graph with  $n \geq 2$  vertices has at least two vertices with degree 1. Do not use the words “tree” or “leaf”, or any well-known properties of trees; your proof should follow entirely from the definitions.
4. Consider the following sketch of a “reverse greedy” algorithm. The input is a connected undirected graph  $G$  with weighted edges, represented by an adjacency list.

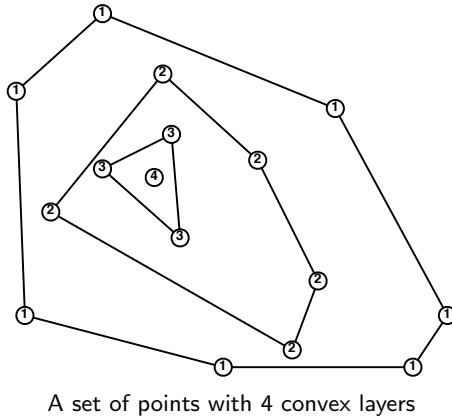
$\text{REVERSEGREEDYMST}(G)$ : <hr/> sort the edges $E$ of $G$ by weight for $i \leftarrow 1$ to $ E $ $e \leftarrow$ $i$ th heaviest edge in $E$ if $G \setminus e$ is connected $\quad$ remove $e$ from $G$
--

- (a) [4 pts] What is the worst-case running time of this algorithm? (Answering this question will require fleshing out a few details.)
- (b) [6 pts] **Prove** that the algorithm transforms  $G$  into its minimum spanning tree.

5. SUBSETSUM and PARTITION are two closely related NP-hard problems.

- SUBSETSUM: Given a set  $X$  of integers and an integer  $k$ , does  $X$  have a subset whose elements sum up to  $k$ ?
- PARTITION: Given a set  $X$  of integers, can  $X$  be partitioned into two subsets whose sums are equal?
  - (a) [2 pts] Prove that PARTITION and SUBSETSUM are both in NP.
  - (b) [1 pt] Suppose we knew that SUBSETSUM is NP-hard, and we wanted to prove that PARTITION is NP-hard. Which of the following arguments should we use?
    - Given a set  $X$  and an integer  $k$ , compute a set  $Y$  such that  $\text{PARTITION}(Y)$  is true if and only if  $\text{SUBSETSUM}(X, k)$  is true.
    - Given a set  $X$ , construct a set  $Y$  and an integer  $k$  such that  $\text{PARTITION}(X)$  is true if and only if  $\text{SUBSETSUM}(Y, k)$  is true.
  - (c) [3 pts] Describe and analyze a polynomial-time reduction from PARTITION to SUBSETSUM. (See part (b).)
  - (d) [4 pts] Describe and analyze a polynomial-time reduction from SUBSETSUM to PARTITION. (See part (b).)

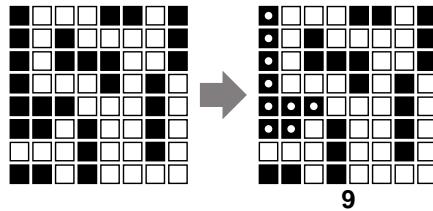
6. Let  $P$  be a set of points in the plane. The *convex layers* of  $P$  are defined recursively as follows. If  $P$  is empty, it has no convex layers. Otherwise, the first convex layer is the convex hull of  $P$ , and the remaining convex layers are the convex layers of  $P$  minus its convex hull.



Describe and analyze an algorithm to compute the number of convex layers of a point set  $P$  as quickly as possible. For example, given the points illustrated above, your algorithm would return the number 4.

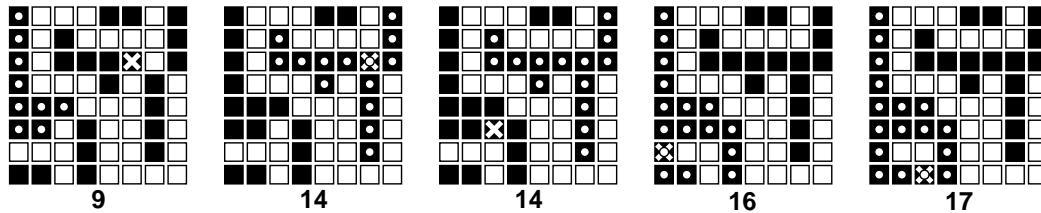
7. (a) [4 pts] Describe and analyze an algorithm to compute the size of the largest connected component of black pixels in an  $n \times n$  bitmap  $B[1..n, 1..n]$ .

For example, given the bitmap below as input, your algorithm should return the number 9, because the largest connected black component (marked with white dots on the right) contains nine pixels.



- (b) [4 pts] Design and analyze an algorithm  $\text{BLACKEN}(i, j)$  that colors the pixel  $B[i, j]$  black and returns the size of the largest black component in the bitmap. For full credit, the *amortized* running time of your algorithm (starting with an all-white bitmap) must be as small as possible.

For example, at each step in the sequence below, we blacken the pixel marked with an X. The largest black component is marked with white dots; the number underneath shows the correct output of the  $\text{BLACKEN}$  algorithm.



- (c) [2 pts] What is the *worst-case* running time of your  $\text{BLACKEN}$  algorithm?

**Write your answers in the separate answer booklet.**

1. A *data stream* is an extremely long sequence of items that you can only read only once, in order. A good example of a data stream is the sequence of packets that pass through a router. Data stream algorithms must process each item in the stream quickly, using very little memory; there is simply too much data to store, and it arrives too quickly for any complex computations. Every data stream algorithm looks roughly like this:

```
DOsomethingINTERESTING(stream  $S$ ):
repeat
     $x \leftarrow$  next item in  $S$ 
     $\langle\!\langle$  do something fast with  $x$   $\rangle\!\rangle$ 
until  $S$  ends
return  $\langle\!\langle$  something  $\rangle\!\rangle$ 
```

Describe and analyze an algorithm that chooses one element uniformly at random from a data stream, *without knowing the length of the stream in advance*. Your algorithm should spend  $O(1)$  time per stream element and use  $O(1)$  space (not counting the stream itself). Assume you have a subroutine  $\text{RANDOM}(n)$  that returns a random integer between 1 and  $n$ , each with equal probability, given any integer  $n$  as input.

2. Consider a random walk on a path with vertices numbered  $1, 2, \dots, n$  from left to right. We start at vertex 1. At each step, we flip a coin to decide which direction to walk, moving one step left or one step right with equal probability. The random walk ends when we fall off one end of the path, either by moving left from vertex 1 or by moving right from vertex  $n$ .

**Prove** that the probability that the walk ends by falling off the *left* end of the path is exactly  $n/(n+1)$ . [Hint: Set up a recurrence and verify that  $n/(n+1)$  satisfies it.]

3. Consider the following algorithms for maintaining a family of disjoint sets. The **UNION** algorithm uses a heuristic called *union by size*.

```
MAKESET( $x$ ):
parent( $x$ )  $\leftarrow x$ 
size( $x$ )  $\leftarrow 1$ 
```

```
UNION( $x, y$ ):
 $\bar{x} \leftarrow \text{FIND}(x)$ 
 $\bar{y} \leftarrow \text{FIND}(y)$ 
if size( $\bar{x}$ ) < size( $\bar{y}$ )
    parent( $\bar{x}$ )  $\leftarrow \bar{y}$ 
    size( $\bar{x}$ )  $\leftarrow$  size( $\bar{x}$ ) + size( $\bar{y}$ )
else
    parent( $\bar{y}$ )  $\leftarrow \bar{x}$ 
    size( $\bar{y}$ )  $\leftarrow$  size( $\bar{x}$ ) + size( $\bar{y}$ )
```

**Prove** that if we use union by size,  $\text{FIND}(x)$  runs in  $O(\log n)$  time *in the worst case*, where  $n$  is the size of the set containing element  $x$ .

4. Recall the SUBSETSUM problem: Given a set  $X$  of integers and an integer  $k$ , does  $X$  have a subset whose elements sum to  $k$ ?

- (a) [7 pts] Describe and analyze an algorithm that solves SUBSETSUM in time  $O(nk)$ .
- (b) [3 pts] SUBSETSUM is NP-hard. Does part (a) imply that P=NP? Justify your answer.

5. Suppose we want to maintain a set  $X$  of numbers under the following operations:

- $\text{INSERT}(x)$ : Add  $x$  to the set  $X$ .
- $\text{PRINTANDDELETEBETWEEN}(a, z)$ : Print every element  $x \in X$  such that  $a \leq x \leq z$ , in order from smallest to largest, and then delete those elements from  $X$ .

For example,  $\text{PRINTANDDELETEBETWEEN}(-\infty, \infty)$  prints all the elements of  $X$  in sorted order and then deletes everything.

- (a) [6 pts] Describe and analyze a data structure that supports these two operations, each in  $O(\log n)$  *amortized* time, where  $n$  is the maximum number of elements in  $X$ .
- (b) [2 pts] What is the running time of your  $\text{INSERT}$  algorithm in the worst case?
- (c) [2 pts] What is the running time of your  $\text{PRINTANDDELETEBETWEEN}$  algorithm in the worst case?

# CS 473G: Combinatorial Algorithms, Fall 2005

## Homework 0

Due Thursday, September 1, 2005, at the beginning of class (12:30pm CDT)

Name:	
Net ID:	Alias:

I understand the Homework Instructions and FAQ.

- 
- Neatly print your full name, your NetID, and an alias of your choice in the boxes above. Grades will be listed on the course web site by alias. Please write the same alias on every homework and exam! For privacy reasons, your alias should not resemble your name or NetID. By providing an alias, you agree to let us list your grades; if you do not provide an alias, your grades will not be listed. **Never give us your Social Security number!**
  - Read the “Homework Instructions and FAQ” on the course web page, and then check the box above. This page describes what we expect in your homework solutions—start each numbered problem on a new sheet of paper, write your name and NetID on every page, don’t turn in source code, analyze and prove everything, use good English and good logic, and so on—as well as policies on grading standards, regrading, and plagiarism. **See especially the course policies regarding the magic phrases “I don’t know” and “and so on”.** If you have *any* questions, post them to the course newsgroup or ask during lecture.
  - Don’t forget to submit this cover sheet with the rest of your homework solutions.
  - This homework tests your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, discrete probability, and most importantly, induction—to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Chapters 1–10 of CLRS should be sufficient review, but you may also want consult your discrete mathematics and data structures textbooks.
  - Every homework will have five required problems. Most homeworks will also include one extra-credit problem and several practice (no-credit) problems. Each numbered problem is worth 10 points.
- 

#	1	2	3	4	5	6*	Total
Score							
Grader							

1. Solve the following recurrences. State tight asymptotic bounds for each function in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ . You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway, just for practice. Assume reasonable but nontrivial base cases. **If your solution requires specific base cases, state them!**

(a)  $A(n) = 2A(n/4) + \sqrt{n}$

(b)  $B(n) = \max_{n/3 < k < 2n/3} (B(k) + B(n - k) + n)$

(c)  $C(n) = 3C(n/3) + n/\lg n$

(d)  $D(n) = 3D(n - 1) - 3D(n - 2) + D(n - 3)$

(e)  $E(n) = \frac{E(n - 1)}{3E(n - 2)}$  [Hint: This is easy!]

(f)  $F(n) = F(n - 2) + 2/n$

(g)  $G(n) = 2G(\lceil(n + 3)/4\rceil - 5n/\sqrt{\lg n} + 6\lg\lg n) + 7\sqrt[8]{n - 9} - \lg^{10} n/\lg\lg n + 11^{\lg^* n} - 12$

\*(h)  $H(n) = 4H(n/2) - 4H(n/4) + 1$  [Hint: Careful!]

(i)  $I(n) = I(n/2) + I(n/4) + I(n/8) + I(n/12) + I(n/24) + n$

★(j)  $J(n) = 2\sqrt{n} \cdot J(\sqrt{n}) + n$

[Hint: First solve the secondary recurrence  $j(n) = 1 + j(\sqrt{n})$ .]

2. Penn and Teller agree to play the following game. Penn shuffles a standard deck<sup>1</sup> of playing cards so that every permutation is equally likely. Then Teller draws cards from the deck, one at a time without replacement, until he draws the three of clubs ( $3\clubsuit$ ), at which point the remaining undrawn cards instantly burst into flames and the game is over.

The first time Teller draws a card from the deck, he gives it to Penn. From then on, until the game ends, whenever Teller draws a card whose value is smaller than the previous card he gave to Penn, he gives the new card to Penn. To make the rules unambiguous, they agree on the numerical values  $A = 1$ ,  $J = 11$ ,  $Q = 12$ , and  $K = 13$ .

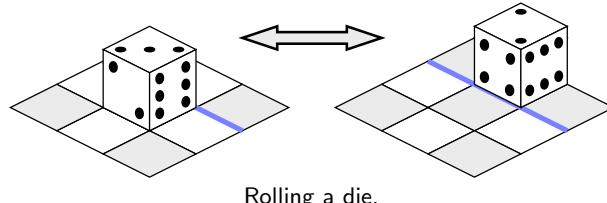
- (a) What is the expected number of cards that Teller draws?
- (b) What is the expected *maximum* value among the cards Teller gives to Penn?
- (c) What is the expected *minimum* value among the cards Teller gives to Penn?
- (d) What is the expected number of cards that Teller gives to Penn?

Full credit will be given only for *exact* answers (with correct proofs, of course).

---

<sup>1</sup>In a standard deck of 52 cards, each card has a *suit* in the set  $\{\spadesuit, \heartsuit, \clubsuit, \diamondsuit\}$  and a *value* in the set  $\{A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K\}$ , and every possible suit-value pair appears in the deck exactly once. Penn and Teller normally use exploding razor-sharp ninja throwing cards for this trick.

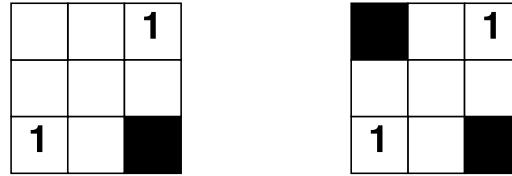
3. A *rolling die maze* is a puzzle involving a standard six-sided die<sup>2</sup> and a grid of squares. You should imagine the grid lying on top of a table; the die always rests on and exactly covers one square. In a single step, you can *roll* the die 90 degrees around one of its bottom edges, moving it to an adjacent square one step north, south, east, or west.



Rolling a die.

Some squares in the grid may be *blocked*; the die can never rest on a blocked square. Other squares may be *labeled* with a number; whenever the die rests on a labeled square, the number of pips on the *top* face of the die must equal the label. Squares that are neither labeled nor marked are *free*. You may not roll the die off the edges of the grid. A rolling die maze is *solvable* if it is possible to place a die on the lower left square and roll it to the upper right square under these constraints.

For example, here are two rolling die mazes. Black squares are blocked. The maze on the left can be solved by placing the die on the lower left square with 1 pip on the top face, and then rolling it north, then north, then east, then east. The maze on the right is not solvable.



Two rolling die mazes. Only the maze on the left is solvable.

- (a) Suppose the input is a two-dimensional array  $L[1..n][1..n]$ , where each entry  $L[i][j]$  stores the label of the square in the  $i$ th row and  $j$ th column, where 0 means the square is free and  $-1$  means the square is blocked. Describe and analyze a polynomial-time algorithm to determine whether the given rolling die maze is solvable.
- \*(b) Now suppose the maze is specified *implicitly* by a list of labeled and blocked squares. Specifically, suppose the input consists of an integer  $M$ , specifying the height and width of the maze, and an array  $S[1..n]$ , where each entry  $S[i]$  is a triple  $(x, y, L)$  indicating that square  $(x, y)$  has label  $L$ . As in the explicit encoding, label  $-1$  indicates that the square is blocked; free squares are not listed in  $S$  at all. Describe and analyze an efficient algorithm to determine whether the given rolling die maze is solvable. For full credit, the running time of your algorithm should be polynomial in the input size  $n$ .

[Hint: You have some freedom in how to place the initial die. There are rolling die mazes that can only be solved if the initial position is chosen correctly.]

---

<sup>2</sup>A standard die is a cube, where each side is labeled with a different number of dots, called *pips*, between 1 and 6. The labeling is chosen so that any pair of opposite sides has a total of 7 pips.

4. Whenever groups of pigeons gather, they instinctively establish a *pecking order*. For any pair of pigeons, one pigeon always pecks the other, driving it away from food or potential mates. The same pair of pigeons will always choose the same pecking order, even after years of separation, no matter what other pigeons are around. (Like most things, revenge is a foreign concept to pigeons.) Surprisingly, the overall pecking order in a set of pigeons can contain cycles—for example, pigeon A pecks pigeon B, which pecks pigeon C, which pecks pigeon A. Prove that any set of pigeons can be arranged in a row so that every pigeon pecks the pigeon immediately to its right.

5. Scientists have recently discovered a planet, tentatively named “Ygdrasil”, which is inhabited by a bizarre species called “vodes”. All vodes trace their ancestry back to a particular vode named Rudy. Rudy is still quite alive, as is every one of his many descendants. Vodes reproduce asexually, like bees; each vode has exactly one parent (except Rudy, who has no parent). There are three different colors of vodes—cyan, magenta, and yellow. The color of each vode is correlated exactly with the number and colors of its children, as follows:

- Each cyan vode has two children, exactly one of which is yellow.
- Each yellow vode has exactly one child, which is not yellow.
- Magenta vodes have no children.

In each of the following problems, let  $C$ ,  $M$ , and  $Y$  respectively denote the number of cyan, magenta, and yellow vodes on Ygdrasil.

- (a) Prove that  $M = C + 1$ .
- (b) Prove that either  $Y = C$  or  $Y = M$ .
- (c) Prove that  $Y = M$  if and only if Rudy is yellow.

[Hint: Be very careful to **prove** that you have considered **all** possibilities.]

**\*6. [Extra credit]<sup>3</sup>**

*Lazy binary* is a variant of standard binary notation for representing natural numbers where we allow each “bit” to take on one of three values: 0, 1, or 2. Lazy binary notation is defined inductively as follows.

- The lazy binary representation of zero is 0.
- Given the lazy binary representation of any non-negative integer  $n$ , we can construct the lazy binary representation of  $n + 1$  as follows:
  - (a) increment the rightmost digit;
  - (b) if any digit is equal to 2, replace the rightmost 2 with 0 and increment the digit immediately to its left.

Here are the first several natural numbers in lazy binary notation:

0, 1, 10, 11, 20, 101, 110, 111, 120, 201, 210, 1011, 1020, 1101, 1110, 1111, 1120, 1201, 1210, 2011, 2020, 2101, 2110, 10111, 10120, 10201, 10210, 11011, 11020, 11101, 11110, 11111, 11120, 11201, 11210, 12011, 12020, 12101, 12110, 20111, 20120, 20201, 20210, 21011, 21020, 21101, 21110, 101111, 101120, 101201, 101210, 102011, 102020, 102101, 102110, ...

- (a) Prove that in any lazy binary number, between any two 2s there is at least one 0, and between two 0s there is at least one 2.
- (b) Prove that for any natural number  $N$ , the sum of the digits of the lazy binary representation of  $N$  is exactly  $\lfloor \lg(N + 1) \rfloor$ .

---

<sup>3</sup>The “I don’t know” rule does not apply to extra credit problems. There is no such thing as “partial extra credit”.

## Practice Problems

The remaining problems are for practice only. Please do not submit solutions. On the other hand, feel free to discuss these problems in office hours or on the course newsgroup.

- Sort the functions in each box from asymptotically smallest to asymptotically largest, indicating ties if there are any. You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway, just for practice.

1	$\lg n$	$\lg^2 n$	$\sqrt{n}$	$n$	$n^2$	$2^{\sqrt{n}}$	$\sqrt{2}^n$
$2^{\sqrt{\lg n}}$	$2^{\lg \sqrt{n}}$	$\sqrt{2^{\lg n}}$	$\sqrt{2}^{\lg n}$	$\lg 2^{\sqrt{n}}$	$\lg \sqrt{2}^n$	$\lg \sqrt{2^n}$	$\sqrt{\lg 2^n}$
$\lg n^{\sqrt{2}}$	$\lg \sqrt{n^2}$	$\lg \sqrt{n^2}$	$\sqrt{\lg n^2}$	$\lg^2 \sqrt{n}$	$\lg^{\sqrt{2}} n$	$\sqrt{\lg^2 n}$	$\sqrt{\lg n^2}$

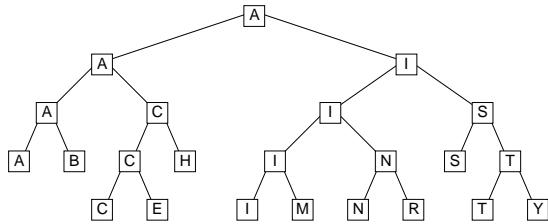
To simplify your answers, write  $f(n) \ll g(n)$  to mean  $f(n) = o(g(n))$ , and write  $f(n) \equiv g(n)$  to mean  $f(n) = \Theta(g(n))$ . For example, the functions  $n^2, n, \binom{n}{2}, n^3$  could be sorted either as  $n \ll n^2 \equiv \binom{n}{2} \ll n^3$  or as  $n \ll \binom{n}{2} \equiv n^2 \ll n^3$ .

- Recall the standard recursive definition of the Fibonacci numbers:  $F_0 = 0$ ,  $F_1 = 1$ , and  $F_n = F_{n-1} + F_{n-2}$  for all  $n \geq 2$ . Prove the following identities for all positive integers  $n$  and  $m$ .
  - $F_n$  is even if and only if  $n$  is divisible by 3.
  - $\sum_{i=0}^n F_i = F_{n+2} - 1$
  - $F_n^2 - F_{n+1}F_{n-1} = (-1)^{n+1}$
  - If  $n$  is an integer multiple of  $m$ , then  $F_n$  is an integer multiple of  $F_m$ .
- Penn and Teller have a special deck of fifty-two cards, with no face cards and nothing but clubs—the ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ..., 52 of clubs. (They’re big cards.) Penn shuffles the deck until each each of the  $52!$  possible orderings of the cards is equally likely. He then takes cards one at a time from the top of the deck and gives them to Teller, stopping as soon as he gives Teller the three of clubs.
  - On average, how many cards does Penn give Teller?
  - On average, what is the smallest-numbered card that Penn gives Teller?
  - On average, what is the largest-numbered card that Penn gives Teller?

Prove that your answers are correct. (If you have to appeal to “intuition” or “common sense”, your answers are probably wrong.) [Hint: Solve for an  $n$ -card deck, and then set  $n$  to 52.]

4. Algorithms and data structures were developed millions of years ago by the Martians, but not quite in the same way as the recent development here on Earth. Intelligent life evolved independently on Mars' two moons, Phobos and Deimos.<sup>4</sup> When the two races finally met on the surface of Mars, after thousands of years of separate philosophical, cultural, religious, and scientific development, their disagreements over the proper structure of binary search trees led to a bloody (or more accurately, ichorous) war, ultimately leading to the destruction of all Martian life.

A *Phobian* binary search tree is a full binary tree that stores a set  $X$  of search keys. The root of the tree stores the *smallest* element in  $X$ . If  $X$  has more than one element, then the left subtree stores all the elements less than some pivot value  $p$ , and the right subtree stores everything else. Both subtrees are *nonempty* Phobian binary search trees. The actual pivot value  $p$  is *never* stored in the tree.



A Phobian binary search tree for the set {M, A, R, T, I, N, B, Y, S, E, C, H}.

- (a) Describe and analyze an algorithm  $\text{FIND}(x, T)$  that returns TRUE if  $x$  is stored in the Phobian binary search tree  $T$ , and FALSE otherwise.
- (b) A *Deimoid* binary search tree is almost exactly the same as its Phobian counterpart, except that the *largest* element is stored at the root, and both subtrees are Deimoid binary search trees. Describe and analyze an algorithm to transform an  $n$ -node Phobian binary search tree into a Deimoid binary search tree in  $O(n)$  time, using as little additional space as possible.
5. Tatami are rectangular mats used to tile floors in traditional Japanese houses. Exact dimensions of tatami mats vary from one region of Japan to the next, but they are always twice as long in one dimension than in the other. (In Tokyo, the standard size is 180cm×90cm.)
- (a) How many different ways are there to tile a  $2 \times n$  rectangular room with  $1 \times 2$  tatami mats? Set up a recurrence and derive an *exact* closed-form solution. [Hint: The answer involves a familiar recursive sequence.]
- (b) According to tradition, tatami mats are always arranged so that four corners never meet. How many different *traditional* ways are there to tile a  $3 \times n$  rectangular room with  $1 \times 2$  tatami mats? Set up a recurrence and derive an *exact* closed-form solution.
- \*(c) How many different *traditional* ways are there to tile an  $n \times n$  square with  $1 \times 2$  tatami mats? Prove your answer is correct.

<sup>4</sup>Greek for “fear” and “panic”, respectively. Doesn’t that make you feel better?

# CS 473G: Combinatorial Algorithms, Fall 2005

## Homework 1

Due Tuesday, September 13, 2005, by midnight (11:59:59pm CDT)

Name:	
Net ID:	Alias:

Name:	
Net ID:	Alias:

Name:	
Net ID:	Alias:

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade.

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Staple this sheet to the top of your answer to problem 1.

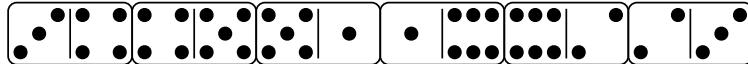
There are two steps required to prove NP-completeness: (1) Prove that the problem is in NP, by describing a polynomial-time verification algorithm. (2) Prove that the problem is NP-hard, by describing a polynomial-time reduction from some other NP-hard problem. Showing that the reduction is correct requires proving an if-and-only-if statement; don't forget to prove both the "if" part and the "only if" part.

---

### Required Problems

1. Some NP-Complete problems
  - (a) Show that the problem of deciding whether one graph is a subgraph of another is NP-complete.
  - (b) Given a boolean circuit that embeds in the plane so that no 2 wires cross, PLANARCIRCUITSAT is the problem of determining if there is a boolean assignment to the inputs that makes the circuit output true. Prove that PLANARCIRCUITSAT is NP-Complete.
  - (c) Given a set  $S$  with  $3n$  numbers, 3PARTITION is the problem of determining if  $S$  can be partitioned into  $n$  disjoint subsets, each with 3 elements, so that every subset sums to the same value. Given a set  $S$  and a collection of three element subsets of  $S$ , X3M (or *exact 3-dimensional matching*) is the problem of determining whether there is a subcollection of  $n$  disjoint triples that exactly cover  $S$ .  
Describe a polynomial-time reduction from 3PARTITION to X3M.

- (d) A *domino* is a  $1 \times 2$  rectangle divided into two squares, each of which is labeled with an integer.<sup>1</sup> In a *legal arrangement* of dominoes, the dominoes are lined up end-to-end so that the numbers on adjacent ends match.



A legal arrangement of dominos, where every integer between 1 and 6 appears twice

Prove that the following problem is NP-complete: Given an arbitrary collection  $D$  of dominoes, is there a legal arrangement of a subset of  $D$  in which every integer between 1 and  $n$  appears exactly twice?

2. Prove that the following problems are all polynomial-time equivalent, that is, if *any* of these problems can be solved in polynomial time, then *all* of them can.

- CLIQUE: Given a graph  $G$  and an integer  $k$ , does there exist a clique of size  $k$  in  $G$ ?
- FINDCLIQUE: Given a graph  $G$  and an integer  $k$ , find a clique of size  $k$  in  $G$  if one exists.
- MAXCLIQUE: Given a graph  $G$ , find the size of the largest clique in the graph.
- FINDMAXCLIQUE: Given a graph  $G$ , find a clique of maximum size in  $G$ .

3. Consider the following problem: Given a set of  $n$  points in the plane, find a set of line segments connecting the points which form a closed loop and do not intersect each other.

Describe a linear time reduction from the problem of sorting  $n$  numbers to the problem described above.

4. In graph coloring, the vertices of a graph are assigned colors so that no adjacent vertices receive the same color. We saw in class that determining if a graph is 3-colorable is NP-Complete.

Suppose you are handed a magic black box that, given a graph as input, tells you *in constant time* whether or not the graph is 3-colorable. Using this black box, give a *polynomial-time* algorithm to 3-color a graph.

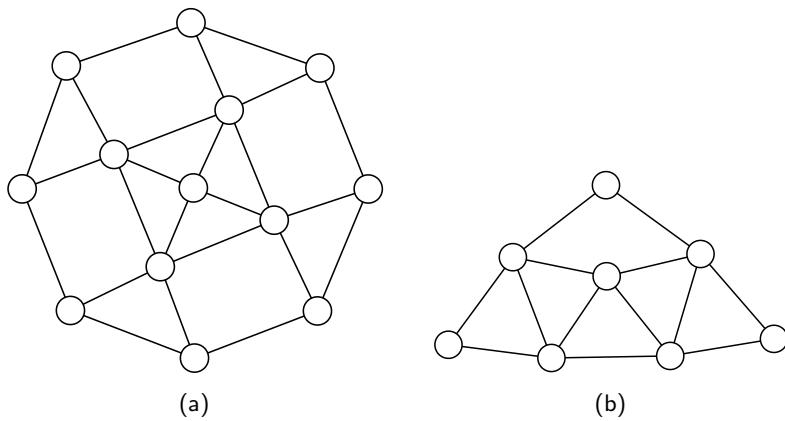
5. Suppose that Cook had proved that graph coloring was NP-complete first, instead of CIRCUITSAT. Using only the fact that graph coloring is NP-complete, show that CIRCUITSAT is NP-complete.

---

<sup>1</sup>These integers are usually represented by pips, exactly like dice. On a standard domino, the number of pips on each side is between 0 and 6; we will allow arbitrary integer labels. A standard set of dominoes has one domino for each possible unordered pair of labels; we do not require that every possible label pair is in our set.

# Practice Problems

- Given an initial configuration consisting of an undirected graph  $G = (V, E)$  and a function  $p : V \rightarrow \mathbb{N}$  indicating an initial number of pebbles on each vertex, PEBBLE-DESTRUCTION asks if there is a sequence of pebbling moves starting with the initial configuration and ending with a single pebble on only one vertex of  $V$ . Here, a pebbling move consists of removing two pebbles from a vertex  $v$  and adding one pebble to a neighbor of  $v$ . Prove that PEBBLE-DESTRUCTION is NP-complete.
  - Consider finding the median of 5 numbers by using only comparisons. What is the *exact* worst case number of comparisons needed to find the median? To prove your answer is correct, you must exhibit both an algorithm that uses that many comparisons and a proof that there is no faster algorithm. Do the same for 6 numbers.
  - PARTITION is the problem of deciding, given a set  $S$  of numbers, whether it can be partitioned into two subsets whose sums are equal. (A *partition* of  $S$  is a collection of disjoint subsets whose union is  $S$ .) SUBSETSUM is the problem of deciding, given a set  $S$  of numbers and a target sum  $t$ , whether any subset of numbers in  $S$  sum to  $t$ .
    - Describe a polynomial-time reduction from SUBSETSUM to PARTITION.
    - Describe a polynomial-time reduction from PARTITION to SUBSETSUM.
  - Recall from class that the problem of deciding whether a graph can be colored with three colors, so that no edge joins nodes of the same color, is NP-complete.
    - Using the gadget in Figure 1(a), prove that deciding whether a *planar* graph can be 3-colored is NP-complete. [Hint: Show that the gadget can be 3-colored, and then replace any crossings in a planar embedding with the gadget appropriately.]



**Figure 1.** (a) Gadget for planar 3-colorability. (b) Gadget for degree-4 planar 3-colorability.

- (b) Using the previous result and the gadget in figure 1(b), prove that deciding whether a planar graph with maximum degree 4 can be 3-colored is NP-complete. [Hint: Show that you can replace any vertex with degree greater than 4 with a collection of gadgets connected in such a way that no degree is greater than four.]

5. (a) Prove that if  $G$  is an undirected bipartite graph with an odd number of vertices, then  $G$  is nonhamiltonian. Describe a polynomial-time algorithm to find a hamiltonian cycle in an undirected bipartite graph, or establish that no such cycle exists.  
(b) Describe a polynomial time algorithm to find a hamiltonian *path* in a directed acyclic graph, or establish that no such path exists.  
(c) Why don't these results imply that P=NP?
6. Consider the following pairs of problems:
  - (a) MIN SPANNING TREE and MAX SPANNING TREE
  - (b) SHORTEST PATH and LONGEST PATH
  - (c) TRAVELING SALESMAN PROBLEM and VACATION TOUR PROBLEM (the longest tour is sought).
  - (d) MIN CUT and MAX CUT (between  $s$  and  $t$ )
  - (e) EDGE COVER and VERTEX COVER
  - (f) TRANSITIVE REDUCTION and MIN EQUIVALENT DIGRAPH

(all of these seem dual or opposites, except the last, which are just two versions of minimal representation of a graph).  
Which of these pairs are polytime equivalent and which are not? Why?
7. Prove that PRIMALITY (Given  $n$ , is  $n$  prime?) is in  $\text{NP} \cap \text{co-NP}$ . [Hint: co-NP is easy—What's a certificate for showing that a number is composite? For NP, consider a certificate involving primitive roots and recursively their primitive roots. Show that this tree of primitive roots can be verified and used to show that  $n$  is prime in polynomial time.]
8. How much wood would a woodchuck chuck if a woodchuck could chuck wood?

# CS 473G: Combinatorial Algorithms, Fall 2005

## Homework 2

Due Thursday, September 22, 2005, by midnight (11:59:59pm CDT)

Name:
Net ID:      Alias:

Name:
Net ID:      Alias:

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade.

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Staple this sheet to the top of your homework.

---

### Required Problems

1. (a) Suppose Lois has an algorithm to compute the shortest common supersequence of two arrays of integers in  $O(n)$  time. Describe an  $O(n \log n)$ -time algorithm to compute the longest common subsequence of two arrays of integers, using Lois's algorithm as a subroutine.  
(b) Describe an  $O(n \log n)$ -time algorithm to compute the longest increasing subsequence of an array of integers, using Lois's algorithm as a subroutine.  
(c) Now suppose Lisa has an algorithm that can compute the longest increasing subsequence of an array of integers in  $O(n)$  time. Describe an  $O(n \log n)$ -time algorithm to compute the longest common subsequence of two arrays  $A[1..n]$  and  $B[1..n]$  of integers, **where  $A[i] \neq A[j]$  for all  $i \neq j$** , using Lisa's algorithm as a subroutine.<sup>1</sup>

---

<sup>1</sup>For extra credit, remove the assumption that the elements of  $A$  are distinct. This is probably impossible.

2. In a previous incarnation, you worked as a cashier in the lost 19th-century Antarctic colony of Nadira, spending the better part of your day giving change to your customers. Because paper is a very rare and valuable resource on Antarctica, cashiers were required by law to use the fewest bills possible whenever they gave change. Thanks to the numerological predilections of one of its founders, the currency of Nadira, called Dream Dollars, was available in the following denominations: \$1, \$4, \$7, \$13, \$28, \$52, \$91, \$365.<sup>2</sup>

- (a) The greedy change algorithm repeatedly takes the largest bill that does not exceed the target amount. For example, to make \$122 using the greedy algorithm, we first take a \$91 bill, then a \$28 bill, and finally three \$1 bills. Give an example where this greedy algorithm uses more Dream Dollar bills than the minimum possible.
- (b) Describe and analyze an efficient algorithm that computes, given an integer  $n$ , the minimum number of bills needed to make  $n$  Dream Dollars.

3. Scientists have branched out from the bizarre planet of Yggdrasil to study the vodes which have settled on Ygdrasil's moon, Xryltcon. All vodes on Xryltcon are descended from the first vode to arrive there, named George. Each vode has a color, either cyan, magenta, or yellow, but breeding patterns are *not* the same as on Yggdrasil; every vode, regardless of color, has either two children (with arbitrary colors) or no children.

George and all his descendants are alive and well, and they are quite excited to meet the scientists who wish to study them. Unsurprisingly, these vodes have had some strange mutations in their isolation on Xryltcon. Each vode has a *weirdness* rating; weirder vodes are more interesting to the visiting scientists. (Some vodes even have negative weirdness ratings; they make other vodes more boring just by standing next to them.)

Also, Xryltconian society is strictly governed by a number of sacred cultural traditions.

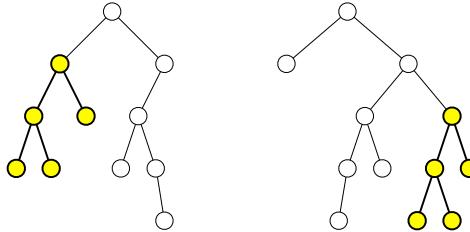
- No cyan vode may be in the same room as its non-cyan children (if it has any).
- No magenta vode may be in the same room as its parent (if it has one).
- Each yellow vode must be attended at all times by its grandchildren (if it has any).
- George must be present at any gathering of more than fifty vodes.

The scientists have exactly one chance to study a group of vodes in a single room. You are given the family tree of all the vodes on Xryltcon, along with the weirdness value of each vode. Design and analyze an efficient algorithm to decide which vodes the scientists should invite to maximize the sum of the weirdness values of the vodes in the room. Be careful to respect all of the vodes' cultural taboos.

---

<sup>2</sup>For more details on the history and culture of Nadira, including images of the various denominations of Dream Dollars, see <http://www.dream-dollars.com>. Really.

4. A *subtree* of a (rooted, ordered) binary tree  $T$  consists of a node and all its descendants. Design and analyze an efficient algorithm to compute the *largest common subtree* of two given binary trees  $T_1$  and  $T_2$ , that is, the largest subtree of  $T_1$  that is isomorphic to a subtree in  $T_2$ . The *contents* of the nodes are irrelevant; we are only interested in matching the underlying combinatorial structure.



Two binary trees, with their largest common subtree emphasized

5. Let  $D[1..n]$  be an array of digits, each an integer between 0 and 9. An *digital subsequence* of  $D$  is an sequence of positive integers composed in the usual way from disjoint substrings of  $D$ . For example, 3, 4, 5, 6, 23, 38, 62, 64, 83, 279 is an increasing digital subsequence of the first several digits of  $\pi$ :

$$\boxed{3}, 1, \boxed{4}, 1, \boxed{5}, 9, \boxed{6}, \boxed{2, 3}, 4, \boxed{3, 8}, 4, \boxed{6, 2}, \boxed{6, 4}, 3, 3, \boxed{8, 3}, \boxed{2, 7, 9}$$

The *length* of a digital subsequence is the number of integers it contains, *not* the number of digits; the previous example has length 10.

Describe and analyze an efficient algorithm to compute the longest increasing digital subsequence of  $D$ . [Hint: Be careful about your computational assumptions. How long does it take to compare two  $k$ -digit numbers?]

- \*6. [Extra credit] The *chromatic number* of a graph  $G$  is the minimum number of colors needed to color the nodes of  $G$  so that no pair of adjacent nodes have the same color.

- (a) Describe and analyze a *recursive* algorithm to compute the chromatic number of an  $n$ -vertex graph in  $O(4^n \text{ poly}(n))$  time. [Hint: Catalan numbers play a role here.]
- (b) Describe and analyze an algorithm to compute the chromatic number of an  $n$ -vertex graph in  $O(3^n \text{ poly}(n))$  time. [Hint: Use dynamic programming. What is  $(1+x)^n$ ?]
- (c) Describe and analyze an algorithm to compute the chromatic number of an  $n$ -vertex graph in  $O((1+3^{1/3})^n \text{ poly}(n))$  time. [Hint: Use (but don't regurgitate) the algorithm in the lecture notes that counts all the maximal independent sets in an  $n$ -vertex graph in  $O(3^{n/3})$  time.]

## Practice Problems

- \*1. Describe an algorithm to solve 3SAT in time  $O(\phi^n \text{ poly}(n))$ , where  $\phi = (1 + \sqrt{5})/2$ . [Hint: Prove that in each recursive call, either you have just eliminated a pure literal, or the formula has a clause with at most two literals.]
2. Describe and analyze an algorithm to compute the longest increasing subsequence in an  $n$ -element array of integers in  $O(n \log n)$  time. [Hint: Modify the  $O(n^2)$ -time algorithm presented in class.]
3. The *edit distance* between two strings  $A$  and  $B$ , denoted  $\text{Edit}(A, B)$ , is the minimum number of insertions, deletions, or substitutions required to transform  $A$  into  $B$  (or vice versa). Edit distance is sometimes also called the *Levenshtein distance*.

Let  $\mathbf{A} = \{A_1, A_2, \dots, A_k\}$  be a set of strings. The *edit radius* of  $\mathbf{A}$  is the minimum over all strings  $X$  of the maximum edit distance from  $X$  to any string  $A_i$ :

$$\text{>EditRadius}(\mathbf{A}) = \min_{\text{strings } X} \max_{1 \leq i \leq k} \text{Edit}(X, A_i)$$

A string  $X$  that achieves this minimum is called an *edit center* of  $\mathbf{A}$ . A set of strings may have several edit centers, but the edit radius is unique.

Describe an efficient algorithm to compute the edit radius of three given strings.

4. Given 5 sequences of numbers, each of length  $n$ , design and analyze an efficient algorithm to compute the longest common subsequence among all 5 sequences.
5. Suppose we want to display a paragraph of text on a computer screen. The text consists of  $n$  words, where the  $i$ th word is  $W[i]$  pixels wide. We want to break the paragraph into several lines, each exactly  $L$  pixels long. Depending on which words we put on each line, we will need to insert different amounts of white space between the words. The paragraph should be fully justified, meaning that the first word on each line starts at its leftmost pixel, and *except for the last line*, the last character on each line ends at its rightmost pixel. (Look at the paragraph you are reading right now!) There must be at least one pixel of white space between any two words on the same line. Thus, if a line contains words  $i$  through  $j$ , then the amount of *extra* white space on that line is  $L - j + i - \sum_{k=i}^j W[k]$ .

Define the *slop* of a paragraph layout as the sum, over all lines *except the last*, of the *cube* of the extra white space in each line. Describe an efficient algorithm to layout the paragraph with minimum slop, given the list  $W[1..n]$  of word widths as input. You can assume that  $W[i] < L/2$  for each  $i$ , so that each line contains at least two words.

6. A *partition* of a positive integer  $n$  is a multiset of positive integers that sum to  $n$ . Traditionally, the elements of a partition are written in non-decreasing order, separated by + signs. For example, the integer 7 has exactly twelve partitions:

$$\begin{array}{lll} 1 + 1 + 1 + 1 + 1 + 1 + 1 & 3 + 1 + 1 + 1 + 1 & 4 + 1 + 1 + 1 \\ 2 + 1 + 1 + 1 + 1 + 1 & 3 + 2 + 1 + 1 & 4 + 2 + 1 \\ 2 + 2 + 1 + 1 + 1 & 3 + 2 + 2 & 4 + 3 \\ 2 + 2 + 2 + 1 & 3 + 3 + 1 & 7 \end{array}$$

The *roughness* of a partition  $a_1 + a_2 + \dots + a_k$  is defined as follows:

$$\rho(a_1 + a_2 + \dots + a_k) = \sum_{i=1}^{k-1} |a_{i+1} - a_i - 1| + a_k - 1$$

A *smoothest* partition of  $n$  is the partition of  $n$  with minimum roughness. Intuitively, the smoothest partition is the one closest to a descending arithmetic series  $k + \dots + 3 + 2 + 1$ , which is the only partition that has roughness 0. For example, the smoothest partitions of 7 are  $4 + 2 + 1$  and  $3 + 2 + 1 + 1$ :

$$\begin{array}{lll} \rho(1 + 1 + 1 + 1 + 1 + 1 + 1) = 6 & \rho(3 + 1 + 1 + 1 + 1) = 4 & \rho(4 + 1 + 1 + 1) = 4 \\ \rho(2 + 1 + 1 + 1 + 1 + 1) = 4 & \rho(3 + 2 + 1 + 1) = 1 & \rho(4 + 2 + 1) = 1 \\ \rho(2 + 2 + 1 + 1 + 1) = 3 & \rho(3 + 2 + 2) = 2 & \rho(4 + 3) = 2 \\ \rho(2 + 2 + 2 + 1) = 2 & \rho(3 + 3 + 1) = 2 & \rho(7) = 7 \end{array}$$

Describe and analyze an algorithm to compute, given a positive integer  $n$ , a smoothest partition of  $n$ .

# CS 473G: Combinatorial Algorithms, Fall 2005

## Homework 3

Due Tuesday, October 18, 2005, at midnight

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade.

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Staple this sheet to the top of your homework.

---

1. Consider the following greedy approximation algorithm to find a vertex cover in a graph:

```
GREEDYVERTEXCOVER( $G$ ):  
     $C \leftarrow \emptyset$   
    while  $G$  has at least one edge  
         $v \leftarrow$  vertex in  $G$  with maximum degree  
         $G \leftarrow G \setminus v$   
         $C \leftarrow C \cup v$   
    return  $C$ 
```

In class we proved that the approximation ratio of this algorithm is  $O(\log n)$ ; your task is to prove a matching lower bound. Specifically, prove that for any integer  $n$ , there is a graph  $G$  with  $n$  vertices such that  $\text{GREEDYVERTEXCOVER}(G)$  returns a vertex cover that is  $\Omega(\log n)$  times larger than optimal.

2. Prove that for *any* constant  $k$  and *any* graph coloring algorithm  $A$ , there is a graph  $G$  such that  $A(G) > OPT(G) + k$ , where  $A(G)$  is the number of colors generated by algorithm  $A$  for graph  $G$ , and  $OPT(G)$  is the optimal number of colors for  $G$ .

[Note: This does not contradict the possibility of a constant **factor** approximation algorithm.]

3. Let  $R$  be a set of rectangles in the plane, with horizontal and vertical edges. A *stabbing set* for  $R$  is a set of points  $S$  such that every rectangle in  $R$  contains at least one point in  $S$ . The *rectangle stabbing* problem asks, given a set  $R$  of rectangles, for the smallest stabbing set  $S$ .

- (a) Prove that the rectangle stabbing problem is NP-hard.
- (b) Describe and analyze an efficient approximation algorithm for the rectangle stabbing problem. Give bounds on the approximation ratio of your algorithm.

4. Consider the following approximation scheme for coloring a graph  $G$ .

```

TREECOLOR( $G$ ):
 $T \leftarrow$  any spanning tree of  $G$ 
Color the tree  $T$  with two colors
 $c \leftarrow 2$ 
for each edge  $(u, v) \in G \setminus T$ 
     $T \leftarrow T \cup \{(u, v)\}$ 
    if  $\text{color}(u) = \text{color}(v)$     «Try recoloring  $u$  with an existing color»
        for  $i \leftarrow 1$  to  $c$ 
            if no neighbor of  $u$  in  $T$  has color  $i$ 
                 $\text{color}(u) \leftarrow i$ 
    if  $\text{color}(u) = \text{color}(v)$     «Try recoloring  $v$  with an existing color»
        for  $i \leftarrow 1$  to  $c$ 
            if no neighbor of  $v$  in  $T$  has color  $i$ 
                 $\text{color}(v) \leftarrow i$ 
    if  $\text{color}(u) = \text{color}(v)$     «Give up and use a new color»
         $c \leftarrow c + 1$ 
         $\text{color}(u) \leftarrow c$ 
return  $c$ 
```

- (a) Prove that this algorithm correctly colors any bipartite graph.
- (b) Prove an upper bound  $C$  on the number of colors used by this algorithm. Give a sample graph and run that requires  $C$  colors.
- (c) Does this algorithm approximate the minimum number of colors up to a constant factor? In other words, is there a constant  $\alpha$  such that  $\text{TREECOLOR}(\mathcal{G}) < \alpha \cdot \text{OPT}(\mathcal{G})$  for any graph  $\mathcal{G}$ ? Justify your answer.

5. In the *bin packing* problem, we are given a set of  $n$  items, each with weight between 0 and 1, and we are asked to load the items into as few bins as possible, such that the total weight in each bin is at most 1. It's not hard to show that this problem is NP-Hard; this question asks you to analyze a few common approximation algorithms. In each case, the input is an array  $W[1..n]$  of weights, and the output is the number of bins used.

```
NEXTFIT( $W[1..n]$ ):
```

```

 $b \leftarrow 0$ 
 $Total[0] \leftarrow \infty$ 
for  $i \leftarrow 1$  to  $n$ 
  if  $Total[b] + W[i] > 1$ 
     $b \leftarrow b + 1$ 
     $Total[b] \leftarrow W[i]$ 
  else
     $Total[b] \leftarrow Total[b] + W[i]$ 
return  $b$ 
```

```
FIRSTFIT( $W[1..n]$ ):
```

```

 $b \leftarrow 0$ 
for  $i \leftarrow 1$  to  $n$ 
   $j \leftarrow 1$ ;  $found \leftarrow \text{FALSE}$ 
  while  $j \leq b$  and  $found = \text{FALSE}$ 
    if  $Total[j] + W[i] \leq 1$ 
       $Total[j] \leftarrow Total[j] + W[i]$ 
       $found \leftarrow \text{TRUE}$ 
       $j \leftarrow j + 1$ 
    if  $found = \text{FALSE}$ 
       $b \leftarrow b + 1$ 
       $Total[b] = W[i]$ 
return  $b$ 
```

- (a) Prove that NEXTFIT uses at most twice the optimal number of bins.
- (b) Prove that FIRSTFIT uses at most twice the optimal number of bins.
- (c) Prove that if the weight array  $W$  is initially sorted in decreasing order, then FIRSTFIT uses at most  $(4 \cdot OPT + 1)/3$  bins, where  $OPT$  is the optimal number of bins. The following facts may be useful (but you need to prove them if your proof uses them):
  - In the packing computed by FIRSTFIT, every item with weight more than  $1/3$  is placed in one of the first  $OPT$  bins.
  - FIRSTFIT places at most  $OPT - 1$  items outside the first  $OPT$  bins.

# CS 473G: Combinatorial Algorithms, Fall 2005

## Homework 4

Due Thursday, October 27, 2005, at midnight

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

---

Homeworks may be done in teams of up to three people. Each team turns in just one solution; every member of a team gets the same grade.

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Staple this sheet to the top of your solution to problem 1.

**If you are an I2CS student, print “(I2CS)” next to your name. Teams that include both on-campus and I2CS students can have up to four members. Any team containing both on-campus and I2CS students automatically receives 3 points of extra credit.**

---

For the rest of the semester, unless specifically stated otherwise, you may assume that the function  $\text{RANDOM}(m)$  returns an integer chosen uniformly at random from the set  $\{1, 2, \dots, m\}$  in  $O(1)$  time. For example, a fair coin flip is obtained by calling  $\text{RANDOM}(2)$ .

1. Consider the following randomized algorithm for choosing the largest bolt. Draw a bolt uniformly at random from the set of  $n$  bolts, and draw a nut uniformly at random from the set of  $n$  nuts. If the bolt is smaller than the nut, discard the bolt, draw a new bolt uniformly at random from the unchosen bolts, and repeat. Otherwise, discard the nut, draw a new nut uniformly at random from the unchosen nuts, and repeat. Stop either when every nut has been discarded, or every bolt except the one in your hand has been discarded.

What is the *exact* expected number of nut-bolt tests performed by this algorithm? Prove your answer is correct. [Hint: What is the expected number of unchosen nuts and bolts when the algorithm terminates?]

2. A *meldable priority queue* stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:

- **MAKEQUEUE**: Return a new priority queue containing the empty set.
- **FINDMIN( $Q$ )**: Return the smallest element of  $Q$  (if any).
- **DELETEMIN( $Q$ )**: Remove the smallest element in  $Q$  (if any).
- **INSERT( $Q, x$ )**: Insert element  $x$  into  $Q$ , if it is not already there.
- **DECREASEKEY( $Q, x, y$ )**: Replace an element  $x \in Q$  with a smaller key  $y$ . (If  $y > x$ , the operation fails.) The input is a pointer directly to the node in  $Q$  containing  $x$ .
- **DELETE( $Q, x$ )**: Delete the element  $x \in Q$ . The input is a pointer directly to the node in  $Q$  containing  $x$ .
- **MELD( $Q_1, Q_2$ )**: Return a new priority queue containing all the elements of  $Q_1$  and  $Q_2$ ; this operation destroys  $Q_1$  and  $Q_2$ .

A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm:

```

MELD( $Q_1, Q_2$ ):
  if  $Q_1$  is empty return  $Q_2$ 
  if  $Q_2$  is empty return  $Q_1$ 
  if  $key(Q_1) > key(Q_2)$ 
    swap  $Q_1 \leftrightarrow Q_2$ 
  with probability  $1/2$ 
     $left(Q_1) \leftarrow \text{MELD}(left(Q_1), Q_2)$ 
  else
     $right(Q_1) \leftarrow \text{MELD}(right(Q_1), Q_2)$ 
  return  $Q_1$ 
```

- (a) Prove that for *any* heap-ordered binary trees  $Q_1$  and  $Q_2$  (not just those constructed by the operations listed above), the expected running time of  $\text{MELD}(Q_1, Q_2)$  is  $O(\log n)$ , where  $n = |Q_1| + |Q_2|$ . [Hint: How long is a random root-to-leaf path in an  $n$ -node binary tree if each left/right choice is made with equal probability?]
- (b) Prove that  $\text{MELD}(Q_1, Q_2)$  runs in  $O(\log n)$  time with high probability.
- (c) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and  $O(1)$  additional time. (This implies that every operation takes  $O(\log n)$  time with high probability.)

3. Let  $M[1..n][1..n]$  be an  $n \times n$  matrix in which every row and every column is sorted. Such an array is called *totally monotone*. No two elements of  $M$  are equal.

- (a) Describe and analyze an algorithm to solve the following problem in  $O(n)$  time: Given indices  $i, j, i', j'$  as input, compute the number of elements of  $M$  smaller than  $M[i][j]$  and larger than  $M[i'][j']$ .
- (b) Describe and analyze an algorithm to solve the following problem in  $O(n)$  time: Given indices  $i, j, i', j'$  as input, return an element of  $M$  chosen uniformly at random from the elements smaller than  $M[i][j]$  and larger than  $M[i'][j']$ . Assume the requested range is always non-empty.
- (c) Describe and analyze a randomized algorithm to compute the median element of  $M$  in  $O(n \log n)$  expected time.

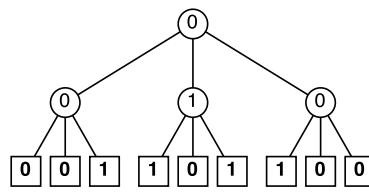
4. Let  $X[1..n]$  be an array of  $n$  distinct real numbers, and let  $N[1..n]$  be an array of indices with the following property: If  $X[i]$  is the largest element of  $X$ , then  $X[N[i]]$  is the smallest element of  $X$ ; otherwise,  $X[N[i]]$  is the smallest element of  $X$  that is larger than  $X[i]$ .

For example:

$i$	1	2	3	4	5	6	7	8	9
$X[i]$	83	54	16	31	45	99	78	62	27
$N[i]$	6	8	9	5	2	3	1	7	4

Describe and analyze a randomized algorithm that determines whether a given number  $x$  appears in the array  $X$  in  $O(\sqrt{n})$  expected time. **Your algorithm may not modify the arrays  $X$  and  $Next$ .**

5. A *majority tree* is a complete ternary tree with depth  $n$ , where every leaf is labeled either 0 or 1. The *value* of a leaf is its label; the *value* of any internal node is the majority of the values of its three children. Consider the problem of computing the value of the root of a majority tree, given the sequence of  $3^n$  leaf labels as input. For example, if  $n = 2$  and the leaves are labeled 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, the root has value 0.



A majority tree with depth  $n = 2$ .

- (a) Prove that *any* deterministic algorithm that computes the value of the root of a majority tree *must* examine every leaf. [Hint: Consider the special case  $n = 1$ . Recurse.]
- (b) Describe and analyze a randomized algorithm that computes the value of the root in worst-case expected time  $O(c^n)$  for some constant  $c < 3$ . [Hint: Consider the special case  $n = 1$ . Recurse.]

\*6. [Extra credit] In the usual theoretical presentation of treaps, the priorities are random real numbers chosen uniformly from the interval  $[0, 1]$ , but in practice, computers only have access to random *bits*. This problem asks you to analyze a modification of treaps that takes this limitation into account.

Suppose the priority of a node  $v$  is abstractly represented as an infinite sequence  $\pi_v[1 .. \infty]$  of random bits, which is interpreted as the rational number

$$\text{priority}(v) = \sum_{i=1}^{\infty} \pi_v[i] \cdot 2^{-i}.$$

However, only a finite number  $\ell_v$  of these bits are actually known at any given time. When a node  $v$  is first created, *none* of the priority bits are known:  $\ell_v = 0$ . We generate (or ‘reveal’) new random bits only when they are necessary to compare priorities. The following algorithm compares the priorities of any two nodes in  $O(1)$  expected time:

```
LARGERPRIORITY( $v, w$ ):
for  $i \leftarrow 1$  to  $\infty$ 
    if  $i > \ell_v$ 
         $\ell_v \leftarrow i$ ;  $\pi_v[i] \leftarrow \text{RANDOMBIT}$ 
    if  $i > \ell_w$ 
         $\ell_w \leftarrow i$ ;  $\pi_w[i] \leftarrow \text{RANDOMBIT}$ 
    if  $\pi_v[i] > \pi_w[i]$ 
        return  $v$ 
    else if  $\pi_v[i] < \pi_w[i]$ 
        return  $w$ 
```

Suppose we insert  $n$  items one at a time into an initially empty treap. Let  $L = \sum_v \ell_v$  denote the total number of random bits generated by calls to LARGERPRIORITY during these insertions.

- (a) Prove that  $E[L] = \Theta(n)$ .
- (b) Prove that  $E[\ell_v] = \Theta(1)$  for any node  $v$ . [Hint: This is equivalent to part (a). Why?]
- (c) Prove that  $E[\ell_{\text{root}}] = \Theta(\log n)$ . [Hint: Why doesn’t this contradict part (b)?]

# CS 473G: Combinatorial Algorithms, Fall 2005

## Homework 5

Due Thursday, November 17, 2005, at midnight

(because you *really* don't want homework due over Thanksgiving break)

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

---

Homeworks may be done in teams of up to three people. Each team turns in just one solution; every member of a team gets the same grade.

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Attach this sheet (or the equivalent information) to the top of your solution to problem 1.

**If you are an I2CS student, print “(I2CS)” next to your name. Teams that include both on-campus and I2CS students can have up to four members. Any team containing both on-campus and I2CS students automatically receives 3 points of extra credit.**

Problems labeled  $\heartsuit$  are likely to require techniques from next week's lectures on cuts, flows, and matchings. See also Chapter 7 in Kleinberg and Tardos, or Chapter 26 in CLRS.

---

- $\heartsuit$  1. Suppose you are asked to construct the minimum spanning tree of a graph  $G$ , but you are not completely sure of the edge weights. Specifically, you have a *conjectured* weight  $\tilde{w}(e)$  for every edge  $e$  in the graph, but you also know that up to  $k$  of these conjectured weights are wrong. With the exception of one edge  $e$  whose true weight you know exactly, you don't know which edges are wrong, or even how they're wrong; the true weights of those edges could be larger or smaller than the conjectured weights. Given this unreliable information, it is of course impossible to reliably construct the true minimum spanning tree of  $G$ , but it is still possible to say something about your special edge.

Describe and analyze an efficient algorithm to determine whether a specific edge  $e$ , whose *actual* weight is known, is *definitely not* in the minimum spanning tree of  $G$  under the stated conditions. The input consists of the graph  $G$ , the conjectured weight function  $\tilde{w} : E(G) \rightarrow \mathbb{R}$ , the positive integer  $k$ , and the edge  $e$ .

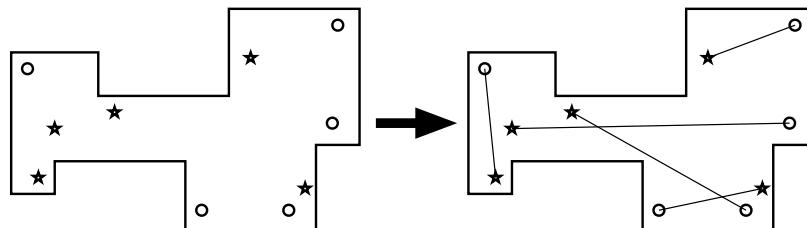
2. Most classical minimum-spanning-tree algorithms use the notions of ‘safe’ and ‘useless’ edges described in the lecture notes, but there is an alternate formulation. Let  $G$  be a weighted undirected graph, where the edge weights are distinct. We say that an edge  $e$  is *dangerous* if it is the longest edge in some cycle in  $G$ , and *useful* if it does not lie in any cycle in  $G$ .

- (a) Prove that the minimum spanning tree of  $G$  contains every useful edge.
- (b) Prove that the minimum spanning tree of  $G$  does not contain any dangerous edge.
- (c) Describe and analyze an efficient implementation of the “anti-Kruskal” MST algorithm: Examine the edges of  $G$  in *decreasing* order; if an edge is dangerous, remove it from  $G$ . [Hint: It won’t be as fast as the algorithms you saw in class.]

γ 3. The UIUC Computer Science department has decided to build a mini-golf course in the basement of the Siebel Center! The playing field is a closed polygon bounded by  $m$  horizontal and vertical line segments, meeting at right angles. The course has  $n$  *starting points* and  $n$  *holes*, in one-to-one correspondence. It is always possible hit the ball along a straight line directly from each starting point to the corresponding hole, without touching the boundary of the playing field. (Players are not allowed to bounce golf balls off the walls; too much glass.) The  $n$  starting points and  $n$  holes are all at distinct locations.

Sadly, the architect’s computer crashed just as construction was about to begin. Thanks to the herculean efforts of their sysadmins, they were able to recover the *locations* of the starting points and the holes, but all information about which starting points correspond to which holes was lost!

Describe and analyze an algorithm to compute a one-to-one correspondence between the starting points and the holes that meets the straight-line requirement, or to report that no such correspondence exists. The input consists of the  $x$ - and  $y$ -coordinates of the  $m$  corners of the playing field, the  $n$  starting points, and the  $n$  holes. Assume you can determine in constant time whether two line segments intersect, given the  $x$ - and  $y$ -coordinates of their endpoints.



A minigolf course with five starting points ( $\star$ ) and five holes ( $\circ$ ), and a legal correspondence between them.

γ 4. Let  $G = (V, E)$  be a directed graph where the in-degree of each vertex is equal to its out-degree. Prove or disprove the following claim: For any two vertices  $u$  and  $v$  in  $G$ , the number of mutually edge-disjoint paths from  $u$  to  $v$  is equal to the number of mutually edge-disjoint paths from  $v$  to  $u$ .

5. You are given a set of  $n$  boxes, each specified by its height, width, and depth. The order of the dimensions is unimportant; for example, a  $1 \times 2 \times 3$  box is exactly the same as a  $3 \times 1 \times 2$  box or a  $2 \times 1 \times 3$  box. You can nest box  $A$  inside box  $B$  if and only if  $A$  can be rotated so that it has strictly smaller height, strictly smaller width, and strictly smaller depth than  $B$ .
- (a) Design and analyze an efficient algorithm to determine the largest sequence of boxes that can be nested inside one another. [*Hint: Model the nesting relationship as a graph.*]
  - γ (b) Describe and analyze an efficient algorithm to nest all  $n$  boxes into as few groups as possible, where each group consists of a nested sequence. You are not allowed to put two boxes side-by-side inside a third box, even if they are small enough to fit.<sup>1</sup> [*Hint: Model the nesting relationship as a **different** graph.*]
6. [*Extra credit*] Prove that Ford's generic shortest-path algorithm (described in the lecture notes) can take exponential time in the worst case when implemented with a stack instead of a heap (like Dijkstra) or a queue (like Bellman-Ford). Specifically, construct for every positive integer  $n$  a weighted directed  $n$ -vertex graph  $G_n$ , such that the stack-based shortest-path algorithm call RELAX  $\Omega(2^n)$  times when  $G_n$  is the input graph. [*Hint: Towers of Hanoi.*]

---

<sup>1</sup>Without this restriction, the problem is NP-hard, even for one-dimensional “boxes”.

# CS 473G: Combinatorial Algorithms, Fall 2005

## Homework 6

Practice only; nothing to turn in.

---

1. A small airline, Ivy Air, flies between three cities: Ithaca (a small town in upstate New York), Newark (an eyesore in beautiful New Jersey), and Boston (a yuppie town in Massachusetts). They offer several flights but, for this problem, let us focus on the Friday afternoon flight that departs from Ithaca, stops in Newark, and continues to Boston. There are three types of passengers:

- (a) Those traveling from Ithaca to Newark (god only knows why).
- (b) Those traveling from Newark to Boston (a very good idea).
- (c) Those traveling from Ithaca to Boston (it depends on who you know).

The aircraft is a small commuter plane that seats 30 passengers. The airline offers three fare classes:

- (a) Y class: full coach.
- (b) B class: nonrefundable.
- (c) M class: nonrefundable, 3-week advanced purchase.

Ticket prices, which are largely determined by external influences (i.e., competitors), have been set and advertised as follows:

	Ithaca-Newark	Newark-Boston	Ithaca-Boston
Y	300	160	360
B	220	130	280
M	100	80	140

Based on past experience, demand forecasters at Ivy Air have determined the following upper bounds on the number of potential customers in each of the 9 possible origin-destination/fare-class combinations:

	Ithaca-Newark	Newark-Boston	Ithaca-Boston
Y	4	8	3
B	8	13	10
M	22	20	18

The goal is to decide how many tickets from each of the 9 origin/destination/fare-class combinations to sell. The constraints are that the place cannot be overbooked on either the two legs of the flight and that the number of tickets made available cannot exceed the forecasted maximum demand. The objective is to maximize the revenue.

Formulate this problem as a linear programming problem.

2. (a) Suppose we are given a directed graph  $G = (V, E)$ , a length function  $\ell : E \rightarrow \mathbb{R}$ , and a source vertex  $s \in V$ . Write a linear program to compute the shortest-path distance from  $s$  to every other vertex in  $V$ . [Hint: Define a variable for each vertex representing its distance from  $s$ . What objective function should you use?]
- (b) In the *minimum-cost multicommodity-flow* problem, we are given a directed graph  $G = (V, E)$ , in which each edge  $u \rightarrow v$  has an associated nonnegative *capacity*  $c(u \rightarrow v) \geq 0$  and an associated *cost*  $\alpha(u \rightarrow v)$ . We are given  $k$  different commodities, each specified by a triple  $K_i = (s_i, t_i, d_i)$ , where  $s_i$  is the source node of the commodity,  $t_i$  is the target node for the commodity  $i$ , and  $d_i$  is the *demand*: the desired flow of commodity  $i$  from  $s_i$  to  $t_i$ . A *flow* for commodity  $i$  is a non-negative function  $f_i : E \rightarrow \mathbb{R}_{\geq 0}$  such that the total flow into any vertex other than  $s_i$  or  $t_i$  is equal to the total flow out of that vertex. The *aggregate flow*  $F : E \rightarrow \mathbb{R}$  is defined as the sum of these individual flows:  $F(u \rightarrow v) = \sum_{i=1}^k f_i(u \rightarrow v)$ . The aggregate flow  $F(u \rightarrow v)$  on any edge must not exceed the capacity  $c(u \rightarrow v)$ . The goal is to find an aggregate flow whose total *cost*  $\sum_{u \rightarrow v} F(u \rightarrow v) \cdot \alpha(u \rightarrow v)$  is as small as possible. (Costs may be negative!) Express this problem as a linear program.
3. In class we described the duality transformation only for linear programs in canonical form:

$$\begin{array}{ccc} \text{Primal (II)} & \iff & \text{Dual (II)} \\ \boxed{\begin{array}{ll} \max & c \cdot x \\ \text{s.t.} & Ax \leq b \\ & x \geq 0 \end{array}} & & \boxed{\begin{array}{ll} \min & y \cdot b \\ \text{s.t.} & yA \geq c \\ & y \geq 0 \end{array}} \end{array}$$

Describe precisely how to dualize the following more general linear programming problem:

$$\begin{aligned} & \text{maximize} \quad \sum_{j=1}^d c_j x_j \\ & \text{subject to} \quad \sum_{j=1}^d a_{ij} x_j \leq b_i \quad \text{for each } i = 1..p \\ & \quad \sum_{j=1}^d a_{ij} x_j = b_i \quad \text{for each } i = p+1..p+q \\ & \quad \sum_{j=1}^d a_{ij} x_j \geq b_i \quad \text{for each } i = p+q+1..n \end{aligned}$$

Your dual problem should have one variable for each primal constraint, and the dual of your dual program should be precisely the original linear program.

4. (a) Model the maximum-cardinality bipartite matching problem as a linear programming problem. The input is a bipartite graph  $G = (U, V; E)$ , where  $E \subseteq U \times V$ ; the output is the largest matching in  $G$ . Your linear program should have one variable for every edge.
- (b) Now dualize the linear program from part (a). What do the dual variables represent? What does the objective function represent? What problem is this!?

5. An *integer program* is a linear program with the additional constraint that the variables must take only integer values.

- (a) Prove that deciding whether an integer program has a feasible solution is NP-complete.
- (b) Prove that finding the optimal feasible solution to an integer program is NP-hard.

*[Hint: Almost any NP-hard decision problem can be rephrased as an integer program. Pick your favorite.]*

6. Consider the LP formulation of the shortest path problem presented in class:

$$\begin{aligned} & \text{maximize} && d_t \\ & \text{subject to} && d_s = 0 \\ & && d_v - d_u \leq \ell_{u \rightarrow v} \quad \text{for every edge } u \rightarrow v \end{aligned}$$

Characterize the feasible bases for this linear program in terms of the original weighted graph. What does a simplex pivoting operation represent? What is a locally optimal (*i.e.*, dual feasible) basis? What does a dual pivoting operation represent?

7. Consider the LP formulation of the maximum-flow problem presented in class:

$$\begin{aligned} & \text{maximize} && \sum_w f_{s \rightarrow w} - \sum_u f_{u \rightarrow s} \\ & \text{subject to} && \sum_w f_{v \rightarrow w} - \sum_u f_{u \rightarrow v} = 0 \quad \text{for every vertex } v \neq s, t \\ & && f_{u \rightarrow v} \leq c_{u \rightarrow v} \quad \text{for every edge } u \rightarrow v \\ & && f_{u \rightarrow v} \geq 0 \quad \text{for every edge } u \rightarrow v \end{aligned}$$

Is the Ford-Fulkerson augmenting path algorithm an instance of the simplex algorithm applied to this linear program? Why or why not?

\*8. *Helly's theorem* says that for any collection of convex bodies in  $\mathbb{R}^n$ , if every  $n + 1$  of them intersect, then there is a point lying in the intersection of all of them. Prove Helly's theorem for the special case that the convex bodies are halfspaces. *[Hint: Show that if a system of inequalities  $Ax \geq b$  does not have a solution, then we can select  $n + 1$  of the inequalities such that the resulting system does not have a solution. Construct a primal LP from the system by choosing a 0 cost vector.]*

You have 90 minutes to answer four of these questions.

**Write your answers in the separate answer booklet.**

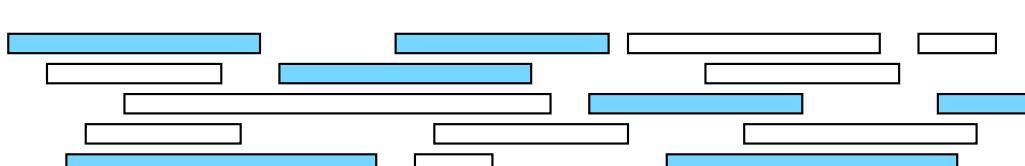
You may take the question sheet with you when you leave.

1. You and your eight-year-old nephew Elmo decide to play a simple card game. At the beginning of the game, the cards are dealt face up in a long row. Each card is worth a different number of points. After all the cards are dealt, you and Elmo take turns removing either the leftmost or rightmost card from the row, until all the cards are gone. At each turn, you can decide which of the two cards to take. The winner of the game is the player that has collected the most points when the game ends.

Having never taken an algorithms class, Elmo follows the obvious greedy strategy—when it's his turn, Elmo *always* takes the card with the higher point value. Your task is to find a strategy that will beat Elmo whenever possible. (It might seem mean to beat up on a little kid like this, but Elmo absolutely *hates* it when grown-ups let him win.)

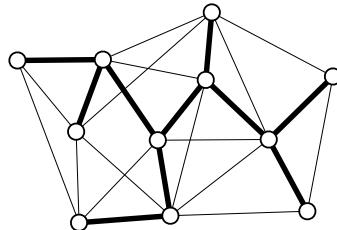
- (a) Prove that you should not also use the greedy strategy. That is, show that there is a game that you can win, but only if you do *not* follow the same greedy strategy as Elmo.
  - (b) Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that you can collect playing against Elmo.
- 
2. Suppose you are given a magical black box that can tell you in constant time whether or not a given graph has a Hamiltonian cycle. Using this magic black box as a subroutine, describe and analyze a *polynomial-time* algorithm to actually compute a Hamiltonian cycle in a given graph, if one exists.
- 
3. Let  $X$  be a set of  $n$  intervals on the real line. A subset of intervals  $Y \subseteq X$  is called a *tiling path* if the intervals in  $Y$  cover the intervals in  $X$ , that is, any real value that is contained in some interval in  $X$  is also contained in some interval in  $Y$ . The *size* of a tiling cover is just the number of intervals.

Describe and analyze an algorithm to compute the smallest tiling path of  $X$  as quickly as possible. Assume that your input consists of two arrays  $X_L[1..n]$  and  $X_R[1..n]$ , representing the left and right endpoints of the intervals in  $X$ .



A set of intervals. The seven shaded intervals form a tiling path.

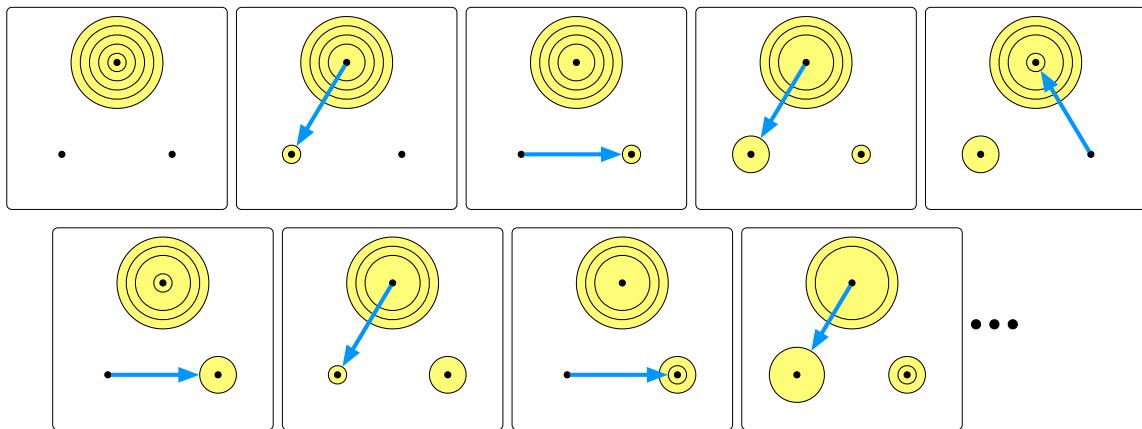
4. Prove that the following problem is NP-complete: Given an undirected graph, does it have a spanning tree in which every node has degree at most 3?



A graph with a spanning tree of maximum degree 3.

5. The *Tower of Hanoi* puzzle, invented by Edouard Lucas in 1883, consists of three pegs and  $n$  disks of different sizes. Initially, all  $n$  disks are on the same peg, stacked in order by size, with the largest disk on the bottom and the smallest disk on top. In a single move, you can move the topmost disk on any peg to another peg; however, you are never allowed to place a larger disk on top of a smaller one. Your goal is to move all  $n$  disks to a different peg.

- (a) Prove that the Tower of Hanoi puzzle can be solved in exactly  $2^n - 1$  moves. [Hint: You've probably seen this before.]
- (b) Now suppose the pegs are arranged in a circle and you are *only* allowed to move disks *counterclockwise*. How many moves do you need to solve this restricted version of the puzzle? Give an upper bound in the form  $O(f(n))$  for some function  $f(n)$ . Prove your upper bound is correct.



A top view of the first eight moves in a clockwise Towers of Hanoi solution

You have 90 minutes to answer four of these questions.

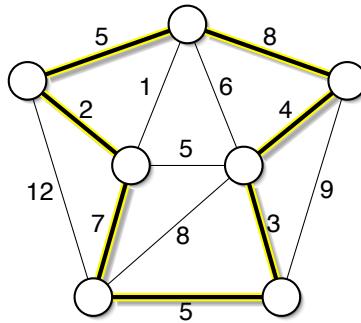
**Write your answers in the separate answer booklet.**

You may take the question sheet with you when you leave.

**Chernoff Bounds:** If  $X$  is the sum of independent indicator variables and  $\mu = \mathbb{E}[X]$ , then the following inequalities hold for any  $\delta > 0$ :

$$\Pr[X < (1 - \delta)\mu] < \left(\frac{e^{-\delta}}{(1 - \delta)^{1-\delta}}\right)^\mu \quad \Pr[X > (1 + \delta)\mu] < \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}}\right)^\mu$$

1. Describe and analyze an algorithm that randomly shuffles an array  $X[1..n]$ , so that each of the  $n!$  possible permutations is equally likely, in  $O(n)$  time. (Assume that the subroutine  $\text{RANDOM}(m)$  returns an integer chosen uniformly at random from the set  $\{1, 2, \dots, m\}$  in  $O(1)$  time.)
2. Let  $G$  be an undirected graph with weighted edges. A *heavy Hamiltonian cycle* is a cycle  $C$  that passes through each vertex of  $G$  exactly once, such that the total weight of the edges in  $C$  is at least half of the total weight of all edges in  $G$ . Prove that deciding whether a graph has a heavy Hamiltonian cycle is NP-complete.



A heavy Hamiltonian cycle. The cycle has total weight 34; the graph has total weight 67.

3. A sequence of numbers  $\langle a_1, a_2, a_3, \dots, a_n \rangle$  is *oscillating* if  $a_i < a_{i+1}$  for every odd index  $i$  and  $a_i > a_{i+1}$  for every even index  $i$ . Describe and analyze an efficient algorithm to compute the longest oscillating subsequence in a sequence of  $n$  integers.
4. This problem asks you to how to efficiently modify a maximum flow if one of the edge capacities changes. Specifically, you are given a directed graph  $G = (V, E)$  with capacities  $c : E \rightarrow \mathbb{Z}_+$ , and a maximum flow  $F : E \rightarrow \mathbb{Z}$  from some vertex  $s$  to some other vertex  $t$  in  $G$ . Describe and analyze efficient algorithms for the following operations:
  - (a)  $\text{INCREMENT}(e)$  — Increase the capacity of edge  $e$  by 1 and update the maximum flow  $F$ .
  - (b)  $\text{DECREMENT}(e)$  — Decrease the capacity of edge  $e$  by 1 and update the maximum flow  $F$ .

Both of your algorithms should be significantly faster than recomputing the maximum flow from scratch.

5.

6. Let  $G = (V, E)$  be an undirected graph, each of whose vertices is colored either red, green, or blue. An edge in  $G$  is *boring* if its endpoints have the same color, and *interesting* if its endpoints have different colors. The *most interesting 3-coloring* is the 3-coloring with the maximum number of interesting edges, or equivalently, with the fewest boring edges.

- (a) Prove that it is NP-hard to compute the most interesting 3-coloring of a graph. [Hint: There is a one-line proof. Use one of the NP-hard problems described in class.]
- (b) Let  $zzz(G)$  denote the number of boring edges in the most interesting 3-coloring of a graph  $G$ . Prove that it is NP-hard to approximate  $zzz(G)$  within a factor of  $10^{10^{100}}$ . [Hint: There is a one-line proof.]
- (c) Let  $wow(G)$  denote the number of interesting edges in the most interesting 3-coloring of  $G$ . Suppose we assign each vertex in  $G$  a *random* color from the set {red, green, blue}. Prove that the expected number of interesting edges is at least  $\frac{2}{3}wow(G)$ .

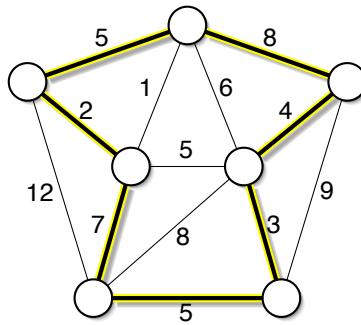
7.

You have 180 minutes to answer six of these questions.

**Write your answers in the separate answer booklet.**

You may take the question sheet with you when you leave.

1. Describe and analyze an algorithm that randomly shuffles an array  $X[1..n]$ , so that each of the  $n!$  possible permutations is equally likely, in  $O(n)$  time. (Assume that the subroutine  $\text{RANDOM}(m)$  returns an integer chosen uniformly at random from the set  $\{1, 2, \dots, m\}$  in  $O(1)$  time.)
2. Let  $G$  be an undirected graph with weighted edges. A *heavy Hamiltonian cycle* is a cycle  $C$  that passes through each vertex of  $G$  exactly once, such that the total weight of the edges in  $C$  is at least half of the total weight of all edges in  $G$ . Prove that deciding whether a graph has a heavy Hamiltonian cycle is NP-complete.



A heavy Hamiltonian cycle. The cycle has total weight 34; the graph has total weight 67.

3. Suppose you are given a directed graph  $G = (V, E)$  with capacities  $c : E \rightarrow \mathbb{Z}_+$  and a maximum flow  $F : E \rightarrow \mathbb{Z}$  from some vertex  $s$  to some other vertex  $t$  in  $G$ . Describe and analyze efficient algorithms for the following operations:

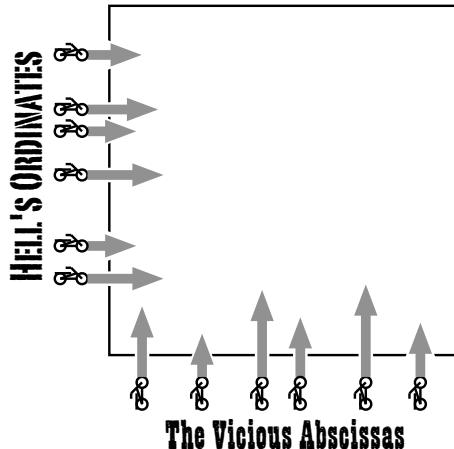
- (a)  $\text{INCREMENT}(e)$  — Increase the capacity of edge  $e$  by 1 and update the maximum flow  $F$ .
- (b)  $\text{DECREMENT}(e)$  — Decrease the capacity of edge  $e$  by 1 and update the maximum flow  $F$ .

Both of your algorithms should be significantly faster than recomputing the maximum flow from scratch.

4. Suppose you are given an undirected graph  $G$  and two vertices  $s$  and  $t$  in  $G$ . Two paths from  $s$  to  $t$  are *vertex-disjoint* if the only vertices they have in common are  $s$  and  $t$ . Describe and analyze an efficient algorithm to compute the maximum number of vertex-disjoint paths between  $s$  and  $t$  in  $G$ . [Hint: Reduce this to a more familiar problem on a suitable directed graph  $G'$ .]

5. A sequence of numbers  $\langle a_1, a_2, a_3, \dots, a_n \rangle$  is *oscillating* if  $a_i < a_{i+1}$  for every *odd* index  $i$  and  $a_i > a_{i+1}$  for every *even* index  $i$ . For example, the sequence  $\langle 2, 7, 1, 8, 2, 8, 1, 8, 3 \rangle$  is oscillating. Describe and analyze an efficient algorithm to compute the longest oscillating subsequence in a sequence of  $n$  integers.
6. Let  $G = (V, E)$  be an undirected graph, each of whose vertices is colored either red, green, or blue. An edge in  $G$  is *boring* if its endpoints have the same color, and *interesting* if its endpoints have different colors. The *most interesting 3-coloring* is the 3-coloring with the maximum number of interesting edges, or equivalently, with the fewest boring edges. Computing the most interesting 3-coloring is NP-hard, because the standard 3-coloring problem we saw in class is a special case.
- Let  $zzz(G)$  denote the number of boring edges in the most interesting 3-coloring of a graph  $G$ . Prove that it is NP-hard to approximate  $zzz(G)$  within a factor of  $10^{10^{100}}$ .
  - Let  $wow(G)$  denote the number of interesting edges in the most interesting 3-coloring of  $G$ . Suppose we assign each vertex in  $G$  a *random* color from the set {red, green, blue}. Prove that the expected number of interesting edges is at least  $\frac{2}{3}wow(G)$ .
7. It's time for the 3rd Quasi-Annual Champaign-Urbana Ice Motorcycle Demolition Derby Race-O-Rama and Spaghetti Bake-Off! The main event is a competition between two teams of  $n$  motorcycles in a huge square ice-covered arena. All of the motorcycles have spiked tires so that they can ride on the ice. Each motorcycle drags a long metal chain behind it. Whenever a motorcycle runs over a chain, the chain gets caught in the tire spikes, and the motorcycle crashes. Two motorcycles can also crash by running directly into each other. All the motorcycle start simultaneously. Each motorcycle travels in a straight line at a constant speed until it either crashes or reaches the opposite wall—no turning, no braking, no speeding up, no slowing down. The Vicious Abscissas start at the south wall of the arena and ride directly north (vertically). Hell's Ordinates start at the west wall of the arena and ride directly east (horizontally). If any motorcycle completely crosses the arena, that rider's entire team wins the competition.

Describe and analyze an efficient algorithm to decide which team will win, given the starting position and speed of each motorcycle.



# CS 473U: Undergraduate Algorithms, Fall 2006

## Homework 0

Due Friday, September 1, 2006 at noon in 3229 Siebel Center

Name:	
Net ID:	Alias:

I understand the Homework Instructions and FAQ.

- 
- Neatly print your full name, your NetID, and an alias of your choice in the boxes above, and submit this page with your solutions. We will list homework and exam grades on the course web site by alias. For privacy reasons, your alias should not resemble your name, your NetID, your university ID number, or (God forbid) your Social Security number. Please use the same alias for every homework and exam.

Federal law forbids us from publishing your grades, even anonymously, without your explicit permission. **By providing an alias, you grant us permission to list your grades on the course web site; if you do not provide an alias, your grades will not be listed.**

- Please carefully read the Homework Instructions and FAQ on the course web page, and then check the box above. This page describes what we expect in your homework solutions—start each numbered problem on a new sheet of paper, write your name and NetID on every page, don’t turn in source code, analyze and prove everything, use good English and good logic, and so on—as well as policies on grading standards, regrading, and plagiarism. **See especially the policies regarding the magic phrases “I don’t know” and “and so on”.** If you have *any* questions, post them to the course newsgroup or ask in lecture.
  - This homework tests your familiarity with prerequisite material—basic data structures, big-Oh notation, recurrences, discrete probability, and most importantly, induction—to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Each numbered problem is worth 10 points; not all subproblems have equal weight.
- 

#	1	2	3	4	5	6*	Total
Score							
Grader							

Please put your answers to problems 1 and 2 on the same page.

1. Sort the functions listed below from asymptotically smallest to asymptotically largest, indicating ties if there are any. **Do not turn in proofs**, but you should probably do them anyway, just for practice.

To simplify your answers, write  $f(n) \ll g(n)$  to mean  $f(n) = o(g(n))$ , and write  $f(n) \equiv g(n)$  to mean  $f(n) = \Theta(g(n))$ . For example, the functions  $n^2, n, \binom{n}{2}, n^3$  could be sorted either as  $n \ll n^2 \equiv \binom{n}{2} \ll n^3$  or as  $n \ll \binom{n}{2} \equiv n^2 \ll n^3$ .

$$\begin{array}{cccccccc}
 \lg n & \ln n & \sqrt{n} & n & n \lg n & n^2 & 2^n & n^{1/n} \\
 n^{1+1/\lg n} & \lg^{1000} n & 2^{\sqrt{\lg n}} & (\sqrt{2})^{\lg n} & \lg^{\sqrt{2}} n & n^{\sqrt{2}} & (1 + \frac{1}{n})^n & n^{1/1000} \\
 H_n & H_{\sqrt{n}} & 2^{H_n} & H_{2^n} & F_n & F_{n/2} & \lg F_n & F_{\lg n}
 \end{array}$$

In case you've forgotten:

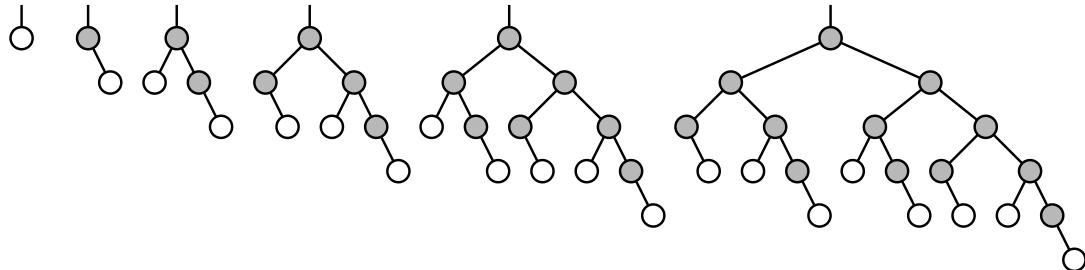
- $\lg n = \log_2 n \neq \ln n = \log_e n$
- $\lg^3 n = (\lg n)^3 \neq \lg \lg \lg n$ .
- The harmonic numbers:  $H_n = \sum_{i=1}^n 1/i \approx \ln n + 0.577215\dots$
- The Fibonacci numbers:  $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$  for all  $n \geq 2$

2. Solve the following recurrences. State tight asymptotic bounds for each function in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ . Proofs are *not* required; just give us the list of answers. **Don't turn in proofs**, but you should do them anyway, just for practice. Assume reasonable but nontrivial base cases. **If your solution requires specific base cases, state them.** Extra credit will be awarded for more exact solutions.

- (a)  $A(n) = 2A(n/4) + \sqrt{n}$
- (b)  $B(n) = 3B(n/3) + n/\lg n$
- (c)  $C(n) = \frac{2C(n-1)}{C(n-2)}$  [Hint: This is easy!]
- (d)  $D(n) = D(n-1) + 1/n$
- (e)  $E(n) = E(n/2) + D(n)$
- (f)  $F(n) = 4F\left(\left\lceil \frac{n-8}{2} \right\rceil + \left\lfloor \frac{3n}{\log_\pi n} \right\rfloor\right) + 6\binom{n+5}{2} - 42n \lg^7 n + \sqrt{13n-6} + \frac{\lg \lg n + 1}{\lg n \lg \lg \lg n}$
- (g)  $G(n) = 2G(n-1) - G(n-2) + n$
- (h)  $H(n) = 2H(n/2) - 2H(n/4) + 2^n$
- (i)  $I(n) = I(n/2) + I(n/4) + I(n/6) + I(n/12) + n$
- ★(j)  $J(n) = \sqrt{n} \cdot J(2\sqrt{n}) + n$   
[Hint: First solve the secondary recurrence  $j(n) = 1 + j(2\sqrt{n})$ .]

3. The  $n$ th *Fibonacci binary tree*  $\mathcal{F}_n$  is defined recursively as follows:

- $\mathcal{F}_1$  is a single root node with no children.
- For all  $n \geq 2$ ,  $\mathcal{F}_n$  is obtained from  $\mathcal{F}_{n-1}$  by adding a right child to every leaf and adding a left child to every node that has only one child.



The first six Fibonacci binary trees. In each tree  $\mathcal{F}_n$ , the subtree of gray nodes is  $\mathcal{F}_{n-1}$ .

- Prove that the number of leaves in  $\mathcal{F}_n$  is precisely the  $n$ th Fibonacci number:  $F_0 = 0$ ,  $F_1 = 1$ , and  $F_n = F_{n-1} + F_{n-2}$  for all  $n \geq 2$ .
- How many nodes does  $\mathcal{F}_n$  have? For full credit, give an *exact*, closed-form answer in terms of Fibonacci numbers, and prove your answer is correct.
- Prove that the left subtree of  $\mathcal{F}_n$  is a copy of  $\mathcal{F}_{n-2}$ .

4. Describe and analyze a data structure that stores set of  $n$  records, each with a numerical *key* and a numerical *priority*, such that the following operation can be performed quickly:

$\text{RANGETOP}(a, z) : \text{return the highest-priority record whose key is between } a \text{ and } z.$

For example, if the (key, priority) pairs are

$$(3, 1), (4, 9), (9, 2), (6, 3), (5, 8), (7, 5), (1, 4), (0, 7),$$

then  $\text{RANGETOP}(2, 8)$  would return the record with key 4 and priority 9 (the second record in the list).

You may assume that no two records have equal keys or equal priorities, and that no record has a key equal to  $a$  or  $z$ . Analyze both the size of your data structure and the running time of your  $\text{RANGETOP}$  algorithm. For full credit, your data structure must be as small as possible and your  $\text{RANGETOP}$  algorithm must be as fast as possible.

[Hint: How would you compute the number of keys between  $a$  and  $z$ ? How would you solve the problem if you knew that  $a$  is always  $-\infty$ ?]

5. Penn and Teller agree to play the following game. Penn shuffles a standard deck<sup>1</sup> of playing cards so that every permutation is equally likely. Then Teller draws cards from the deck, one at a time without replacement, until he draws the three of clubs ( $3\clubsuit$ ), at which point the remaining undrawn cards instantly burst into flames.

The first time Teller draws a card from the deck, he gives it to Penn. From then on, until the game ends, whenever Teller draws a card whose value is smaller than the last card he gave to Penn, he gives the new card to Penn.<sup>2</sup> To make the rules unambiguous, they agree beforehand that  $A = 1$ ,  $J = 11$ ,  $Q = 12$ , and  $K = 13$ .

- (a) What is the expected number of cards that Teller draws?
- (b) What is the expected *maximum* value among the cards Teller gives to Penn?
- (c) What is the expected *minimum* value among the cards Teller gives to Penn?
- (d) What is the expected number of cards that Teller gives to Penn?

Full credit will be given only for *exact* answers (with correct proofs, of course).

\*6. [Extra credit]<sup>3</sup>

*Lazy binary* is a variant of standard binary notation for representing natural numbers where we allow each “bit” to take on one of three values: 0, 1, or 2. Lazy binary notation is defined inductively as follows.

- The lazy binary representation of zero is 0.
- Given the lazy binary representation of any non-negative integer  $n$ , we can construct the lazy binary representation of  $n + 1$  as follows:
  - (a) increment the rightmost digit;
  - (b) if any digit is equal to 2, replace the rightmost 2 with 0 and increment the digit immediately to its left.

Here are the first several natural numbers in lazy binary notation:

0, 1, 10, 11, 20, 101, 110, 111, 120, 201, 210, 1011, 1020, 1101, 1110, 1111, 1120, 1201, 1210, 2011, 2020, 2101, 2110, 10111, 10120, 10201, 10210, 11011, 11020, 11101, 11110, 11111, 11120, 11201, 11210, 12011, 12020, 12101, 12110, 20111, 20120, 20201, 20210, 21011, 21020, 21101, 21110, 101111, 101120, 101201, 101210, 102011, 102020, 102101, 102110, ...

- (a) Prove that in any lazy binary number, between any two 2s there is at least one 0, and between two 0s there is at least one 2.
- (b) Prove that for any natural number  $N$ , the sum of the digits of the lazy binary representation of  $N$  is exactly  $\lfloor \lg(N + 1) \rfloor$ .

---

<sup>1</sup>In a standard deck of 52 cards, each card has a *suit* in the set  $\{\spadesuit, \heartsuit, \clubsuit, \diamondsuit\}$  and a *value* in the set  $\{A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K\}$ , and every possible suit-value pair appears in the deck exactly once. Actually, to make the game more interesting, Penn and Teller normally use razor-sharp ninja throwing cards.

<sup>2</sup>Specifically, he hurls them from the opposite side of the stage directly into the back of Penn’s right hand.

<sup>3</sup>The “I don’t know” rule does not apply to extra credit problems. There is no such thing as “partial extra credit”.

# CS 473U: Undergraduate Algorithms, Fall 2006

## Homework 1

Due Tuesday, September 12, 2006 in 3229 Siebel Center

---

Starting with this homework, groups of up to three students can submit or present a single joint solution. If your group is submitting a written solution, please remember to **print the names, NetIDs, and aliases of every group member on every page**. Please remember to submit **separate, individually stapled** solutions to each of the problems.

---

1. Recall from lecture that a *subsequence* of a sequence  $A$  consists of a (not necessarily contiguous) collection of elements of  $A$ , arranged in the same order as they appear in  $A$ . If  $B$  is a subsequence of  $A$ , then  $A$  is a *supersequence* of  $B$ .
  - (a) Describe and analyze a **simple** recursive algorithm to compute, given two sequences  $A$  and  $B$ , the length of the *longest common subsequence* of  $A$  and  $B$ . For example, given the strings ALGORITHM and ALTRUISTIC, your algorithm would return 5, the length of the longest common subsequence ALRIT.
  - (b) Describe and analyze a **simple** recursive algorithm to compute, given two sequences  $A$  and  $B$ , the length of a *shortest common supersequence* of  $A$  and  $B$ . For example, given the strings ALGORITHM and ALTRUISTIC, your algorithm would return 14, the length of the shortest common supersequence ALGTORUISTHIMC.
  - (c) Let  $|A|$  denote the length of sequence  $A$ . For any two sequences  $A$  and  $B$ , let  $\text{lcs}(A, B)$  denote the length of the longest common subsequence of  $A$  and  $B$ , and let  $\text{scs}(A, B)$  denote the length of the shortest common supersequence of  $A$  and  $B$ .  
Prove that  $|A| + |B| = \text{lcs}(A, B) + \text{scs}(A, B)$  for all sequences  $A$  and  $B$ . [Hint: There is a simple non-inductive proof.]

In parts (a) and (b), we are *not* looking for the most efficient algorithms, but for algorithms with simple and correct recursive structure.

2. You are a contestant on a game show, and it is your turn to compete in the following game. You are presented with an  $m \times n$  grid of boxes, each containing a unique number. It costs \$100 to open a box. Your goal is to find a box whose number is larger than its neighbors in the grid (above, below, left, and right). If you spend less money than your opponents, you win a week-long trip for two to Las Vegas and a year's supply of Rice-A-Roni™, to which you are hopelessly addicted.
  - (a) Suppose  $m = 1$ . Describe an algorithm that finds a number that is bigger than any of its neighbors. How many boxes does your algorithm open in the worst case?
  - (b) Suppose  $m = n$ . Describe an algorithm that finds a number that is bigger than any of its neighbors. How many boxes does your algorithm open in the worst case?
  - (c) **[Extra credit]**<sup>1</sup> Prove that your solution to part (b) is asymptotically optimal.

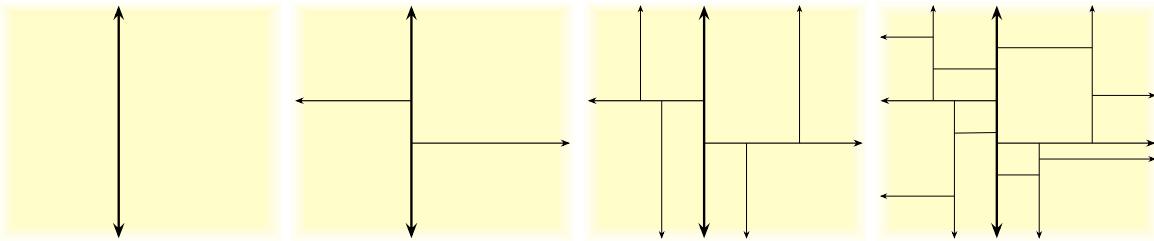
---

<sup>1</sup>The “I don’t know” rule does not apply to extra credit problems. There is no such thing as “partial extra credit”.

3. A kd-tree is a rooted binary tree with three types of nodes: horizontal, vertical, and leaf. Each vertical node has a *left* child and a *right* child; each horizontal node has a *high* child and a *low* child. The non-leaf node types alternate: non-leaf children of vertical nodes are horizontal and vice versa. Each non-leaf node  $v$  stores a real number  $p_v$  called its *pivot value*. Each node  $v$  has an associated *region*  $R(v)$ , defined recursively as follows:

- $R(\text{root})$  is the entire plane.
- If  $v$  is a horizontal node, the horizontal line  $y = p_v$  partitions  $R(v)$  into  $R(\text{high}(v))$  and  $R(\text{low}(v))$  in the obvious way.
- If  $v$  is a vertical node, the vertical line  $x = p_v$  partitions  $R(v)$  into  $R(\text{left}(v))$  and  $R(\text{right}(v))$  in the obvious way.

Thus, each region  $R(v)$  is an axis-aligned rectangle, possibly with one or more sides at infinity. If  $v$  is a leaf, we call  $R(v)$  a *leaf box*.



The first four levels of a typical kd-tree.

Suppose  $T$  is a perfectly balanced kd-tree with  $n$  leaves (and thus with depth exactly  $\lg n$ ).

- Consider the horizontal line  $y = t$ , where  $t \neq p_v$  for all nodes  $v$  in  $T$ . Exactly how many leaf boxes of  $T$  does this line intersect? [Hint: The parity of the root node matters.] Prove your answer is correct. A correct  $\Theta(\cdot)$  bound is worth significant partial credit.
- Describe and analyze an efficient algorithm to compute, given  $T$  and an arbitrary horizontal line  $\ell$ , the number of leaf boxes of  $T$  that lie entirely above  $\ell$ .

# CS 473U: Undergraduate Algorithms, Fall 2006

## Homework 2

Due Tuesday, September 19, 2006 in 3229 Siebel Center

Remember to turn in separate, individually stapled solutions to each of the problems.

---

1. You are given an  $m \times n$  matrix  $M$  in which each entry is a 0 or 1. A *solid block* is a rectangular subset of  $M$  in which each entry is 1. Give a correct efficient algorithm to find a solid block in  $M$  with maximum area.

1	1	0	1	1
0	1	1	1	0
1	1	1	1	1
1	1	0	1	1

An algorithm that runs in  $\Theta(n^c)$  time will earn  $19 - 3c$  points.

2. You are a bus driver with a soda fountain machine in the back and a bus full of very hyper students, who are drinking more soda as they ride along the highway. Your goal is to drop the students off as quickly as possible. More specifically, every minute that a student is on your bus, he drinks another ounce of soda. Your goal is to drop the students off quickly, so that in total they drink as little soda as possible.

You know how many students will get off of the bus at each exit. Your bus begins partway along the highway (probably not at either end), and moves at a constant rate. You must drive the bus along the highway- however you may drive forward to one exit then backward to an exit in the other direction, switching as often as you like (you can stop the bus, drop off students, and turn around instantaneously).

Give an efficient algorithm to drop the students off so that they drink as little soda as possible. The input to the algorithm should be: the bus route (a list of the exits, together with the travel time between successive exits), the number of students you will drop off at each exit, and the current location of your bus (you may assume it is at an exit).

3. Suppose we want to display a paragraph of text on a computer screen. The text consists of  $n$  words, where the  $i$ th word is  $p_i$  pixels wide. We want to break the paragraph into several lines, each exactly  $P$  pixels long. Depending on which words we put on each line, we will need to insert different amounts of white space between the words. The paragraph should be fully justified, meaning that the first word on each line starts at its leftmost pixel, and *except for the last line*, the last character on each line ends at its rightmost pixel. There must be at least one pixel of whitespace between any two words on the same line.

Define the *slop* of a paragraph layout as the sum over all lines, *except the last*, of the cube of the number of extra white-space pixels in each line (not counting the one pixel required between every adjacent pair of words). Specifically, if a line contains words  $i$  through  $j$ , then the amount of extra white space on that line is  $P - j + i - \sum_{k=i}^j P_k$ . Describe a dynamic programming algorithm to print the paragraph with minimum slop.

# CS 473U: Undergraduate Algorithms, Fall 2006

## Homework 3

Due Wednesday, October 4, 2006 in 3229 Siebel Center

---

Remember to turn in separate, individually stapled solutions to each of the problems.

---

1. Consider a perfect tree of height  $h$ , where every non-leaf node has 3 children. (Therefore, each of the  $3^h$  leaves is at distance  $h$  from the root.) Every leaf has a boolean value associated with it - either 0 or 1. Every internal node gets the boolean value assigned to the majority of its children. Given the values assigned to the leaves, we want to find an algorithm that computes the value (0 or 1) of the root.

It is not hard to find a (deterministic) algorithm that looks at every leaf and correctly determines the value of the root, but this takes  $O(3^h)$  time. Describe and analyze a *randomized* algorithm that, on average, looks at asymptotically fewer leaves. That is, the expected number of leaves your algorithm examines should be  $o(3^h)$ .

2. We define a *meldable heap* to be a binary tree of elements, each of which has a priority, such that the priority of any node is less than the priority of its parent. (Note that the heap does **not** have to be balanced, and that the element with greatest priority is the root.) We also define the priority of a heap to be the priority of its root.

The *meld* operation takes as input two (meldable) heaps and returns a single meldable heap  $H$  that contains all the elements of both input heaps. We define *meld* as follows:

- Let  $H_1$  be the input heap with greater priority, and  $H_2$  the input heap with lower priority. (That is, the priority of  $\text{root}(H_1)$  is greater than the priority of  $\text{root}(H_2)$ .) Let  $H_L$  be the left subtree of  $\text{root}(H_1)$  and  $H_R$  be the right subtree of  $\text{root}(H_1)$ .
  - We set  $\text{root}(H) = \text{root}(H_1)$ .
  - We now flip a coin that comes up either “Left” or “Right” with equal probability.
    - If it comes up “Left”, we set the left subtree of  $\text{root}(H)$  to be  $H_L$ , and the right subtree of  $\text{root}(H)$  to be  $\text{meld}(H_R, H_2)$  (defined recursively).
    - If the coin comes up “Right”, we set the right subtree of  $\text{root}(H)$  to be  $H_R$ , and the left subtree of  $\text{root}(H)$  to be  $\text{meld}(H_L, H_2)$ .
  - As a base case, melding any heap  $H_1$  with an empty heap gives  $H_1$ .
- (a) Analyze the expected running time of  $\text{meld}(H_a, H_b)$  if  $H_a$  is a (meldable) heap with  $n$  elements, and  $H_b$  is a (meldable) heap with  $m$  elements.
- (b) Describe how to perform each of the following operations using only melds, and give the running time of each.
- $\text{DeleteMax}(H)$ , which deletes the element with greatest priority.
  - $\text{Insert}(H, x)$ , which inserts the element  $x$  into the heap  $H$ .
  - $\text{Delete}(H, x)$ , which - given a pointer to element  $x$  in heap  $H$  - returns the heap with  $x$  deleted.

3. Randomized Selection. Given an (unsorted) array of  $n$  distinct elements and an integer  $k$ , SELECTION is the problem of finding the  $k$ th smallest element in the array. One easy solution is to sort the array in increasing order, and then look up the  $k$ th entry, but this takes  $\Theta(n \log n)$  time. The randomized algorithm below attempts to do better, at least on average.

```

QuickSelect(Array A, n, k)
pivot ← Random(1, n)
S ← { $x \mid x \in A, x < A[pivot]$ }
s ← |S|
L ← { $x \mid x \in A, x > A[pivot]$ }
if ( $k = s + 1$ )
    return  $A[pivot]$ 
else if ( $k \leq s$ )
    return QuickSelect( $S, s, k$ )
else
    return QuickSelect( $L, n - (s + 1), k - (s + 1)$ )

```

Here we assume that  $\text{Random}(a, b)$  returns an integer chosen uniformly at random from  $a$  to  $b$  (inclusive of  $a$  and  $b$ ). The pivot position is randomly chosen;  $S$  is the set of elements smaller than the pivot element, and  $L$  the set of elements larger than the pivot. The sets  $S$  and  $L$  are found by comparing every other element of  $A$  to the pivot. We partition the elements into these two ‘halves’, and recurse on the appropriate half.

- (a) Write a recurrence relation for the expected running time of QuickSelect.
  - (b) Given any two elements  $x, y \in A$ , what is the probability that  $x$  and  $y$  will be compared?
  - (c) Either from part (a) or part (b), find the expected running time of QuickSelect.
4. [Extra Credit]: In the previous problem, we found a  $\Theta(n)$  algorithm for selecting the  $k$ th smallest element, but the constant hidden in the  $\Theta(\cdot)$  notation is somewhat large. It is easy to find the *smallest* element using at most  $n$  comparisons; we would like to be able to extend this to larger  $k$ . Can you find a randomized algorithm that uses  $n + \Theta(k \log k \log n)$ <sup>1</sup> expected comparisons? (Note that there is no constant multiplying the  $n$ .)

*Hint:* While scanning through a random permutation of  $n$  elements, how many times does the smallest element seen so far change? (See HBS 0.) How many times does the  $k$ th smallest element so far change?

---

<sup>1</sup>There is an algorithm that uses  $n + \Theta(k \log(n/k))$  comparisons, but this is even harder.

# CS 473U: Undergraduate Algorithms, Fall 2006

## Homework 4

Due Tuesday, October 10, 2006 in 3229 Siebel Center

---

Remember to submit **separate, individually stapled** solutions to each of the problems.

---

1. Chicago has many tall buildings, but only some of them have a clear view of Lake Michigan. Suppose we are given an array  $A[1..n]$  that stores the height of  $n$  buildings on a city block, indexed from west to east. Building  $i$  has a good view of Lake Michigan if every building to the east of  $i$  is shorter than  $i$ . We present an algorithm that computes which buildings have a good view of Lake Michigan. Use the taxation method of amortized analysis to bound the amortized time spent in each iteration of the for loop. What is the total runtime?

```
GOODVIEW( $A[1..n]$ ):  
    Initialize a stack  $S$   
    for  $i = 1$  to  $n$   
        while ( $S$  not empty and  $A[i] \geq A[S.\text{top}]$ )  
            POP( $S$ )  
        PUSH( $S, i$ )  
    return  $S$ 
```

2. Design and analyze a simple data structure that maintains a list of integers and supports the following operations.
  - (a)  $\text{CREATE}()$ : creates and returns a new list  $L$
  - (b)  $\text{PUSH}(L, x)$ : appends  $x$  to the end of  $L$
  - (c)  $\text{POP}(L)$ : deletes the last entry of  $L$  and returns it
  - (d)  $\text{LOOKUP}(L, k)$ : returns the  $k$ th entry of  $L$

Your solution may use these primitive data structures: arrays, balanced binary search trees, heaps, queues, single or doubly linked lists, and stacks. If your algorithm uses anything fancier, you must give an explicit implementation. Your data structure should support all operations in amortized constant time. In addition, your data structure should support  $\text{LOOKUP}()$  in worst-case  $O(1)$  time. At all times, your data structure should use space which is linear in the number of objects it stores.

3. Consider a computer game in which players must navigate through a field of landmines, which are represented as points in the plane. The computer creates new landmines which the players must avoid. A player may ask the computer how many landmines are contained in any simple polygonal region; it is your job to design an algorithm which answers these questions efficiently.

You have access to an efficient static data structure which supports the following operations.

- $\text{CREATE}_S(\{p_1, p_2, \dots, p_n\})$ : creates a new data structure  $S$  containing the points  $\{p_1, \dots, p_n\}$ . It has a worst-case running time of  $T(n)$ . Assume that  $T(n)/n \geq T(n-1)/(n-1)$ , so that the average processing time of elements does not decrease as  $n$  grows.
- $\text{DUMP}_S(S)$ : destroys  $S$  and returns the set of points that  $S$  stored. It has a worst-case running time of  $O(n)$ , where  $n$  is the number of points in  $S$ .
- $\text{QUERY}_S(S, R)$ : returns the number of points in  $S$  that are contained in the region  $R$ . It has a worst-case running time of  $Q(n)$ , where  $n$  is the number of points stored in  $S$ .

Unfortunately, the data structure does not support point insertion, which is required in your application. Using the given static data structure, design and analyze a dynamic data structure that supports the following operations.

- (a)  $\text{CREATED}()$ : creates a new data structure  $D$  containing no points. It should have a worst-case constant running time.
- (b)  $\text{INSERT}_D(D, p)$ : inserts  $p$  into  $D$ . It should run in amortized  $O(\log n) \cdot T(n)/n$  time.
- (c)  $\text{QUERY}_D(D, R)$ : returns the number of points in  $D$  that are contained in the region  $R$ . It should have a worst-case running time of  $O(\log n) \cdot Q(n)$ .

# CS 473U: Undergraduate Algorithms, Fall 2006

## Homework 5

Due Tuesday, October 24, 2006 in 3229 Siebel Center

Remember to turn in separate, individually stapled solutions to each of the problems.

---

### 1. Makefiles:

In order to facilitate recompiling programs from multiple source files when only a small number of files have been updated, there is a UNIX utility called ‘make’ that only recompiles those files that were changed after the most recent compilation, *and* any intermediate files in the compilation that depend on those that were changed. A Makefile is typically composed of a list of source files that must be compiled. Each of these source files is dependent on some of the other files that must be compiled. Thus a source file must be recompiled if a file on which it depends is changed.

Assuming you have a list of which files have been recently changed, as well as a list for each source file of the files on which it depends, design and analyze an efficient algorithm to recompile only the necessary files. DO NOT worry about the details of parsing a Makefile.

### 2. Consider a graph $G$ , with $n$ vertices. Show that if any two of the following properties hold for $G$ , then the third property must also hold.

- $G$  is connected.
- $G$  is acyclic.
- $G$  has  $n - 1$  edges.

### 3. The weight of a spanning tree is the sum of the weights on the edges of the tree. Given a graph, $G$ , describe an efficient algorithm (the most efficient one you can) to find the $k$ lightest (with least weight) spanning trees of $G$ .

Analyze the running time of your algorithm. Be sure to prove your algorithm is correct.

# CS 473U: Undergraduate Algorithms, Fall 2006

## Homework 6

Due Wednesday, November 8, 2006 in 3229 Siebel Center

---

Remember to turn in separate, individually stapled solutions to each of the problems.

---

1. Dijkstra's algorithm can be used to determine shortest paths on graphs with some negative edge weights (as long as there are no negative cycles), but the worst-case running time is much worse than the  $O(E + V \log V)$  it takes when the edge weights are all positive. Construct an infinite family of graphs - with negative edge weights - for which the asymptotic running time of Dijkstra's algorithm is  $\Omega(2^{|V|})$ .
2. It's a cold and rainy night, and you have to get home from Siebel Center. Your car has broken down, and it's too windy to walk, which means you have to take a bus. To make matters worse, there is no bus that goes directly from Siebel Center to your apartment, so you have to change buses some number of times on your way home. Since it's cold outside, you want to spend as little time as possible waiting in bus shelters.

From a computer in Siebel Center, you can access an online copy of the MTD bus schedule, which lists bus routes and the arrival time of every bus at each stop on its route. Describe an algorithm which, given the schedule, finds a way for you to get home that minimizes the time you spend at bus shelters (the amount of time you spend on the bus doesn't matter). Since Siebel Center is warm and the nearest bus stop is right outside, you can assume that you wait inside Siebel until the first bus you want to take arrives outside. Analyze the efficiency of your algorithm and prove that it is correct.

3. The Floyd-Warshall all-pairs shortest path algorithm computes, for each  $u, v \in V$ , the shortest path from  $u$  to  $v$ . However, if the graph has negative cycles, the algorithm fails. Describe a modified version of the algorithm (*with the same asymptotic time complexity*) that correctly returns shortest-path distances, even if the graph contains negative cycles. That is, if there is a path from  $u$  to some negative cycle, and a path from that cycle to  $v$ , the algorithm should output  $dist(u, v) = -\infty$ . For any other pair  $u, v$ , the algorithm should output the length of the shortest directed path from  $u$  to  $v$ .

# CS 473U: Undergraduate Algorithms, Fall 2006

## Homework 6

Due at **4 p.m.** on Friday, November 17, 2006 in 3229 Siebel Center

---

Remember to turn in separate, individually stapled solutions to each of the problems.

---

1. Given an undirected graph  $G(V, E)$ , with three vertices  $u, v, w \in V$ , you want to know whether there exists a path from  $u$  to  $w$  via  $v$ . (That is, the path from  $u$  to  $w$  must use  $v$  as an intermediate vertex.) Describe an efficient algorithm to solve this problem.
  
2. *Ad-hoc Networks*, made up of cheap, low-powered wireless devices, are often used on battle-fields, in regions that have recently suffered from natural disasters, and in other situations where people might want to monitor conditions in hard-to-reach areas. The idea is that a large collection of the wireless devices could be dropped into the area from an airplane (for instance), and then they could be configured into an efficiently functioning network.

Since the devices are cheap and low-powered, they frequently fail, and we would like our networks to be reliable. If a device detects that it is likely to fail, it should transmit the information it has to some other device (called a *backup*) within range of it. The range is limited; we assume that there is a distance  $d$  such that two devices can communicate if and only if they are within distance  $d$  of each other. To improve reliability, we don't want a device to transmit information to a neighbor that has already failed, and so we require each device  $v$  to have at least  $k$  backup devices that it could potentially contact, all of which must be within  $d$  meters of it. We call this the *backup set* of  $v$ . Also, we do not want any device to be in the backup set of too many other devices; if it were, and it failed, a large fraction of our network would be affected.

The input to our problem is a collection of  $n$  devices, and for each pair  $u, v$  of devices, the distance between  $u$  and  $v$ . We are also given the distance  $d$  that determines the range of a device, and parameters  $b$  and  $k$ . Describe an algorithm that determines if, for each device, we can find a backup set of size  $k$ , while also requiring that no device appears in the backup set of more than  $b$  other devices.

3. **UPDATED:** Given a piece of text  $T$  and a pattern  $P$  (the ‘search string’), an algorithm for the string-matching problem either finds the first occurrence of  $P$  in  $T$ , or reports that there is none. Modify the Knuth-Morris-Pratt (KMP) algorithm so that it solves the string-matching problem, even if the pattern contains the wildcards ‘?’ and ‘\*’. Here, ‘?’ represents any *single* character of the text, and ‘\*’ represents any substring of the text (including the empty substring). For example, the pattern “A?B\*?A” matches the text “ABACBCABCACBA” starting in position 3 (in three different ways), and position 7 (in two ways). For this input, your algorithm would need to return ‘3’.

**UPDATE:** You may assume that the pattern you are trying to match containst at most 3 blocks of question marks; the usage of ‘\*’ wildcards is stll unrestricted. Here, a block refers to a string of consecutive ‘?’s in the pattern. For example, AAB??ACA??????BB contains 2 blocks of question marks; A?B?C?A?C contains 4 blocks of question marks.

4. In the two-dimensional pattern-matching problem, you are given an  $m \times n$  matrix  $M$  and a  $p \times q$  pattern  $P$ . You wish to find all positions  $(i, j)$  in  $M$  such that the the submatrix of  $M$  between rows  $i$  and  $i + p - 1$  and between columns  $j$  and  $j + q - 1$  is identical to  $P$ . (That is, the  $p \times q$  sub-matrix of  $M$  below and to the right of position  $(i, j)$  should be identical to  $P$ .) Describe and analyze an efficient algorithm to solve this problem.<sup>1</sup>

---

<sup>1</sup>Note that the normal string-matching problem is the special case of the 2-dimensional problem where  $m = p = 1$ .

# CS 473U: Undergraduate Algorithms, Fall 2006

## Homework 8

Due Wednesday, December 6, 2006 in 3229 Siebel Center

---

Remember to submit **separate, individually stapled** solutions to each of the problems.

---

1. Given an array  $A[1..n]$  of  $n \geq 2$  distinct integers, we wish to find the second largest element using as few comparisons as possible.
  - (a) Give an algorithm which finds the second largest element and uses at most  $n + \lceil \lg n \rceil - 2$  comparisons in the worst case.
  - \*(b) Prove that every algorithm which finds the second largest element uses at least  $n + \lceil \lg n \rceil - 2$  comparisons in the worst case.
2. Let  $R$  be a set of rectangles in the plane. For each point  $p$  in the plane, we say that the *rectangle depth* of  $p$  is the number of rectangles in  $R$  that contain  $p$ .
  - (a) (Step 1: Algorithm Design) Design and analyze a polynomial-time algorithm which, given  $R$ , computes the maximum rectangle depth.
  - (b) (Step 2: ???) Describe and analyze a polynomial-time reduction from the maximum rectangle depth problem to the maximum clique problem.
  - (c) (Step 3: Profit!) In 2000, the Clay Mathematics Institute described the Millennium Problems: seven challenging open problems which are central to ongoing mathematical research. The Clay Institute established seven prizes, each worth one million dollars, to be awarded to anyone who solves a Millennium problem. One of these problems is the  $P = NP$  question. In (a), we developed a polynomial-time algorithm for the maximum rectangle depth problem. In (b), we found a reduction from this problem to an NP-complete problem. We know from class that if we find a polynomial-time algorithm for any NP-complete problem, then we have shown  $P = NP$ . Why hasn't Jeff used (a) and (b) to show  $P = NP$  and become a millionaire?
3. Let  $G$  be a complete graph with integer edge weights. If  $C$  is a cycle in  $G$ , we say that the *cost* of  $C$  is the sum of the weights of edges in  $C$ . Given  $G$ , the traveling salesman problem (TSP) asks us to compute a Hamiltonian cycle of minimum cost. Given  $G$ , the traveling salesman cost problem (TSCP) asks us to compute the cost of a minimum cost Hamiltonian cycle. Given  $G$  and an integer  $k$ , the traveling salesman decision problem (TSDP) asks us to decide if there is a Hamiltonian cycle in  $G$  of cost at most  $k$ .
  - (a) Describe and analyze a polynomial-time reduction from TSP to TSCP.
  - (b) Describe and analyze a polynomial-time reduction from TSCP to TSDP.
  - (c) Describe and analyze a polynomial-time reduction from TSDP to TSP.

- (d) What can you conclude about the relative computational difficulty of TSP, TSCP, and TSDP?
4. Let  $G$  be a graph. A set  $S$  of vertices of  $G$  is a *dominating set* if every vertex in  $G$  is either in  $S$  or adjacent to a vertex in  $S$ . Show that, given  $G$  and an integer  $k$ , deciding if  $G$  contains a dominating set of size at most  $k$  is NP-complete.

1. Probability

- (a)  $n$  people have checked their hats with a hat clerk. The clerk is somewhat absent-minded and returns the hats uniformly at random (with no regard for whether each hat is returned to its owner). On average, how many people will get back their own hats?
  - (b) Let  $S$  be a uniformly random permutation of  $\{1, 2, \dots, n-1, n\}$ . As we move from the left to the right of the permutation, let  $X$  denote the smallest number seen so far. On average, how many different values will  $X$  take?
2. A *tournament* is a directed graph where each pair of distinct vertices  $u, v$  has either the edge  $uv$  or the edge  $vu$  (but not both). A *Hamiltonian path* is a (directed) path that visits each vertex of the (di)graph. Prove that every tournament has a Hamiltonian path.
3. Describe and analyze a data structure that stores a set of  $n$  records, each with a numerical *key*, such that the following operation can be performed quickly:

**Foo( $a$ ):** return the sum of the records with keys at least as large as  $a$ .

For example, if the keys are:

3 4 9 6 5 8 7 1 0

then **Foo(2)** would return 42, since 3, 4, 5, 6, 7, 8, 9 are all larger than 2 and  $3 + 4 + 5 + 6 + 7 + 8 + 9 = 42$ . You may assume that no two records have equal keys, and that no record has a key equal to  $a$ . Analyze both the size of your data structure and the running time of your **Foo** algorithm. Your data structure must be as small as possible and your **Foo** algorithm must be as fast as possible.

1. The Acme Company is planning a company party. In planning the party, each employee is assigned a *fun value* (a positive real number). The goal of the party planners is to maximize the total fun value (sum of the individual fun values) of the employees invited to the party. However, the planners are not allowed to invite both an employee and his direct boss. Given a tree containing the boss/underling structure of Acme, find the invitation list with the highest allowable fun value.
  2. An *inversion* in an array  $A$  is a pair  $i, j$  such that  $i < j$  and  $A[i] > A[j]$ . (In an  $n$ -element array, the number of inversions is between 0 and  $\binom{n}{2}$ .)  
Find an efficient algorithm to count the number of inversions in an  $n$ -element array.
  3. A *tromino* is a geometric shape made from three squares joined along complete edges. There are only two possible trominoes: the three component squares may be joined in a line or an L-shape.
    - (a) Show that it is possible to cover all but one square of a  $64 \times 64$  checkerboard using L-shape trominoes. (In your covering, each tromino should cover three squares and no square should be covered more than once.)
    - (b) Show that you can leave *any* single square uncovered.
    - (c) Can you cover all but one square of a  $64 \times 64$  checkerboard using *line* trominoes? If so, which squares can you leave uncovered?

**1. Moving on a Checkerboard**

Suppose that you are given an  $n \times n$  checkerboard and a checker. You must move the checker from the bottom edge of the board to the top edge of the board according to the following rule. At each step you may move the checker to one of three squares:

- 1) the square immediately above
- 2) the square that is one up and one to the left (but only if the checker is not already in the leftmost column)
- 3) the square that is one up and one to the right (but only if the checker is not already in the rightmost column)

Each time you move from square  $x$  to square  $y$ , you receive  $p(x, y)$  dollars. You are given a list of the values  $p(x, y)$  for each pair  $(x, y)$  for which a move from  $x$  to  $y$  is legal. Do not assume that  $p(x, y)$  is positive.

Give an algorithm that figures out the set of moves that will move the checker from somewhere along the bottom edge to somewhere along the top edge while gathering as many dollars as possible. You algorithm is free to pick any square along the bottom edge as a starting point and any square along the top edge as a destination in order to maximize the number of dollars gathered along the way. What is the running time of your algorithm?

**2. Maximizing Profit**

You are given lists of values  $h_1, h_2, \dots, h_k$  and  $l_1, l_2, \dots, l_k$ . For each  $i$  you can choose  $j_i = h_i$ ,  $j_i = l_i$ , or  $j_i = 0$ ; the only catch is that if  $j_i = h_i$  then  $j_{i-1}$  must be 0 (except for  $i = 1$ ). Your goal is to maximize  $\sum_{i=1}^k j_i$ .

Give an efficient algorithm that returns the maximum possible value of  $\sum_{i=1}^k j_i$ .

**3. Maximum alternating subsequence**

An *alternating sequence* is a sequence  $a_1, a_2, \dots$  such that no three consecutive terms of the sequence satisfy  $a_i > a_{i+1} > a_{i+2}$  or  $a_i < a_{i+1} < a_{i+2}$ .

Given a sequence, efficiently find the longest alternating subsequence it contains. What is the running time of your algorithm?

### 1. Championship Showdown

What excitement! The Champaign Spinners and the Urbana Dreamweavers have advanced to meet each other in the World Series of Basketweaving! The World Champions will be decided by a best of  $2n - 1$  series of head-to-head weaving matches, and the first to win  $n$  matches will take home the coveted Golden Basket (for example, a best-of-7 series requires four match wins, but we will keep the generalized case). We know that for any given match there is a constant probability  $p$  that Champaign will win, and a subsequent probability  $q = 1 - p$  that Urbana will win.

Let  $P(i, j)$  be the probability that Champaign will win the series given that they still need  $i$  more victories, whereas Urbana needs  $j$  more victories for the championship.  $P(0, j) = 1$ ,  $1 \leq j \leq n$ , because Champaign needs no more victories to win.  $P(i, 0) = 0$ ,  $1 \leq i \leq n$ , as Champaign cannot possibly win if Urbana already has.  $P(0, 0)$  is meaningless. Champaign wins any particular match with probability  $p$  and loses with probability  $q$ , so

$$P(i, j) = p \cdot P(i - 1, j) + q \cdot P(i, j - 1)$$

for any  $i \geq 1$  and  $j \geq 1$ .

Create and analyze an  $O(n^2)$ -time dynamic programming algorithm that takes the parameters  $n, p$ , and  $q$  and returns the probability that Champaign will win the series (that is, calculate  $P(n, n)$ ).

### 2. Making Change

Suppose you are a simple shopkeeper living in a country with  $n$  different types of coins, with values  $1 = c[1] < c[2] < \dots < c[n]$ . (In the U.S., for example,  $n = 6$  and the values are 1, 5, 10, 25, 50, and 100 cents.) Your beloved benevolent dictator, El Generalissimo, has decreed that whenever you give a customer change, you must use the smallest possible number of coins, so as not to wear out the image of El Generalissimo lovingly engraved on each coin by servants of the Royal Treasury.

Describe and analyze a dynamic programming algorithm to determine, given a target amount  $A$  and a sorted array  $c[1..n]$  of coin values, the smallest number of coins needed to make  $A$  cents in change. You can assume that  $c[1] = 1$ , so that it is possible to make change for any amount  $A$ .

### 3. Knapsack

You are a thief, who is trying to choose the best collection of treasure (some subset of the  $n$  treasures, numbered 1 through  $n$ ) to steal. The weight of item  $i$  is  $w_i \in \mathbb{N}$  and the profit is  $p_i \in \mathbb{R}$ . Let  $C \in \mathbb{N}$  be the maximum weight that your knapsack can hold. Your goal is to choose a subset of elements  $S \subseteq \{1, 2, \dots, n\}$  that maximizes your total profit  $P(S) = \sum_{i \in S} p_i$ , subject to the constraint that the sum of the weights  $W(S) = \sum_{i \in S} w_i$  is not more than  $C$ .

Give an algorithm that runs in time  $O(Cn)$ .

**1. Randomized Edge Cuts**

We will randomly partition the vertex set of a graph  $G$  into two sets  $S$  and  $T$ . The algorithm is to flip a coin for each vertex and with probability  $1/2$ , put it in  $S$ ; otherwise put it in  $T$ .

- (a) Show that the expected number of edges with one endpoint in  $S$  and the other endpoint in  $T$  is exactly half the edges in  $G$ .
- (b) Now say the edges have weights. What can you say about the sum of the weights of the edges with one endpoint in  $S$  and the other endpoint in  $T$ ?

**2. Skip Lists**

A *skip list* is built in layers. The bottom layer is an ordinary sorted linked list. Each higher layer acts as an “express lane” for the lists below, where an element in layer  $i$  appears in layer  $i+1$  with some fixed probability  $p$ .

```

1
1-----4---6
1---3-4---6-----9
1-2-3-4-5-6-7-8-9-10

```

- (a) What is the probability a node reaches height  $h$ .
- (b) What is the probability any node is above  $c \log n$  (for some fixed value of  $c$ )?  
Compute the value explicitly when  $p = 1/2$  and  $c = 4$ .
- (c) To search for an entry  $x$ , scan the top layer until you find the last entry  $y$  that is less than or equal to  $x$ . If  $y < x$ , drop down one layer and in this new layer (beginning at  $y$ ) find the last entry that is less than or equal to  $x$ . Repeat this process (dropping down a layer, then finding the last entry less than or equal to  $x$ ) until you either find  $x$  or reach the bottom layer and confirm that  $x$  is not in the skip list. What is the expected search time?
- (d) Describe an efficient method for insertion. What is the expected insertion time?

**3. Clock Solitaire**

In a standard deck of 52 cards, put 4 face-down in each of the 12 ‘hour’ positions around a clock, and 4 face-down in a pile in the center. Turn up a card from the center, and look at the number on it. If it’s number  $x$ , place the card face-up next to the face-down pile for  $x$ , and turn up the next card in the face-down pile for  $x$  (that is, the face-down pile corresponding to hour  $x$ ). You win if, for each  $\text{Ace} \leq x \leq \text{Queen}$ , all four cards of value  $x$  are turned face-up before all four Kings (the center cards) are turned face-up.

What is the probability that you win a game of Clock Solitaire?

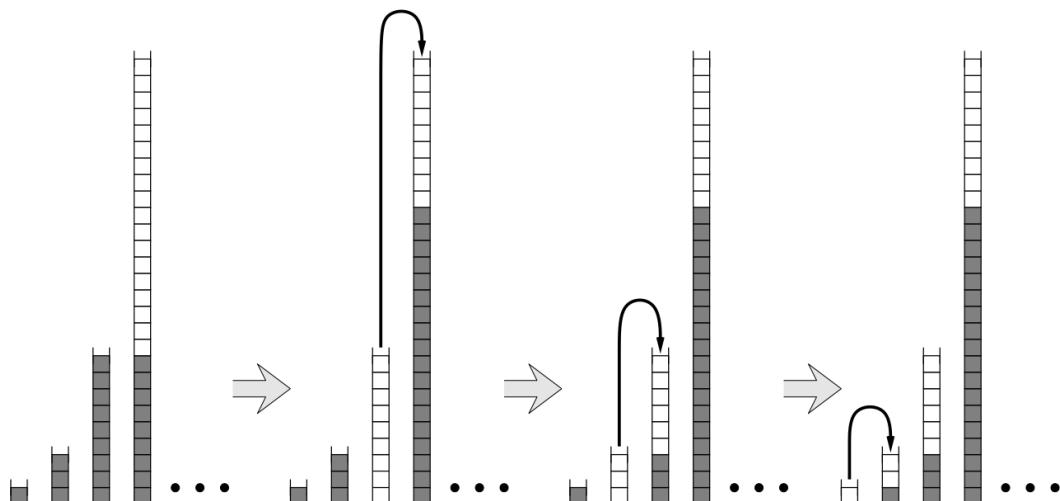
### 1. Simulating Queues with Stacks

A *queue* is a first-in-first-out data structure. It supports two operations *push* and *pop*. Push adds a new item to the back of the queue, while pop removes the first item from the front of the queue. A *stack* is a last-in-first-out data structure. It also supports push and pop. As with a queue, push adds a new item to the back of the queue. However, pop removes the last item from the back of the queue (the one most recently added).

Show how you can simulate a queue by using two stacks. Any sequence of pushes and pops should run in amortized constant time.

### 2. Multistacks

A *multistack* consists of an infinite series of stacks  $S_0, S_1, S_2, \dots$ , where the  $i$ th stack  $S_i$  can hold up to  $3^i$  elements. Whenever a user attempts to push an element onto any full stack  $S_i$ , we first move all the elements in  $S_i$  to stack  $S_{i+1}$  to make room. But if  $S_{i+1}$  is already full, we first move all its members to  $S_{i+2}$ , and so on. To clarify, a user can only push elements onto  $S_0$ . All other pushes and pops happen in order to make space to push onto  $S_0$ . Moving a single element from one stack to the next takes  $O(1)$  time.



**Figure 1.** Making room for one new element in a multistack.

- (a) In the worst case, how long does it take to push one more element onto a multistack containing  $n$  elements?
- (b) Prove that the amortized cost of a push operation is  $O(\log n)$ , where  $n$  is the maximum number of elements in the multistack.

### 3. Powerhungry function costs

A sequence of  $n$  operations is performed on a data structure. The  $i$ th operation costs  $i$  if  $i$  is an exact power of 2, and 1 otherwise. Determine the amortized cost of the operation.

## 1. Representation of Integers

- (a) Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers—if  $F_n$  appears in the sum, then neither  $F_{n+1}$  nor  $F_{n-1}$  will. For example:  $42 = F_9 + F_6$ ,  $25 = F_8 + F_4 + F_2$ ,  $17 = F_7 + F_4 + F_2$ .
- (b) Prove that any integer (positive, negative, or zero) can be written in the form  $\sum_i \pm 3^i$ , where the exponents  $i$  are distinct non-negative integers. For example  $42 = 3^4 - 3^3 - 3^2 - 3^1$ ,  $25 = 3^3 - 3^1 + 3^0$ ,  $17 = 3^3 - 3^2 - 3^0$ .

## 2. Minimal Dominating Set

Suppose you are given a rooted tree  $T$  (not necessarily binary). You want to label each node in  $T$  with an integer 0 or 1, such that every node either has the label 1 or is adjacent to a node with the label 1 (or both). The *cost* of a labeling is the number of nodes with label 1. Describe and analyze an algorithm to compute the minimum cost of any labeling of the given tree  $T$ .

## 3. Names in Boxes

The names of 100 prisoners are placed in 100 wooden boxes, one name to a box, and the boxes are lined up on a table in a room. One by one, the prisoners are led into the room; each may look in at most 50 boxes, but must leave the room exactly as he found it and is permitted no further communication with the others.

The prisoners have a chance to plot their strategy in advance, and they are going to need it, because unless *every single prisoner finds his own name* all will subsequently be executed. Find a strategy for them which has probability of success exceeding 30%. You may assume that the names are distributed in the boxes uniformly at random.

- (a) Calculate the probability of success if each prisoner picks 50 boxes uniformly at random.
- \*(b) Consider the following strategy.

The prisoners number themselves 1 to 100. Prisoner  $i$  begins by looking in box  $i$ . There he finds the name of prisoner  $j$ . If  $j \neq i$ , he continues by looking in box  $j$ . As long as prisoner  $i$  has not found his name, he continues by looking in the box corresponding to the last name he found.

Describe the set of permutations of names in boxes for which this strategy will succeed.

- \*(c) Count the number of permutations for which the strategy above succeeds. Use this sum to calculate the probability of success. You may find it useful to do this calculation for general  $n$ , then set  $n = 100$  at the end.
- (d) We assumed that the names were distributed in the boxes uniformly at random. Explain how the prisoners could augment their strategy to make this assumption unnecessary.

### 1. Dynamic MSTs

Suppose that you already have a minimum spanning tree (MST) in a graph. Now one of the edge weights changes. Give an efficient algorithm to find an MST in the new graph.

### 2. Minimum Bottleneck Trees

In a graph  $G$ , for any pair of vertices  $u, v$ , let  $\text{bottleneck}(u, v)$  be the maximum over all paths  $p_i$  from  $u$  to  $v$  of the minimum-weight edge along  $p_i$ . Construct a spanning tree  $T$  of  $G$  such that for each pair of vertices, their bottleneck in  $G$  is the same as their bottleneck in  $T$ .

One way to think about it is to imagine the vertices of the graph as islands, and the edges as bridges. Each bridge has a maximum weight it can support. If a truck is carrying stuff from  $u$  to  $v$ , how much can the truck carry? We don't care what route the truck takes; the point is that the smallest-weight edge on the route will determine the load.

### 3. Eulerian Tours

An *Eulerian tour* is a “walk along edges of a graph” (in which successive edges must have a common endpoint) that uses each edge exactly once and ends at the vertex where it starts. A graph is called Eulerian if it has an Eulerian tour.

Prove that a connected graph is Eulerian iff each vertex has even degree.

### 1. Alien Abduction

Mulder and Scully have computed, for every road in the United States, the exact probability that someone driving on that road won't be abducted by aliens. Agent Mulder needs to drive from Langley, Virginia to Area 51, Nevada. What route should he take so that he has the least chance of being abducted?

More formally, you are given a directed graph  $G = (V, E)$ , where every edge  $e$  has an independent safety probability  $p(e)$ . The *safety* of a path is the product of the safety probabilities of its edges. Design and analyze an algorithm to determine the safest path from a given start vertex  $s$  to a given target vertex  $t$ .

### 2. The Only SSSP Algorithm

In the lecture notes, Jeff mentions that all SSSP algorithms are special cases of the following generic SSSP algorithm. Each vertex  $v$  in the graph stores two values, which describe a tentative shortest path from  $s$  to  $v$ .

- $\text{dist}(v)$  is the length of the tentative shortest  $s \rightsquigarrow v$  path.
- $\text{pred}(v)$  is the predecessor of  $v$  in the shortest  $s \rightsquigarrow v$  path.

We call an edge *tense* if  $\text{dist}(u) + w(u \rightarrow v) < \text{dist}(v)$ . Our generic algorithm repeatedly finds a tense edge in the graph and *relaxes* it:

$$\boxed{\begin{array}{c} \text{Relax}(u \rightarrow v): \\ \hline \text{dist}(v) \leftarrow \text{dist}(u) + w(u \rightarrow v) \\ \text{pred}(v) \leftarrow u \end{array}}$$

If there are no tense edges, our algorithm is finished, and we have our desired shortest path tree. The correctness of the relaxation algorithm follows directly from three simple claims. The first of these is below. Prove it.

- When the algorithm halts, if  $\text{dist}(v) \neq \infty$ , then  $\text{dist}(v)$  is the total weight of the predecessor chain ending at  $v$ :

$$s \rightarrow \dots \rightarrow (\text{pred}(\text{pred}(v))) \rightarrow \text{pred}(v) \rightarrow v.$$

### 3. Can't find a Cut-edge

A cut-edge is an edge which when deleted disconnects the graph. Prove or disprove the following. Every 3-regular graph has no cut-edge. (A common approach is induction.)

### 1. Max-Flow with vertex capacities

In a standard  $s - t$  Maximum-Flow Problem, we assume edges have capacities, and there is no limit on how much flow is allowed to pass through a node. In this problem, we consider the variant of Maximum-Flow and Minimum-Cut problems with node capacities.

More specifically, each node,  $n_i$ , has a capacity  $c_i$ . The edges have unlimited capacity. Show how you can model this problem as a standard Max-flow problem (where the weights are on the edges).

### 2. Emergency evacuation

Due to large-scale flooding in a region, paramedics have identified a set of  $n$  injured people distributed across the region who need to be reashed to hospitals. There are  $k$  hospitals in the region, and each of the  $n$  people needs to be brought to a hospital that is within a half-hour's driving time of their current location.

At the same time, we don't want to overload any hospital by sending too many patients its way. We'd like to distribute the people so that each hospital receives at most  $\lceil n/k \rceil$  people.

Show how to model this problem as a Max-flow problem.

### 3. Tracking a Hacker

A computer network (with each edge weight 1) is designed to carry traffic from a source  $s$  to a destination  $t$ . Recently, a computer hacker destroyed some of the edges in the graph. Normally, the maximum  $s - t$  flow in  $G$  is  $k$ . Unfortunately, there is currently no path from  $s$  to  $t$ . Fortunately, the sysadmins know that the hacker destroyed at most  $k$  edges of the graph.

The sysadmins are trying to diagnose which of the nodes of the graph are no longer reachable. They would like to avoid testing each node. They are using a monitoring tool with the following behavior. If you use the command  $\text{ping}(v)$ , for a given node  $v$ , it will tell you whether there is currently a path from  $s$  to  $v$  (so  $\text{ping}(t)$  will return `False` but  $\text{ping}(s)$  will return `True`).

Give an algorithm that accomplishes this task using only  $O(k \log n)$  pings. (You may assume that any algorithm you wish to run on the original network (before the hacker destroyed edges) runs for free, since you have a model of that network on your computer.)

### 1. Updating a maximum flow

Suppose you are given a directed graph  $G = (V, E)$ , with a positive integer capacity  $c_e$  on each edge  $e$ , a designated source  $s \in V$ , and a designated sink  $t \in V$ . You are also given a maximum  $s - t$  flow in  $G$ , defined by a flow value  $f_e$  on each edge  $e$ . The flow  $\{f_e\}$  is *acyclic*: There is no cycle in  $G$  on which all edges carry positive flow.

Now suppose we pick a specific edge  $e^* \in E$  and reduce its capacity by 1 unit. Show how to find a maximum flow in the resulting capacitated graph in time  $O(m + n)$ , where  $m$  is the number of edges in  $G$  and  $n$  is the number of nodes.

### 2. Cooking Schedule

You live in a cooperative apartment with  $n$  other people. The co-op needs to schedule cooks for the next  $n$  days, so that each person cooks one day and each day there is one cook. In addition, each member of the co-op has a list of days they are available to cook (and is unavailable to cook on the other days).

Because of your superior CS473 skills, the co-op selects you to come up with a schedule for cooking, so that everyone cooks on a day they are available.

- (a) Describe a bipartite graph  $G$  so that  $G$  has a perfect matching if and only if there is a feasible schedule for the co-op.
- (b) A friend of yours tried to help you out by coming up with a cooking schedule. Unfortunately, when you look at the schedule he created, you notice a big problem.  $n - 2$  of the people are scheduled for different nights on which they are available: no problem there. But the remaining two people are assigned to cook on the same night (and no one is assigned to the last night).

You want to fix your friend's mistake, but without having to recompute everything from scratch. Show that it's possible, using his "almost correct" schedule to decide in  $O(n^2)$  time whether there exists a feasible schedule.

### 3. Disjoint paths in a digraph

Let  $G = (V, E)$  be a directed graph, and suppose that for each node  $v$ , the number of edges into  $v$  is equal to the number of edges out of  $v$ . That is, for all  $v$ ,

$$|\{(u, v) : (u, v) \in E\}| = |\{(v, w) : (v, w) \in E\}|.$$

Let  $x, y$  be two nodes of  $G$ , and suppose that there exist  $k$  mutually edge-disjoint paths from  $x$  to  $y$ . Under these conditions, does it follow that there exist  $k$  mutually edge-disjoint paths from  $y$  to  $x$ . Give a proof or a counterexample with explanation.

**1. String matching: an example**

- (a) Build a finite automata to search for the string “bababoon”.
- (b) Use the automata from part (a) to build the prefix function for Knuth-Morris-Pratt.
- (c) Use the automata or the prefix function to search for “bababoon” in the string “babybaboonbuysbananasforotherbabababoons”.

**2. Cooking Schedule Strikes Back**

You live in a cooperative apartment with  $n$  other people. The co-op needs to schedule cooks for the next  $5n$  days, so that each person cooks five days and each day there is one cook. In addition, each member of the co-op has a list of days they are available to cook (and is unavailable to cook on the other days).

Because of your success at headbanging last week, the co-op again asks you to compose a cooking schedule. Unfortunately, you realize that no such schedule is possible. Give a schedule for the cooking so that no one has to cook on more than 2 days that they claim to be unavailable.

**3. String matching on Trees**

You are given a rooted tree  $T$  (not necessarily binary), in which each node has a character. You are also given a pattern  $P = p_1p_2 \dots p_l$ . Search for the string as a subtree. In other words, search for a subtree in which  $p_i$  is on a child of the node containing  $p_{i-1}$  for each  $2 \leq i \leq l$ .

### 1. Self-reductions

In each case below assume that you are given a black box which can answer the decision version of the indicated problem. Use a polynomial number of calls to the black box to construct the desired set.

- (a) Independent set: Given a graph  $G$  and an integer  $k$ , does  $G$  have a subset of  $k$  vertices that are pairwise nonadjacent?
- (b) Subset sum: Given a multiset (elements can appear more than once)  $X = \{x_1, x_2, \dots, x_k\}$  of positive integers, and a positive integer  $S$  does there exist a subset of  $X$  with sum exactly  $S$ ?

### 2. Lower Bounds

Give adversary arguments to prove the indicated lower bounds for the following problems:

- (a) Searching in a sorted array takes at least  $1 + \lfloor \lg_2 n \rfloor$  queries.
- (b) Let  $M$  be an  $n \times n$  array of real values that is increasing in both rows and columns. Prove that searching for a value requires at least  $n$  queries.

### 3. $k$ -coloring

Show that we can solve the problem of constructing a  $k$ -coloring of a graph by using a polynomial number of calls to a black box that determines whether a graph has such a  $k$ -coloring. (Hint: Try reducing via an intermediate problem that asks whether a partial coloring of a graph can be extended to a proper  $k$ -coloring.)

**1. NP-hardness Proofs: Restriction**

Prove that each of the following problems is NP-hard. In each part, find a special case of the given problem that is equivalent to a known NP-hard problem.

(a) **Longest Path**

Given a graph  $G$  and a positive integer  $k$ , does  $G$  contain a path with  $k$  or more edges?

(b) **Partition into Hamiltonian Subgraphs**

Given a graph  $G$  and a positive integer  $k$ , can the vertices of  $G$  be partitioned into at most  $k$  disjoint sets such that the graph induced by each set has a Hamiltonian cycle?

(c) **Set Packing**

Given a collection of finite sets  $C$  and a positive integer  $k$ , does  $C$  contain  $k$  disjoint sets?

(d) **Largest Common Subgraph**

Given two graphs  $G_1$  and  $G_2$  and a positive integer  $k$ , does there exist a graph  $G_3$  such that  $G_3$  is a subgraph of both  $G_1$  and  $G_2$  and  $G_3$  has at least  $k$  edges?

**2. Domino Line**

You are given an unusual set of dominoes; each domino has a number on each end, but the numbers may be arbitrarily large and some numbers appear on many dominoes, while other numbers only appear on a few dominoes. Your goal is to form a line using all the dominoes so that adjacent dominoes have the same number on their adjacent halves. Either give an efficient algorithm to solve the problem or show that it is NP-hard.

**3. Set Splitting**

Given a finite set  $S$  and a collection of subsets  $C$  is there a partition of  $S$  into two sets  $S_1$  and  $S_2$  such that no subset in  $C$  is contained entirely in  $S_1$  or  $S_2$ ? Show that the problem is NP-hard. (Hint: use NAE-3SAT, which is similar to 3SAT except that a satisfying assignment does not allow all 3 variables in a clause to be true.)

You have 120 minutes to answer four of these five questions.

**Write your answers in the separate answer booklet.**

### 1. Multiple Choice.

Each of the questions on this page has one of the following five answers:

A: $\Theta(1)$	B: $\Theta(\log n)$	C: $\Theta(n)$	D: $\Theta(n \log n)$	E: $\Theta(n^2)$
----------------	---------------------	----------------	-----------------------	------------------

Choose the correct answer for each question. Each correct answer is worth +1 point; each incorrect answer is worth  $-\frac{1}{2}$  point; each “I don’t know” is worth  $+\frac{1}{4}$  point. Your score will be rounded to the nearest *non-negative* integer. You do *not* need to justify your answers; just write the correct letter in the box.

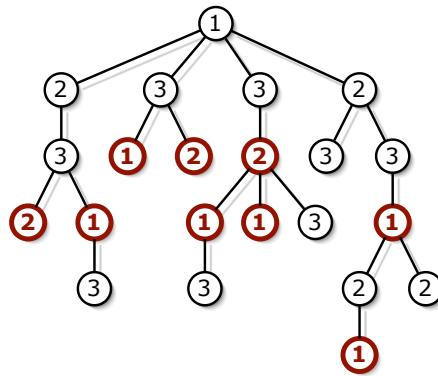
- (a) What is  $\frac{5}{n} + \frac{n}{5}$ ?
- (b) What is  $\sum_{i=1}^n \frac{n}{i}$ ?
- (c) What is  $\sum_{i=1}^n \frac{i}{n}$ ?
- (d) How many bits are required to represent the  $n$ th Fibonacci number in binary?
- (e) What is the solution to the recurrence  $T(n) = 2T(n/4) + \Theta(n)$ ?
- (f) What is the solution to the recurrence  $T(n) = 16T(n/4) + \Theta(n)$ ?
- (g) What is the solution to the recurrence  $T(n) = T(n - 1) + 1/n^2$ ?
- (h) What is the worst-case time to search for an item in a binary search tree?
- (i) What is the worst-case running time of quicksort?
- (j) What is the running time of the fastest possible algorithm to solve Sudoku puzzles?  
A Sudoku puzzle consists of a  $9 \times 9$  grid of squares, partitioned into nine  $3 \times 3$  sub-grids; some of the squares contain digits between 1 and 9. The goal of the puzzle is to enter digits into the blank squares, so that each digit between 1 and 9 appears exactly once in each row, each column, and each  $3 \times 3$  sub-grid. The initial conditions guarantee that the solution is unique.

2							4	
	7	5						
			1		9			
6	4			2				
	8						5	
		9			3		7	
	1		4					
				3		8		
5							6	

A Sudoku puzzle. **Don't try to solve this during the exam!**

2. Oh, no! You have been appointed as the gift czar for Giggle, Inc.'s annual mandatory holiday party! The president of the company, who is certifiably insane, has declared that *every* Giggle employee must receive one of three gifts: (1) an all-expenses-paid six-week vacation anywhere in the world, (2) an all-the-pancakes-you-can-eat breakfast for two at Jumping Jack Flash's Flapjack Stack Shack, or (3) a burning paper bag full of dog poop. Corporate regulations prohibit any employee from receiving the same gift as his/her direct supervisor. Any employee who receives a better gift than his/her direct supervisor will almost certainly be fired in a fit of jealousy. How do you decide what gifts everyone gets if you want to minimize the number of people that get fired?

More formally, suppose you are given a rooted tree  $T$ , representing the company hierarchy. You want to label each node in  $T$  with an integer 1, 2, or 3, such that every node has a different label from its parent.. The *cost* of an labeling is the number of nodes that have smaller labels than their parents. Describe and analyze an algorithm to compute the minimum cost of any labeling of the given tree  $T$ . (Your algorithm does *not* have to compute the actual best labeling—just its cost.)



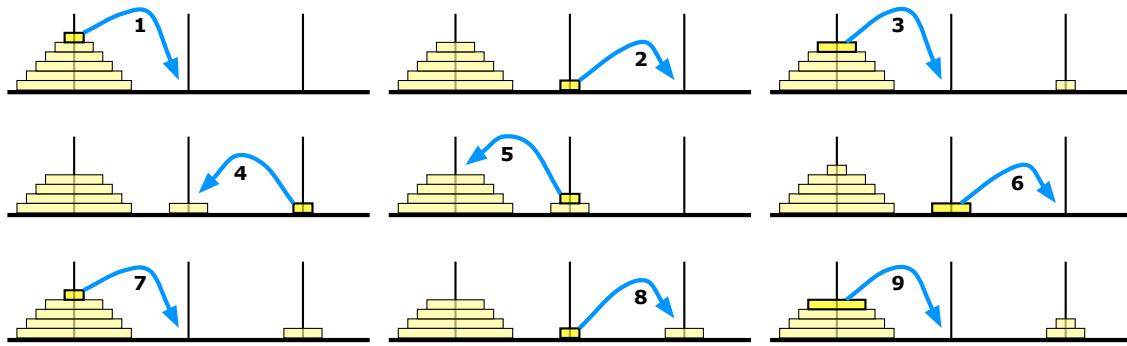
A tree labeling with cost 9. Bold nodes have smaller labels than their parents.  
This is *not* the optimal labeling for this tree.

3. Suppose you are given an array  $A[1..n]$  of  $n$  distinct integers, sorted in increasing order. Describe and analyze an algorithm to determine whether there is an index  $i$  such that  $A[i] = i$ , in  $o(n)$  time. [Hint: Yes, that's **little-oh** of  $n$ . What can you say about the sequence  $A[i] - i$ ?]
4. Describe and analyze a *polynomial-time* algorithm to compute the length of the longest common subsequence of two strings  $A[1..m]$  and  $B[1..n]$ . For example, given the strings 'DYNAMIC' and 'PROGRAMMING', your algorithm would return the number 3, because the longest common subsequence of those two strings is 'AMI'. You must give a complete, self-contained solution; don't just refer to HW1.

5. Recall that the Tower of Hanoi puzzle consists of three pegs and  $n$  disks of different sizes. Initially, all the disks are on one peg, stacked in order by size, with the largest disk on the bottom and the smallest disk on top. In a single move, you can transfer the highest disk on any peg to a different peg, except that you may never place a larger disk on top of a smaller one. The goal is to move all the disks onto one other peg.

Now suppose the pegs are arranged in a row, and you are forbidden to transfer a disk directly between the left and right pegs in a single move; every move must involve the middle peg. How many moves suffice to transfer all  $n$  disks from the left peg to the right peg under this restriction? **Prove your answer is correct.**

For full credit, give an *exact* upper bound. A correct upper bound using  $O(\cdot)$  notation (with a proof of correctness) is worth 7 points.



The first nine moves in a restricted Towers of Hanoi solution.

1. On an overnight camping trip in Sunnydale National Park, you are woken from a restless sleep by a scream. As you crawl out of your tent to investigate, a terrified park ranger runs out of the woods, covered in blood and clutching a crumpled piece of paper to his chest. As he reaches your tent, he gasps, “Get out... while... you...”, thrusts the paper into your hands, and falls to the ground. Checking his pulse, you discover that the ranger is stone dead.

You look down at the paper and recognize a map of the park, drawn as an undirected graph, where vertices represent landmarks in the park, and edges represent trails between those landmarks. (Trails start and end at landmarks and do not cross.) You recognize one of the vertices as your current location; several vertices on the boundary of the map are labeled EXIT.

On closer examination, you notice that someone (perhaps the poor dead park ranger) has written a real number between 0 and 1 next to each vertex and each edge. A scrawled note on the back of the map indicates that a number next to an edge is the probability of encountering a vampire along the corresponding trail, and a number next to a vertex is the probability of encountering a vampire at the corresponding landmark. (Vampires can’t stand each other’s company, so you’ll never see more than one vampire on the same trail or at the same landmark.) The note warns you that stepping off the marked trails will result in a slow and painful death.

You glance down at the corpse at your feet. Yes, his death certainly looked painful. Wait, was that a twitch? Are his teeth getting longer? After driving a tent stake through the undead ranger’s heart, you wisely decide to leave the park immediately.

Describe and analyze an efficient algorithm to find a path from your current location to an arbitrary EXIT node, such that the total *expected number* of vampires encountered along the path is as small as possible. *Be sure to account for both the vertex probabilities and the edge probabilities!*

2. Consider the following solution for the union-find problem, called *union-by-weight*. Each set leader  $\bar{x}$  stores the number of elements of its set in the field  $weight(\bar{x})$ . Whenever we UNION two sets, the leader of the *smaller* set becomes a new child of the leader of the *larger* set (breaking ties arbitrarily).

$\begin{array}{l} \text{MAKESET}(x): \\ \hline \text{parent}(x) \leftarrow x \\ \text{weight}(x) \leftarrow 1 \end{array}$	$\begin{array}{l} \text{UNION}(x, y) \\ \bar{x} \leftarrow \text{FIND}(x) \\ \bar{y} \leftarrow \text{FIND}(y) \\ \text{if } weight(\bar{x}) > weight(\bar{y}) \\ \quad \text{parent}(\bar{y}) \leftarrow \bar{x} \\ \quad weight(\bar{x}) \leftarrow weight(\bar{x}) + weight(\bar{y}) \\ \text{else} \\ \quad \text{parent}(\bar{x}) \leftarrow \bar{y} \\ \quad weight(\bar{x}) \leftarrow weight(\bar{x}) + weight(\bar{y}) \end{array}$
$\begin{array}{l} \text{FIND}(x): \\ \hline \text{while } x \neq \text{parent}(x) \\ \quad x \leftarrow \text{parent}(x) \\ \text{return } x \end{array}$	

**Prove** that if we use union-by-weight, the *worst-case* running time of FIND is  $O(\log n)$ .

3. *Prove or disprove*<sup>1</sup> each of the following statements.

- (a) Let  $G$  be an arbitrary undirected graph with arbitrary distinct weights on the edges. The minimum spanning tree of  $G$  includes the lightest edge in every cycle in  $G$ .
  - (b) Let  $G$  be an arbitrary undirected graph with arbitrary distinct weights on the edges. The minimum spanning tree of  $G$  excludes the heaviest edge in every cycle in  $G$ .
4. In Homework 2, you were asked to analyze the following algorithm to find the  $k$ th smallest element from an unsorted array. (The algorithm is presented here in iterative form, rather than the recursive form you saw in the homework, but it's exactly the same algorithm.)

```

QUICKSELECT( $A[1..n], k$ ):
     $i \leftarrow 1; j \leftarrow n$ 
    while  $i \leq j$ 
         $r \leftarrow \text{PARTITION}(A[i..j], \text{RANDOM}(i, j))$ 
        if  $r = k$ 
            return  $A[r]$ 
        else if  $r > k$ 
             $j \leftarrow r - 1$ 
        else
             $i \leftarrow r + 1$ 

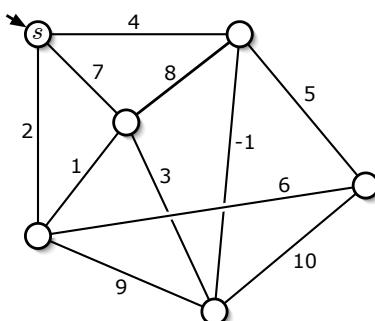
```

The algorithm relies on two subroutines.  $\text{RANDOM}(i, j)$  returns an integer chosen uniformly at random from the range  $[i..j]$ .  $\text{PARTITION}(A[i..j], p)$  partitions the subarray  $A[i..j]$  using the pivot value  $A[p]$  and returns the new index of the pivot value in the partitioned array.

What is the *exact* expected number of iterations of the main loop when  $k = 1$ ? *Prove* your answer is correct. A correct  $\Theta(\cdot)$  bound (with proof) is worth 7 points. You may assume that the input array  $A[]$  contains  $n$  distinct integers.

5. Find the following spanning trees for the weighted graph shown below.

- (a) A breadth-first spanning tree rooted at  $s$ .
- (b) A depth-first spanning tree rooted at  $s$ .
- (c) A shortest-path tree rooted at  $s$ .
- (d) A minimum spanning tree.



You do *not* need to justify your answers; just clearly indicate the edges of each spanning tree. Yes, one of the edges has negative weight.

---

<sup>1</sup>But not both! If you give us *both* a proof and a disproof for the same statement, you will get no credit, even if one of your arguments is correct.

1. A *double-Hamiltonian* circuit in an undirected graph  $G$  is a closed walk that visits every vertex in  $G$  exactly *twice*, possibly by traversing some edges more than once. *Prove* that it is NP-hard to determine whether a given undirected graph contains a double-Hamiltonian circuit.
  
2. Suppose you are running a web site that is visited by the same set of people every day. Each visitor claims membership in one or more *demographic groups*; for example, a visitor might describe himself as male, 31-40 years old, a resident of Illinois, an academic, a blogger, a Joss Whedon fan<sup>1</sup>, and a Sports Racer.<sup>2</sup> Your site is supported by advertisers. Each advertiser has told you which demographic groups should see its ads and how many of its ads you must show each day. Altogether, there are  $n$  visitors,  $k$  demographic groups, and  $m$  advertisers.

Describe an efficient algorithm to determine, given all the data described in the previous paragraph, whether you can show each visitor exactly *one* ad per day, so that every advertiser has its desired number of ads displayed, and every ad is seen by someone in an appropriate demographic group.

3. Describe and analyze a data structure to support the following operations on an array  $X[1..n]$  as quickly as possible. Initially,  $X[i] = 0$  for all  $i$ .
  - Given an index  $i$  such that  $X[i] = 0$ , set  $X[i]$  to 1.
  - Given an index  $i$ , return  $X[i]$ .
  - Given an index  $i$ , return the smallest index  $j \geq i$  such that  $X[j] = 0$ , or report that no such index exists.

For full credit, the first two operations should run in *worst-case constant* time, and the amortized cost of the third operation should be as small as possible. [Hint: Use a modified union-find data structure.]

4. The next time you are at a party, one of the guests will suggest everyone play a round of Three-Way Mumbledypeg, a game of skill and dexterity that requires three teams and a knife. The official Rules of Three-Way Mumbledypeg (fixed during the Holy Roman Three-Way Mumbledypeg Council in 1625) require that (1) each team *must* have at least one person, (2) any two people on the same team *must* know each other, and (3) everyone watching the game *must* be on one of the three teams. Of course, it will be a really *fun* party; nobody will want to leave. There will be several pairs of people at the party who don't know each other. The host of the party, having heard thrilling tales of your prowess in all things algorithmic, will hand you a list of which pairs of partygoers know each other and ask you to choose the teams, while he sharpens the knife.

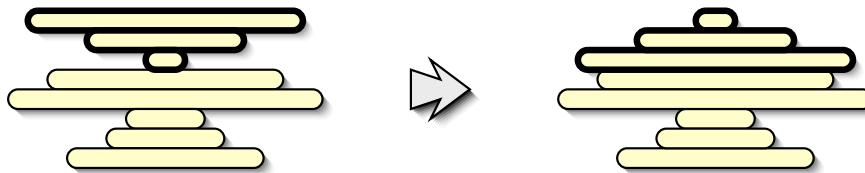
Either describe and analyze a polynomial time algorithm to determine whether the partygoers can be split into three legal Three-Way Mumbledypeg teams, or prove that the problem is NP-hard.

---

<sup>1</sup>Har har har! Mine is an evil laugh! Now die!

<sup>2</sup>It's Ride the Fire Eagle Danger Day!

5. Suppose you are given a stack of  $n$  pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a *flip*—insert a spatula under the top  $k$  pancakes, for some integer  $k$  between 1 and  $n$ , and flip them all over.



Flipping the top three pancakes.

- (a) Describe an efficient algorithm to sort an arbitrary stack of  $n$  pancakes. *Exactly* how many flips does your algorithm perform in the worst case? (For full credit, your algorithm should perform as few flips as possible; an optimal  $\Theta()$  bound is worth three points.)
  - (b) Now suppose one side of each pancake is burned. *Exactly* how many flips do you need to sort the pancakes *and* have the burned side of every pancake on the bottom? (For full credit, your algorithm should perform as few flips as possible; an optimal  $\Theta()$  bound is worth three points.)
6. Describe and analyze an efficient algorithm to find the length of the longest substring that appears both forward and backward in an input string  $T[1..n]$ . The forward and backward substrings must not overlap. Here are several examples:

- Given the input string ALGORITHM, your algorithm should return 0.
- Given the input string RECURSION, your algorithm should return 1, for the substring R.
- Given the input string REDIVIDE, your algorithm should return 3, for the substring EDI. (The forward and backward substrings must not overlap!)
- Given the input string DYNAMICPROGRAMMINGMANYTIMES, your algorithm should return 4, for the substring YNAM.

For full credit, your algorithm should run in  $O(n^2)$  time.

7. A *double-Eulerian* circuit in an undirected graph  $G$  is a closed walk that traverses every edge in  $G$  exactly twice. Describe and analyze a *polynomial-time* algorithm to determine whether a given undirected graph contains a double-Eulerian circuit.

# CS 473G: Graduate Algorithms, Spring 2007

## Homework 0

Due in class at 11:00am, Tuesday, January 30, 2007

Name:	
Net ID:	Alias:

I understand the Course Policies.

- 
- Neatly print your full name, your NetID, and an alias of your choice in the boxes above, and staple this page to your solution to problem 1. **By providing an alias, you agree to let us list your grades; if you do not provide an alias, your grades will not be listed.** For privacy reasons, your alias should not resemble your name, your NetID, your university ID number, or (God forbid!) your Social Security number. Please use the same alias for every homework and exam.
  - Read the Course Policies on the course web site, and then check the box above. Among other things, this page describes what we expect in your homework solutions, as well as policies on grading standards, regrading, extra credit, and plagiarism. In particular:
    - Submit each numbered problem separately, on its own piece(s) of paper. If you need more than one page for a problem, staple just *those* pages together, but keep different problems separate. **Do not staple your entire homework together.**
    - You may use *any* source at your disposal—paper, electronic, or human—but you *must* write your answers in your own words, and you *must* cite every source that you use.
    - Algorithms or proofs containing phrases like “and so on” or “repeat this for all  $n$ ”, instead of an explicit loop, recursion, or induction, are worth zero points.
    - Answering “I don’t know” to any homework or exam problem is worth 25% partial credit.

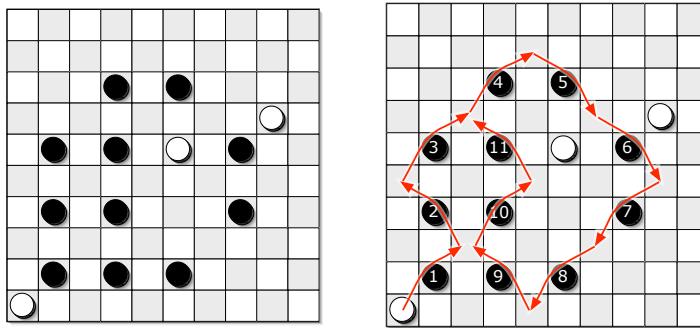
If you have *any* questions, please ask during lecture or office hours, or post your question to the course newsgroup.

- This homework tests your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, discrete probability, graphs, and most importantly, induction—to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** The early chapters of Kleinberg and Tardos (or any algorithms textbook) should be sufficient review, but you may also want consult your favorite discrete mathematics and data structures textbooks.
  - Every homework will have five problems, each worth 10 points. Stars indicate more challenging problems. Many homeworks will also include an extra-credit problem.
-

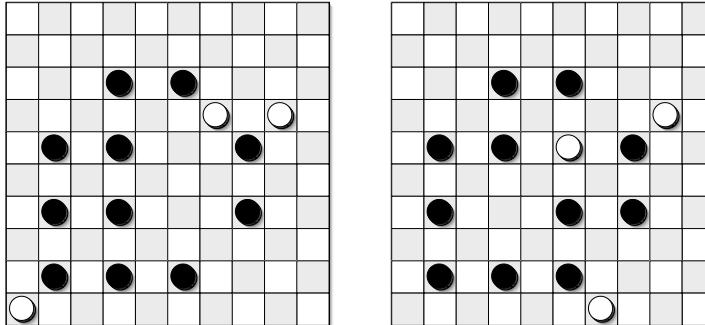
\*1. Draughts/checkers is a game played on an  $m \times m$  grid of squares, alternately colored light and dark. (The game is usually played on an  $8 \times 8$  or  $10 \times 10$  board, but the rules easily generalize to any board size.) Each dark square is occupied by at most one game piece (usually called a *checker* in the U.S.), which is either black or white; light squares are always empty. One player (“White”) moves the white pieces; the other (“Black”) moves the black pieces.

Consider the following simple version of the game, essentially American checkers or British draughts, but where every piece is a king.<sup>1</sup> Pieces can be moved in any of the four diagonal directions, either one or two steps at a time. On each turn, a player either *moves* one of her pieces one step diagonally into an empty square, or makes a series of *jumps* with one of her checkers. In a single jump, a piece moves to an empty square two steps away in any diagonal direction, but only if the intermediate square is occupied by a piece of the opposite color; this enemy piece is *captured* and immediately removed from the board. Multiple jumps are allowed in a single turn as long as they are made by the same piece. A player wins if her opponent has no pieces left on the board.

Describe an algorithm<sup>2</sup> that correctly determines whether White can capture every black piece, thereby winning the game, *in a single turn*. The input consists of the width of the board ( $m$ ), a list of positions of white pieces, and a list of positions of black pieces. For full credit, your algorithm should run in  $O(n)$  time, where  $n$  is the total number of pieces, but any algorithm that runs in time polynomial in  $n$  and  $m$  is worth significant partial credit.



White wins in one turn.



White cannot win in one turn from either of these positions.

[Hint: The greedy strategy—make arbitrary jumps until you get stuck—does **not** always find a winning sequence of jumps even when one exists.]

<sup>1</sup>Most variants of draughts have ‘flying kings’, which behave very differently than what’s described here.

<sup>2</sup>Since you’ve read the Course Policies, you know what this phrase means.

2. (a) Prove that any positive integer can be written as the sum of distinct powers of 2. [*Hint: “Write the number in binary” is **not** a proof; it just restates the problem.*] For example:

$$\begin{aligned}16 + 1 &= 17 = 2^4 + 2^0 \\16 + 4 + 2 + 1 &= 23 = 2^4 + 2^2 + 2^1 + 2^0 \\32 + 8 + 1 &= 42 = 2^5 + 2^3 + 2^1\end{aligned}$$

- (b) Prove that *any* integer (positive, negative, or zero) can be written as the sum of distinct powers of  $-2$ . For example:

$$\begin{aligned}-32 + 16 - 2 + 1 &= -17 = (-2)^5 + (-2)^4 + (-2)^1 + (-2)^0 \\64 - 32 - 8 - 2 + 1 &= 23 = (-2)^6 + (-2)^5 + (-2)^3 + (-2)^1 + (-2)^0 \\64 - 32 + 16 - 8 + 4 - 2 &= 42 = (-2)^6 + (-2)^5 + (-2)^4 + (-2)^3 + (-2)^2 + (-2)^1\end{aligned}$$

3. Whenever groups of pigeons gather, they instinctively establish a *pecking order*. For any pair of pigeons, one pigeon always pecks the other, driving it away from food or potential mates. The same pair of pigeons always chooses the same pecking order, even after years of separation, no matter what other pigeons are around. Surprisingly, the overall pecking order can contain cycles—for example, pigeon A pecks pigeon B, which pecks pigeon C, which pecks pigeon A.

Prove that any finite set of pigeons can be arranged in a row from left to right so that every pigeon pecks the pigeon immediately to its left.

4. On their long journey from Denmark to England, Rosencrantz and Guildenstern amuse themselves by playing the following game with a fair coin. First Rosencrantz flips the coin over and over until it comes up tails. Then Guildenstern flips the coin over and over until he gets as many heads in a row as Rosencrantz got on his turn. Here are three typical games:

Rosencrantz: H H T

Guildenstern: H T H H

Rosencrantz: T

Guildenstern: (no flips)

Rosencrantz: H H H T

Guildenstern: T H H T H H T H T H H H

- (a) What is the expected number of flips in one of Rosencrantz’s turns?
- (b) Suppose Rosencrantz flips  $k$  heads in a row on his turn. What is the expected number of flips in Guildenstern’s next turn?
- (c) What is the expected total number of flips (by both Rosencrantz and Guildenstern) in a single game?

**Prove** that your answers are correct. If you have to appeal to “intuition” or “common sense”, your answer is almost certainly wrong! You must give *exact* answers for full credit, but a correct asymptotic bound for part (b) is worth significant credit.

5. (a) [5 pts] Solve the following recurrences. State tight asymptotic bounds for each function in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ . Assume reasonable but nontrivial base cases. If your solution requires a particular base case, say so.

$$A(n) = 3A(n/9) + \sqrt{n}$$

$$B(n) = 4B(n-1) - 4B(n-2)$$

$$C(n) = \frac{\pi C(n-1)}{\sqrt{2} C(n-2)} \quad [\text{Hint: This is easy!}]$$

$$D(n) = \max_{n/4 < k < 3n/4} (D(k) + D(n-k) + n)$$

$$E(n) = 2E(n/2) + 4E(n/3) + 2E(n/6) + n^2$$

**Do not turn in proofs**—just a list of five functions—but you should do them anyway, just for practice. [*Hint: On the course web page, you can find a handout describing several techniques for solving recurrences.*]

- (b) [5 pts] Sort the functions in the box from asymptotically smallest to asymptotically largest, indicating ties if there are any. **Do not turn in proofs**—just a sorted list of 16 functions—but you should do them anyway, just for practice.

To simplify your answer, write  $f(n) \ll g(n)$  to indicate that  $f(n) = o(g(n))$ , and write  $f(n) \equiv g(n)$  to mean  $f(n) = \Theta(g(n))$ . For example, the functions  $n^2, n, \binom{n}{2}, n^3$  could be sorted either as  $n \ll n^2 \equiv \binom{n}{2} \ll n^3$  or as  $n \ll \binom{n}{2} \equiv n^2 \ll n^3$ .

$n$	$\lg n$	$\sqrt{n}$	$3^n$
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$3\sqrt{n}$	$\sqrt{3^n}$
$3^{\lg n}$	$\lg(3^n)$	$3^{\lg \sqrt{n}}$	$3^{\sqrt{\lg n}}$
$\sqrt{3^{\lg n}}$	$\lg(3^{\sqrt{n}})$	$\lg \sqrt{3^n}$	$\sqrt{\lg(3^n)}$

Recall that  $\lg n = \log_2 n$ .

# CS 473G: Graduate Algorithms, Spring 2007

## Homework 1

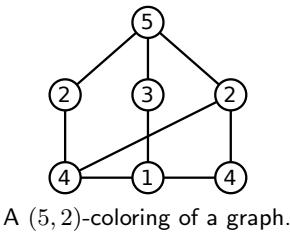
Due February 6, 2007

---

Remember to submit **separate, individually stapled** solutions to each of the problems.

---

1. Jeff tries to make his students happy. At the beginning of class, he passes out a questionnaire to students which lists a number of possible course policies in areas where he is flexible. Every student is asked to respond to each possible course policy with one of “strongly favor”, “mostly neutral”, or “strongly oppose”. Each student may respond with “strongly favor” or “strongly oppose” to at most five questions. Because Jeff’s students are very understanding, each student is happy if he or she prevails in just one of his or her strong policy preferences. Either describe a polynomial time algorithm for setting course policy to maximize the number of happy students or show that the problem is NP-hard.
2. Consider a variant 3SAT' of 3SAT which asks, given a formula  $\phi$  in conjunctive normal form in which each clause contains at most 3 literals and each variable appears in at most 3 clauses, is  $\phi$  satisfiable? Prove that 3SAT' is NP-complete.
3. For each problem below, either describe a polynomial-time algorithm to solve the problem or prove that the problem is NP-complete.
  - (a) A *double-Eulerian* circuit in an undirected graph  $G$  is a closed walk that traverses every edge in  $G$  exactly twice. Given a graph  $G$ , does  $G$  have a double-Eulerian circuit?
  - (b) A *double-Hamiltonian* circuit in an undirected graph  $G$  is a closed walk that visits every vertex in  $G$  exactly twice. Given a graph  $G$ , does  $G$  have a double-Hamiltonian circuit?
4. Suppose you have access to a magic black box; if you give it a graph  $G$  as input, the black box will tell you, in constant time, if there is a proper 3-coloring of  $G$ . Describe a polynomial time algorithm which, given a graph  $G$  that is 3-colorable, uses the black box to compute a 3-coloring of  $G$ .
5. Let  $C_5$  be the graph which is a cycle on five vertices. A  $(5, 2)$ -coloring of a graph  $G$  is a function  $f : V(G) \rightarrow \{1, 2, 3, 4, 5\}$  such that every pair  $\{u, v\}$  of adjacent vertices in  $G$  is mapped to a pair  $\{f(u), f(v)\}$  of vertices in  $C_5$  which are at distance two from each other.



Using a reduction from 5COLOR, prove that the problem of deciding whether a given graph  $G$  has a  $(5, 2)$ -coloring is NP-complete.

# CS 473G: Graduate Algorithms, Spring 2007

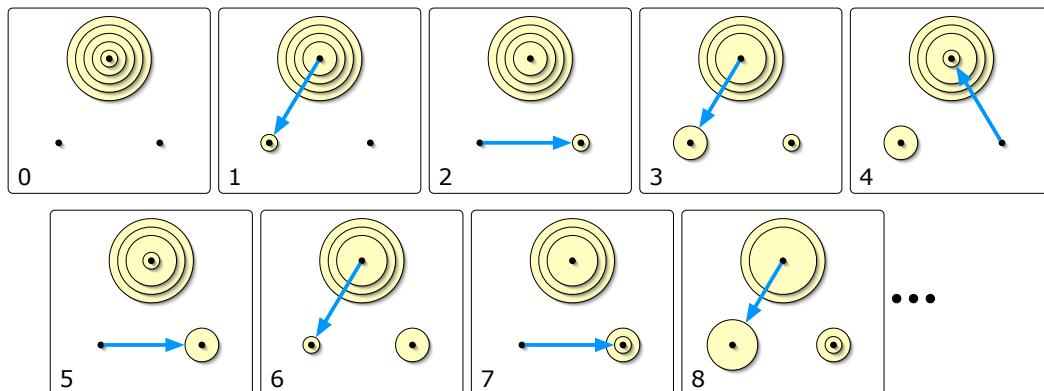
## Homework 2

Due Tuesday, February 20, 2007

Remember to submit **separate, individually stapled** solutions to each problem.

As a general rule, a complete full-credit solution to any homework problem should fit into two typeset pages (or five hand-written pages). If your solution is significantly longer than this, you may be including too much detail.

1. Consider a restricted variant of the Tower of Hanoi puzzle, where the three needles are arranged in a triangle, and you are required to move each disk *counterclockwise*. Describe an algorithm to move a stack of  $n$  disks from one needle to another. *Exactly* how many moves does your algorithm perform? To receive full credit, your algorithm must perform the minimum possible number of moves. [Hint: Your answer will depend on whether you are moving the stack clockwise or counterclockwise.]



A top view of the first eight moves in a counterclockwise Towers of Hanoi solution

- \*2. You find yourself working for The Negation Company (“We Contradict Everything...Not!”), the world’s largest producer of multi-bit Boolean inverters. Thanks to a recent mining discovery, the market prices for amphigen and opoterium, the key elements used in AND and OR gates, have plummeted to almost nothing. Unfortunately, the market price of inverton, the essential element required to build NOT gates, has recently risen sharply as natural supplies are almost exhausted. Your boss is counting on you to radically redesign the company’s only product in response to these radically new market prices.

Design a Boolean circuit that inverts  $n = 2^k - 1$  bits, using only  $k$  NOT gates but *any* number of AND and OR gates. The input to your circuit consists of  $n$  bits  $x_1, x_2, \dots, x_n$ , and the output consists of  $n$  bits  $y_1, y_2, \dots, y_n$ , where each output bit  $y_i$  is the inverse of the corresponding input bit  $x_i$ . [Hint: Solve the case  $k = 2$  first.]

3. (a) Let  $X[1..m]$  and  $Y[1..n]$  be two arbitrary arrays. A *common supersequence* of  $X$  and  $Y$  is another sequence that contains both  $X$  and  $Y$  as subsequences. Give a simple recursive definition for the function  $scs(X, Y)$ , which gives the length of the *shortest* common supersequence of  $X$  and  $Y$ .  
(b) Call a sequence  $X[1..n]$  *oscillating* if  $X[i] < X[i+1]$  for all even  $i$ , and  $X[i] > X[i+1]$  for all odd  $i$ . Give a simple recursive definition for the function  $los(X)$ , which gives the length of the longest oscillating subsequence of an arbitrary array  $X$  of integers.  
(c) Call a sequence  $X[1..n]$  of integers *accelerating* if  $2 \cdot X[i] < X[i-1] + X[i+1]$  for all  $i$ . Give a simple recursive definition for the function  $lxs(X)$ , which gives the length of the longest accelerating subsequence of an arbitrary array  $X$  of integers.

Each recursive definition should translate directly into a recursive algorithm, *but you do not need to analyze these algorithms*. We are looking for correctness and *simplicity*, not algorithmic efficiency. Not yet, anyway.

4. Describe an algorithm to solve 3SAT in time  $O(\phi^n \text{ poly}(n))$ , where  $\phi = (1 + \sqrt{5})/2 \approx 1.618034$ .  
[Hint: Prove that in each recursive call, either you have just eliminated a pure literal, or the formula has a clause with at most two literals. What recurrence leads to this running time?]
5. (a) Describe an algorithm that determines whether a given set of  $n$  integers contains two distinct elements that sum to zero, in  $O(n \log n)$  time.  
(b) Describe an algorithm that determines whether a given set of  $n$  integers contains three distinct elements that sum to zero, in  $O(n^2)$  time.  
(c) Now suppose the input set  $X$  contains  $n$  integers between  $-10000n$  and  $10000n$ . Describe an algorithm that determines whether  $X$  contains three **distinct** elements that sum to zero, in  $O(n \log n)$  time.

For example, if the input set is  $\{-10, -9, -7, -3, 1, 3, 5, 11\}$ , your algorithm for part (a) should return TRUE, because  $(-3) + 3 = 0$ , and your algorithms for parts (b) and (c) should return FALSE, even though  $(-10) + 5 + 5 = 0$ .

# CS 473G: Graduate Algorithms, Spring 2007

## Homework 3

Due Friday, March 9, 2007

---

Remember to submit **separate, individually stapled** solutions to each problem.

As a general rule, a complete, full-credit solution to any homework problem should fit into two typeset pages (or five hand-written pages). If your solution is significantly longer than this, you may be including too much detail.

---

1. (a) Let  $X[1..m]$  and  $Y[1..n]$  be two arbitrary arrays. A *common supersequence* of  $X$  and  $Y$  is another sequence that contains both  $X$  and  $Y$  as subsequences. Describe and analyze an efficient algorithm to compute the function  $scs(X, Y)$ , which gives the length of the shortest common supersequence of  $X$  and  $Y$ .
- (b) Call a sequence  $X[1..n]$  *oscillating* if  $X[i] < X[i+1]$  for all even  $i$ , and  $X[i] > X[i+1]$  for all odd  $i$ . Describe and analyze an efficient algorithm to compute the function  $los(X)$ , which gives the length of the longest oscillating subsequence of an arbitrary array  $X$  of integers.
- (c) Call a sequence  $X[1..n]$  of integers *accelerating* if  $2 \cdot X[i] < X[i-1] + X[i+1]$  for all  $i$ . Describe and analyze an efficient algorithm to compute the function  $lcs(X)$ , which gives the length of the longest accelerating subsequence of an arbitrary array  $X$  of integers.

*[Hint: Use the recurrences you found in Homework 2. You do not need to prove again that these recurrences are correct.]*

2. Describe and analyze an algorithm to solve the traveling salesman problem in  $O(2^n \text{poly}(n))$  time. Given an undirected  $n$ -vertex graph  $G$  with weighted edges, your algorithm should return the weight of the lightest Hamiltonian cycle in  $G$  (or  $\infty$  if  $G$  has no Hamiltonian cycles).
3. Let  $G$  be an arbitrary undirected graph. A set of cycles  $\{c_1, \dots, c_k\}$  in  $G$  is *redundant* if it is non-empty and every edge in  $G$  appears in an even number of  $c_i$ 's. A set of cycles is *independent* if it contains no redundant subsets. (In particular, the empty set is independent.) A maximal independent set of cycles is called a *cycle basis* for  $G$ .

- (a) Let  $C$  be any cycle basis for  $G$ . Prove that for any cycle  $\gamma$  in  $G$  **that is not an element of  $C$** , there is a subset  $A \subseteq C$  such that  $A \cup \{\gamma\}$  is redundant. In other words, prove that  $\gamma$  is the ‘exclusive or’ of some subset of basis cycles.

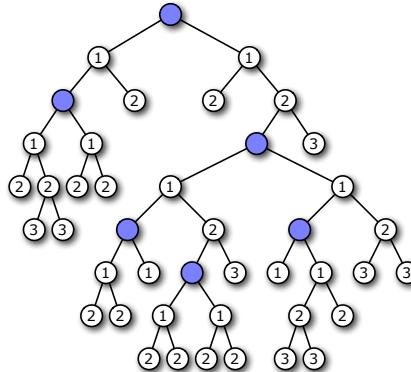
**Solution:** The claim follows directly from the definitions. A cycle basis is a *maximal* independent set, so if  $C$  is a cycle basis, then for any cycle  $\gamma \notin C$ , the larger set  $C \cup \{\gamma\}$  cannot be an independent set, so it must contain a redundant subset. On the other hand, if  $C$  is a basis, then  $C$  is independent, so  $C$  contains no redundant subsets. Thus,  $C \cup \{\gamma\}$  must have a redundant subset  $B$  that contains  $\gamma$ . Let  $A = B \setminus \{\gamma\}$ . ■

- (b) Prove that the set of independent cycle sets form a matroid.
- (c) Now suppose each edge of  $G$  has a weight. Define the weight of a cycle to be the total weight of its edges, and the weight of a set of cycles to be the total weight of all cycles in the set. (Thus, each edge is counted once for every cycle in which it appears.) Describe and analyze an efficient algorithm to compute the minimum-weight cycle basis of  $G$ .
4. Let  $T$  be a rooted binary tree with  $n$  vertices, and let  $k \leq n$  be a positive integer. We would like to mark  $k$  vertices in  $T$  so that every vertex has a nearby marked ancestor. More formally, we define the *clustering cost* of a clustering of any subset  $K$  of vertices as

$$\text{cost}(K) = \max_v \text{cost}(v, K),$$

where the maximum is taken over all vertices  $v$  in the tree, and

$$\text{cost}(v, K) = \begin{cases} 0 & \text{if } v \in K \\ \infty & \text{if } v \text{ is the root of } T \text{ and } v \notin K \\ 1 + \text{cost}(\text{parent}(v)) & \text{otherwise} \end{cases}$$

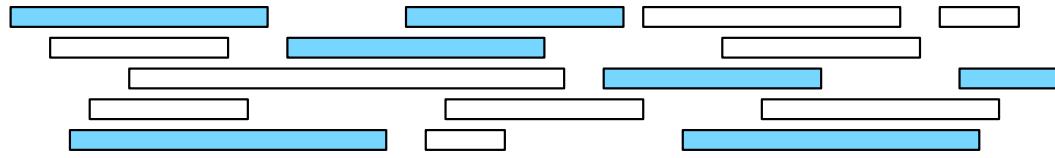


A subset of 5 vertices with clustering cost 3

Describe and analyze a dynamic-programming algorithm to compute the minimum clustering cost of any subset of  $k$  vertices in  $T$ . For full credit, your algorithm should run in  $O(n^2 k^2)$  time.

5. Let  $X$  be a set of  $n$  intervals on the real line. A subset of intervals  $Y \subseteq X$  is called a *tiling path* if the intervals in  $Y$  cover the intervals in  $X$ , that is, any real value that is contained in some interval in  $X$  is also contained in some interval in  $Y$ . The *size* of a tiling cover is just the number of intervals.

Describe and analyze an algorithm to compute the smallest tiling path of  $X$  as quickly as possible. Assume that your input consists of two arrays  $X_L[1..n]$  and  $X_R[1..n]$ , representing the left and right endpoints of the intervals in  $X$ . If you use a greedy algorithm, you must prove that it is correct.



A set of intervals. The seven shaded intervals form a tiling path.

# CS 473G: Graduate Algorithms, Spring 2007

## Homework 4

Due March 29, 2007

---

Please remember to submit **separate, individually stapled** solutions to each problem.

---

1. Given a graph  $G$  with edge weights and an integer  $k$ , suppose we wish to partition the vertices of  $G$  into  $k$  subsets  $S_1, S_2, \dots, S_k$  so that the sum of the weights of the edges that cross the partition (*i.e.*, have endpoints in different subsets) is as large as possible.
  - (a) Describe an efficient  $(1 - 1/k)$ -approximation algorithm for this problem.
  - (b) Now suppose we wish to minimize the sum of the weights of edges that do *not* cross the partition. What approximation ratio does your algorithm from part (a) achieve for the new problem? Justify your answer.
2. In class, we saw a  $(3/2)$ -approximation algorithm for the metric traveling salesman problem. Here, we consider computing minimum cost Hamiltonian *paths*. Our input consists of a graph  $G$  whose edges have weights that satisfy the triangle inequality. Depending upon the problem, we are also given zero, one, or two endpoints.
  - (a) If our input includes zero endpoints, describe a  $(3/2)$ -approximation to the problem of computing a minimum cost Hamiltonian path.
  - (b) If our input includes one endpoint  $u$ , describe a  $(3/2)$ -approximation to the problem of computing a minimum cost Hamiltonian path that starts at  $u$ .
  - (c) If our input includes two endpoints  $u$  and  $v$ , describe a  $(5/3)$ -approximation to the problem of computing a minimum cost Hamiltonian path that starts at  $u$  and ends at  $v$ .
3. Consider the greedy algorithm for metric TSP: start at an arbitrary vertex  $u$ , and at each step, travel to the closest unvisited vertex.
  - (a) Show that the greedy algorithm for metric TSP is an  $O(\log n)$ -approximation, where  $n$  is the number of vertices. [*Hint: Argue that the  $k$ th least expensive edge in the tour output by the greedy algorithm has weight at most  $\text{OPT}/(n - k + 1)$ ; try  $k = 1$  and  $k = 2$  first.*]
  - \*(b) **[Extra Credit]** Show that the greedy algorithm for metric TSP is no better than an  $O(\log n)$ -approximation.
4. In class, we saw that the greedy algorithm gives an  $O(\log n)$ -approximation for vertex cover. Show that our analysis of the greedy algorithm is asymptotically tight by describing, for any positive integer  $n$ , an  $n$ -vertex graph for which the greedy algorithm produces a vertex cover of size  $\Omega(\log n) \cdot \text{OPT}$ .

5. Recall the minimum makespan scheduling problem: Given an array  $T[1..n]$  of processing times for  $n$  jobs, we wish to schedule the jobs on  $m$  machines to minimize the time at which the last job terminates. In class, we proved that the greedy scheduling algorithm has an approximation ratio of at most 2.
- (a) Prove that for any set of jobs, the makespan of the greedy assignment is at most  $(2 - 1/m)$  times the makespan of the optimal assignment.
  - (b) Describe a set of jobs such that the makespan of the greedy assignment is exactly  $(2 - 1/m)$  times the makespan of the optimal assignment.
  - (c) Describe an efficient algorithm to solve the minimum makespan scheduling problem *exactly* if every processing time  $T[i]$  is a power of two.

# CS 473G: Graduate Algorithms, Spring 2007

## Homework 5

Due Thursday, April 17, 2007

---

Please remember to submit **separate, individually stapled** solutions to each problem.

---

Unless a problem specifically states otherwise, you can assume the function  $\text{RANDOM}(k)$ , which returns an integer chosen independently and uniformly at random from the set  $\{1, 2, \dots, k\}$ , in  $O(1)$  time. For example, to perform a fair coin flip, you would call  $\text{RANDOM}(2)$ .

---

1. Suppose we want to write an efficient function  $\text{RANDOMPERMUTATION}(n)$  that returns a permutation of the integers  $\langle 1, \dots, n \rangle$  chosen uniformly at random.

- (a) What is the expected running time of the following  $\text{RANDOMPERMUTATION}$  algorithm?

```
RANDOMPERMUTATION( $n$ ):  
    for  $i \leftarrow 1$  to  $n$   
         $\pi[i] \leftarrow \text{EMPTY}$   
    for  $i \leftarrow 1$  to  $n$   
         $j \leftarrow \text{RANDOM}(n)$   
        while ( $\pi[j] \neq \text{EMPTY}$ )  
             $j \leftarrow \text{RANDOM}(n)$   
         $\pi[j] \leftarrow i$   
    return  $\pi$ 
```

- (b) Consider the following partial implementation of  $\text{RANDOMPERMUTATION}$ .

```
RANDOMPERMUTATION( $n$ ):  
    for  $i \leftarrow 1$  to  $n$   
         $A[i] \leftarrow \text{RANDOM}(n)$   
     $\pi \leftarrow \text{SOMEFUNCTION}(A)$   
    return  $\pi$ 
```

Prove that if the subroutine  $\text{SOMEFUNCTION}$  is deterministic, then this algorithm cannot be correct. [Hint: There is a one-line proof.]

- (c) Describe and analyze an  $\text{RANDOMPERMUTATION}$  algorithm whose expected worst-case running time is  $O(n)$ .
- \*(d) [Extra Credit] Describe and analyze an  $\text{RANDOMPERMUTATION}$  algorithm that uses only fair coin flips; that is, your algorithm can't call  $\text{RANDOM}(k)$  with  $k > 2$ . Your algorithm should run in  $O(n \log n)$  time with high probability.

2. A *meldable priority queue* stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:

- **MAKEQUEUE**: Return a new priority queue containing the empty set.
- **FINDMIN( $Q$ )**: Return the smallest element of  $Q$  (if any).
- **DELETEMIN( $Q$ )**: Remove the smallest element in  $Q$  (if any).
- **INSERT( $Q, x$ )**: Insert element  $x$  into  $Q$ , if it is not already there.
- **DECREASEKEY( $Q, x, y$ )**: Replace an element  $x \in Q$  with a smaller element  $y$ . (If  $y > x$ , the operation fails.) The input is a pointer directly to the node in  $Q$  that contains  $x$ .
- **DELETE( $Q, x$ )**: Delete the element  $x \in Q$ . The input is a pointer directly to the node in  $Q$  that contains  $x$ .
- **MELD( $Q_1, Q_2$ )**: Return a new priority queue containing all the elements of  $Q_1$  and  $Q_2$ ; this operation destroys  $Q_1$  and  $Q_2$ .

A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm:

```

MELD( $Q_1, Q_2$ ):
  if  $Q_1$  is empty, return  $Q_2$ 
  if  $Q_2$  is empty, return  $Q_1$ 
  if  $key(Q_1) > key(Q_2)$ 
    swap  $Q_1 \leftrightarrow Q_2$ 
  with probability  $1/2$ 
     $left(Q_1) \leftarrow \text{MELD}(left(Q_1), Q_2)$ 
  else
     $right(Q_1) \leftarrow \text{MELD}(right(Q_1), Q_2)$ 
  return  $Q_1$ 
```

- (a) Prove that for *any* heap-ordered binary trees  $Q_1$  and  $Q_2$  (not just those constructed by the operations listed above), the expected running time of  $\text{MELD}(Q_1, Q_2)$  is  $O(\log n)$ , where  $n = |Q_1| + |Q_2|$ . [Hint: How long is a random root-to-leaf path in an  $n$ -node binary tree if each left/right choice is made with equal probability?]
  - (b) Prove that  $\text{MELD}(Q_1, Q_2)$  runs in  $O(\log n)$  time with high probability.
  - (c) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and  $O(1)$  additional time. (This implies that every operation takes  $O(\log n)$  time with high probability.)
3. Prove that GUESSMINCUT returns the *second* smallest cut in its input graph with probability  $\Omega(1/n^2)$ . (The second smallest cut could be significantly larger than the minimum cut.)

4. A *heater* is a sort of dual treap, in which the priorities of the nodes are given by the user, but their search keys are random (specifically, independently and uniformly distributed in the unit interval  $[0, 1]$ ).

- (a) Prove that for any  $r$ , the node with the  $r$ th smallest priority has expected depth  $O(\log r)$ .
- (b) Prove that an  $n$ -node heater has depth  $O(\log n)$  with high probability.
- (c) Describe algorithms to perform the operations INSERT and DELETEMIN in a heater. What are the expected worst-case running times of your algorithms?

You may assume all priorities and keys are distinct. [Hint: Cite the relevant parts (but only the relevant parts!) of the treap analysis instead of repeating them.]

5. Let  $n$  be an arbitrary positive integer. Describe a set  $\mathcal{T}$  of binary search trees with the following properties:

- Every tree in  $\mathcal{T}$  has  $n$  nodes, which store the search keys  $1, 2, 3, \dots, n$ .
- For any integer  $k$ , if we choose a tree uniformly at random from  $\mathcal{T}$ , the expected depth of node  $k$  in that tree is  $O(\log n)$ .
- Every tree in  $\mathcal{T}$  has depth  $\Omega(\sqrt{n})$ .

(This is why we had to prove via Chernoff bounds that the maximum depth of an  $n$ -node treap is  $O(\log n)$  with high probability.)

★6. [Extra Credit] Recall that  $F_k$  denotes the  $k$ th Fibonacci number:  $F_0 = 0$ ,  $F_1 = 1$ , and  $F_k = F_{k-1} + F_{k-2}$  for all  $k \geq 2$ . Suppose we are building a hash table of size  $m = F_k$  using the hash function

$$h(x) = (F_{k-1} \cdot x) \bmod F_k$$

Prove that if the consecutive integers  $0, 1, 2, \dots, F_k - 1$  are inserted in order into an initially empty table, each integer is hashed into one of the largest contiguous empty intervals in the table. Among other things, this implies that there are no collisions.

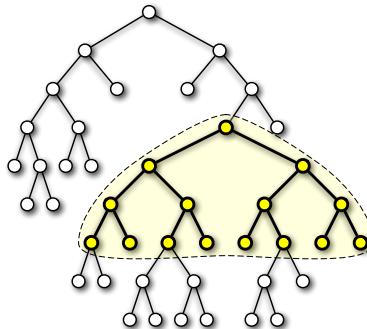
For example, when  $m = 13$ , the hash table is filled as follows.

0												
0									1			
0			2					1				
0			2					1			3	
0			2			4		1			3	
0	5		2			4		1			3	
0	5		2			4		1	6		3	
0	5		2	7		4		1	6		3	8
0	5		2	7		4	9	1	6		3	8
0	5	10	2	7		4	9	1	6		3	8
0	5	10	2	7	12	4	9	1	6	11	3	8
0	5	10	2	7	12	4	9	1	6	11	3	8

You have 90 minutes to answer four of these questions.  
**Write your answers in the separate answer booklet.**  
 You may take the question sheet with you when you leave.

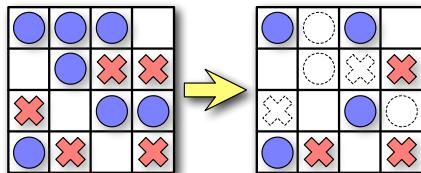
1. Recall that a binary tree is *complete* if every internal node has two children and every leaf has the same depth. An *internal subtree* of a binary tree is a connected subgraph, consisting of a node and some (possibly all or none) of its descendants.

Describe and analyze an algorithm that computes the depth of the *largest complete internal subtree* of a given  $n$ -node binary tree. For full credit, your algorithm should run in  $O(n)$  time.

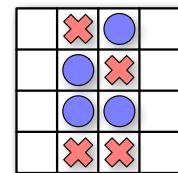


The largest complete internal subtree in this binary tree has depth 3.

2. Consider the following solitaire game. The puzzle consists of an  $n \times m$  grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) every row contains at least one stone, and (2) no column contains stones of both colors. For some initial configurations of stones, reaching this goal is impossible.



A solvable puzzle and one of its many solutions.



An unsolvable puzzle.

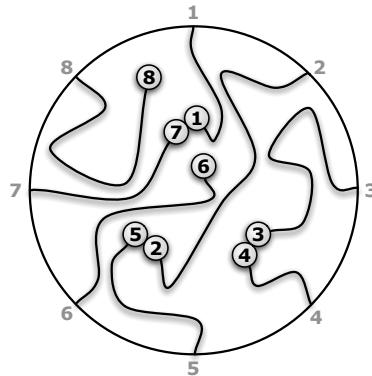
**Prove** that it is NP-hard to determine, given an initial configuration of red and blue stones, whether the puzzle can be solved.

3. Suppose you are given two sorted arrays  $A[1..n]$  and  $B[1..n]$  and an integer  $k$ . Describe and analyze an algorithm to find the  $k$ th largest element in the union of  $A$  and  $B$  in  $O(\log n)$  time. For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 21], \quad B[1..8] = [2, 4, 5, 8, 14, 17, 19, 20], \quad k = 10,$$

your algorithm should return 13. You can assume that the arrays contain no duplicates. [Hint: What can you learn from comparing one element of  $A$  to one element of  $B$ ?]

4. Every year, as part of its annual meeting, the Antarctic Snail Lovers of Upper Glacierville hold a Round Table Mating Race. Several high-quality breeding snails are placed at the edge of a round table. The snails are numbered in order around the table from 1 to  $n$ . During the race, each snail wanders around the table, leaving a trail of slime behind it. The snails have been specially trained never to fall off the edge of the table or to cross a slime trail, even their own. If two snails meet, they are declared a breeding pair, removed from the table, and whisked away to a romantic hole in the ground to make little baby snails. Note that some snails may never find a mate, even if the race goes on forever.



The end of a typical Antarctic SLUG race. Snails 6 and 8 never find mates.

The organizers must pay  $M[3, 4] + M[2, 5] + M[1, 7]$ .

For every pair of snails, the Antarctic SLUG race organizers have posted a monetary reward, to be paid to the owners if that pair of snails meets during the Mating Race. Specifically, there is a two-dimensional array  $M[1..n, 1..n]$  posted on the wall behind the Round Table, where  $M[i, j] = M[j, i]$  is the reward to be paid if snails  $i$  and  $j$  meet.

Describe and analyze an algorithm to compute the maximum total reward that the organizers could be forced to pay, given the array  $M$  as input.

5. SUBSETSUM and PARTITION are two closely-related NP-hard problems.

- SUBSETSUM: Given a set  $X$  of positive integers and an integer  $t$ , determine whether there is a subset of  $X$  whose elements sum to  $t$ .
- PARTITION: Given a set  $X$  of positive integers, determine whether  $X$  can be partitioned into two subsets whose elements sum to the same value.
  - (a) Describe a polynomial-time reduction from SUBSETSUM to PARTITION.
  - (b) Describe a polynomial-time reduction from PARTITION to SUBSETSUM.

Don't forget to **prove** that your reductions are correct.

You have 120 minutes to answer four of these questions.  
**Write your answers in the separate answer booklet.**  
 You may take the question sheet with you when you leave.

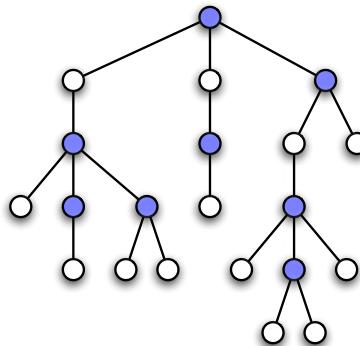
1. Consider the following algorithm for finding the smallest element in an unsorted array:

```
RANDOMMIN( $A[1..n]$ ):
   $min \leftarrow \infty$ 
  for  $i \leftarrow 1$  to  $n$  in random order
    if  $A[i] < min$ 
       $min \leftarrow A[i]$  (*)
  return  $min$ 
```

- (a) [1 pt] In the worst case, how many times does RANDOMMIN execute line (\*)?
- (b) [3 pts] What is the probability that line (\*) is executed during the last iteration of the for loop?
- (c) [6 pts] What is the *exact* expected number of executions of line (\*)? (A correct  $\Theta()$  bound is worth 4 points.)

2. Describe and analyze an efficient algorithm to find the size of the smallest vertex cover of a given tree. That is, given a tree  $T$ , your algorithm should find the size of the smallest subset  $C$  of the vertices, such that every edge in  $T$  has at least one endpoint in  $C$ .

The following hint may be helpful. Suppose  $C$  is a vertex cover that contains a leaf  $\ell$ . If we remove  $\ell$  from the cover and insert its parent, we get another vertex cover of the same size as  $C$ . Thus, there is a minimum vertex cover that includes none of the leaves of  $T$  (except when the tree has only one or two vertices).

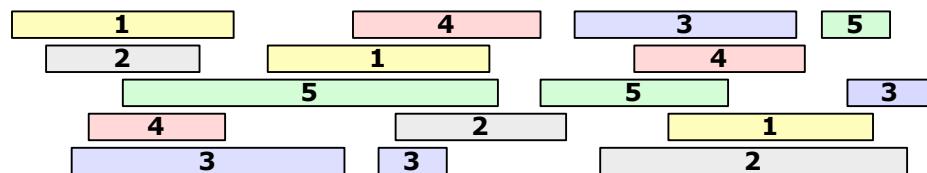


A tree whose smallest vertex cover has size 8.

3. A *dominating set* for a graph  $G$  is a subset  $D$  of the vertices, such that every vertex in  $G$  is either in  $D$  or has a neighbor in  $D$ . The MINDOMINATINGSET problem asks for the size of the smallest dominating set for a given graph.

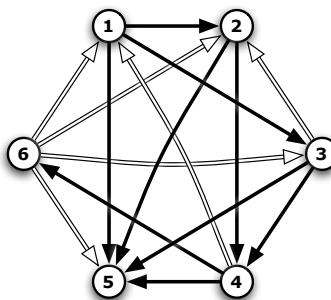
Recall the MINSETCOVER problem from lecture. The input consists of a *ground set*  $X$  and a collection of subsets  $S_1, S_2, \dots, S_k \subseteq X$ . The problem is to find the minimum number of subsets  $S_i$  that completely cover  $X$ . This problem is NP-hard, because it is a generalization of the vertex cover problem.

- (a) [7 pts] Describe a polynomial-time reduction from MINDOMINATINGSET to MINSET-COVER.
  - (b) [3 pts] Describe a polynomial-time  $O(\log n)$ -approximation algorithm for MINDOMINATINGSET. [Hint: There is a two-line solution.]
4. Let  $X$  be a set of  $n$  intervals on the real line. A *proper coloring* of  $X$  assigns a color to each interval, so that any two overlapping intervals are assigned different colors. Describe and analyze an efficient algorithm to compute the minimum number of colors needed to properly color  $X$ . Assume that your input consists of two arrays  $L[1..n]$  and  $R[1..n]$ , where  $L[i]$  and  $R[i]$  are the left and right endpoints of the  $i$ th interval. As usual, if you use a greedy algorithm, you must prove that it is correct.



A proper coloring of a set of intervals using five colors.

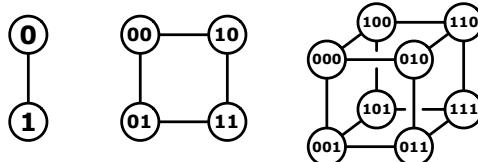
5. The *linear arrangement problem* asks, given an  $n$ -vertex directed graph as input, for an ordering  $v_1, v_2, \dots, v_n$  of the vertices that maximizes the number of *forward edges*: directed edges  $v_i \rightarrow v_j$  such that  $i < j$ . Describe and analyze an efficient 2-approximation algorithm for this problem.



A directed graph with six vertices with nine forward edges (black) and six backward edges (white)

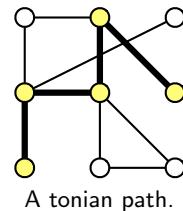
You have 180 minutes to answer six of these questions.  
**Write your answers in the separate answer booklet.**

1. The  $d$ -dimensional hypercube is the graph defined as follows. There are  $2^d$  vertices, each labeled with a different string of  $d$  bits. Two vertices are joined by an edge if and only if their labels differ in exactly one bit.



The 1-dimensional, 2-dimensional, and 3-dimensional hypercubes.

- (a) [8 pts] Recall that a Hamiltonian cycle is a closed walk that visits each vertex in a graph exactly once. **Prove** that for all  $d \geq 2$ , the  $d$ -dimensional hypercube has a Hamiltonian cycle.
- (b) [2 pts] Recall that an Eulerian circuit is a closed walk that traverses each edge in a graph exactly once. Which hypercubes have an Eulerian circuit? [Hint: This is very easy.]
2. The University of Southern North Dakota at Hoople has hired you to write an algorithm to schedule their final exams. Each semester, USNDH offers  $n$  different classes. There are  $r$  different rooms on campus and  $t$  different time slots in which exams can be offered. You are given two arrays  $E[1..n]$  and  $S[1..r]$ , where  $E[i]$  is the number of students enrolled in the  $i$ th class, and  $S[j]$  is the number of seats in the  $j$ th room. At most one final exam can be held in each room during each time slot. Class  $i$  can hold its final exam in room  $j$  only if  $E[i] < S[j]$ . Describe and analyze an efficient algorithm to assign a room and a time slot to each class (or report correctly that no such assignment is possible).
3. What is the *exact* expected number of leaves in an  $n$ -node treap? (The answer is obviously at most  $n$ , so no partial credit for writing " $O(n)$ ".) [Hint: What is the probability that the node with the  $k$ th largest key is a leaf?]
4. A *tonian path* in a graph  $G$  is a simple path in  $G$  that visits more than half of the vertices of  $G$ . (Intuitively, a tonian path is “most of a Hamiltonian path”.) **Prove** that it is NP-hard to determine whether or not a given graph contains a tonian path.



A tonian path.

5. A *palindrome* is a string that reads the same forwards and backwards, like x, pop, noon, redivider, or amanaplanacatahamayakayamahatacanalpanama. Any string can be broken into sequence of palindromes. For example, the string bubbasesabana ('Bubba sees a banana.') can be broken into palindromes in several different ways; for example,

$$\begin{aligned} &\text{bub} + \text{baseesab} + \text{anana} \\ &\text{b} + \text{u} + \text{bb} + \text{a} + \text{sees} + \text{aba} + \text{nan} + \text{a} \\ &\text{b} + \text{u} + \text{bb} + \text{a} + \text{sees} + \text{a} + \text{b} + \text{anana} \\ &\text{b} + \text{u} + \text{b} + \text{b} + \text{a} + \text{s} + \text{e} + \text{e} + \text{s} + \text{a} + \text{b} + \text{a} + \text{n} + \text{a} + \text{n} + \text{a} \end{aligned}$$

Describe and analyze an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string bubbasesabana, your algorithm would return the integer 3.

6. Consider the following modification of the 2-approximation algorithm for minimum vertex cover that we saw in class. The only real change is that we compute a set of edges instead of a set of vertices.

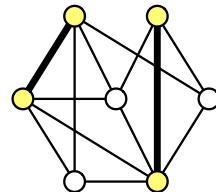
APPROXMINMAXMATCHING( $G$ ):

```

 $M \leftarrow \emptyset$ 
while  $G$  has at least one edge
   $(u, v) \leftarrow$  any edge in  $G$ 
   $G \leftarrow G \setminus \{u, v\}$ 
   $M \leftarrow M \cup \{(u, v)\}$ 
return  $M$ 

```

- (a) [2 pts] **Prove** that the output graph  $M$  is a *matching*—no pair of edges in  $M$  share a common vertex.
- (b) [2 pts] **Prove** that  $M$  is a *maximal matching*— $M$  is not a proper subgraph of another matching in  $G$ .
- (c) [6 pts] **Prove** that  $M$  contains at most twice as many edges as the *smallest maximal matching* in  $G$ .



The smallest maximal matching in a graph.

7. Recall that in the standard maximum-flow problem, the flow through an edge is limited by the capacity of that edge, but there is no limit on how much flow can pass through a vertex. Suppose each vertex  $v$  in our input graph has a capacity  $c(v)$  that limits the total flow through  $v$ , in addition to the usual edge capacities. Describe and analyze an efficient algorithm to compute the maximum  $(s, t)$ -flow with these additional constraints. [Hint: Reduce to the standard max-flow problem.]

# CS 573: Graduate Algorithms, Fall 2008

## Homework 0

Due in class at 12:30pm, Wednesday, September 3, 2008

Name:	
Net ID:	Alias:

I understand the course policies.

- 
- Each student must submit their own solutions for this homework. For all future homeworks, groups of up to three students may submit a single, common solution.
  - Neatly print your full name, your NetID, and an alias of your choice in the boxes above, and staple this page to the front of your homework solutions. We will list homework and exam grades on the course web site by alias.

Federal privacy law and university policy forbid us from publishing your grades, even anonymously, without your explicit written permission. *By providing an alias, you grant us permission to list your grades on the course web site. If you do not provide an alias, your grades will not be listed.* For privacy reasons, your alias should not resemble your name, your NetID, your university ID number, or (God forbid) your Social Security number.

- Please carefully read the course policies linked from the course web site. If you have *any* questions, please ask during lecture or office hours, or post your question to the course newsgroup. Once you understand the policies, please check the box at the top of this page. In particular:
    - You may use any source at your disposal—paper, electronic, or human—but you **must** write your solutions in your own words, and you **must** cite every source that you use.
    - Unless explicitly stated otherwise, **every** homework problem requires a proof.
    - Answering “I don’t know” to any homework or exam problem is worth 25% partial credit.
    - Algorithms or proofs containing phrases like “and so on” or “repeat this for all  $n$ ”, instead of an explicit loop, recursion, or induction, will receive 0 points.
  - This homework tests your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, discrete probability, graphs, and most importantly, induction—to help you identify gaps in your background knowledge. **You are responsible for filling those gaps.** The early chapters of any algorithms textbook should be sufficient review, but you may also want consult your favorite discrete mathematics and data structures textbooks. If you need help, please ask in office hours and/or on the course newsgroup.
-

1. (a) [5 pts] Solve the following recurrences. State tight asymptotic bounds for each function in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ . Assume reasonable but nontrivial base cases. If your solution requires a particular base case, say so.

$$A(n) = 4A(n/8) + \sqrt{n}$$

$$B(n) = B(n/3) + 2B(n/4) + B(n/6) + n$$

$$C(n) = 6C(n-1) - 9C(n-2)$$

$$D(n) = \max_{n/3 < k < 2n/3} (D(k) + D(n-k) + n)$$

$$E(n) = (E(\sqrt{n}))^2 \cdot n$$

- (b) [5 pts] Sort the functions in the box from asymptotically smallest to asymptotically largest, indicating ties if there are any. **Do not turn in proofs**—just a sorted list of 16 functions—but you should do them anyway, just for practice. We use the notation  $\lg n = \log_2 n$ .

$n$	$\lg n$	$\sqrt{n}$	$3^n$
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$3^{\sqrt{n}}$	$\sqrt{3^n}$
$3^{\lg n}$	$\lg(3^n)$	$3^{\lg \sqrt{n}}$	$3^{\sqrt{\lg n}}$
$\sqrt{3^{\lg n}}$	$\lg(3^{\sqrt{n}})$	$\lg \sqrt{3^n}$	$\sqrt{\lg(3^n)}$

2. Describe and analyze a data structure that stores set of  $n$  records, each with a numerical *key* and a numerical *priority*, such that the following operation can be performed quickly:

- $\text{RANGE TOP}(a, z)$  : return the highest-priority record whose key is between  $a$  and  $z$ .

For example, if the (key, priority) pairs are

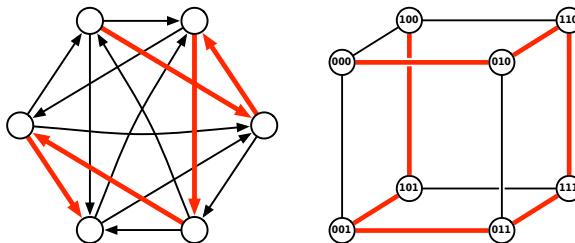
$$(3, 1), (4, 9), (9, 2), (6, 3), (5, 8), (7, 5), (1, 10), (0, 7),$$

then  $\text{RANGE TOP}(2, 8)$  would return the record with key 4 and priority 9 (the second in the list).

Analyze both the size of your data structure and the running time of your  $\text{RANGE TOP}$  algorithm. For full credit, your space and time bounds must both be as small as possible. You may assume that no two records have equal keys or equal priorities, and that no record has  $a$  or  $z$  as its key. *[Hint: How would you compute the number of keys between  $a$  and  $z$ ? How would you solve the problem if you knew that  $a$  is always  $-\infty$ ?]*

3. A *Hamiltonian path* in  $G$  is a path that visits every vertex of  $G$  exactly once. In this problem, you are asked to prove that two classes of graphs always contain a Hamiltonian path.

- (a) [5 pts] A *tournament* is a directed graph with exactly one edge between each pair of vertices. (Think of the nodes in a round-robin tournament, where edges represent games, and each edge points from the loser to the winner.) Prove that every tournament contains a *directed* Hamiltonian path.
- (b) [5 pts] Let  $d$  be an arbitrary non-negative integer. The  $d$ -dimensional *hypercube* is the graph defined as follows. There are  $2^d$  vertices, each labeled with a different string of  $d$  bits. Two vertices are joined by an edge if and only if their labels differ in exactly one bit. Prove that the  $d$ -dimensional hypercube contains a Hamiltonian path.



Hamiltonian paths in a 6-node tournament and a 3-dimensional hypercube.

4. Penn and Teller agree to play the following game. Penn shuffles a standard deck<sup>1</sup> of playing cards so that every permutation is equally likely. Then Teller draws cards from the deck, one at a time without replacement, until he draws the three of clubs ( $3\clubsuit$ ), at which point the remaining undrawn cards instantly burst into flames.

The first time Teller draws a card from the deck, he gives it to Penn. From then on, until the game ends, whenever Teller draws a card whose value is smaller than the last card he gave to Penn, he gives the new card to Penn.<sup>2</sup> To make the rules unambiguous, they agree beforehand that  $A = 1$ ,  $J = 11$ ,  $Q = 12$ , and  $K = 13$ .

- (a) What is the expected number of cards that Teller draws?
- (b) What is the expected *maximum* value among the cards Teller gives to Penn?
- (c) What is the expected *minimum* value among the cards Teller gives to Penn?
- (d) What is the expected number of cards that Teller gives to Penn?

Full credit will be given only for *exact* answers (with correct proofs, of course). [Hint: Let  $13 = n$ .]

---

<sup>1</sup>In a standard deck of playing cards, each card has a *value* in the set  $\{A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K\}$  and a *suit* in the set  $\{\spadesuit, \heartsuit, \clubsuit, \diamondsuit\}$ ; each of the 52 possible suit-value pairs appears in the deck exactly once. Actually, to make the game more interesting, Penn and Teller normally use razor-sharp ninja throwing cards.

<sup>2</sup>Specifically, he hurls them from the opposite side of the stage directly into the back of Penn's right hand. Ouch!

5. (a) The **Fibonacci numbers**  $F_n$  are defined by the recurrence  $F_n = F_{n-1} + F_{n-2}$ , with base cases  $F_0 = 0$  and  $F_1 = 1$ . Here are the first several Fibonacci numbers:

$F_0$	$F_1$	$F_2$	$F_3$	$F_4$	$F_5$	$F_6$	$F_7$	$F_8$	$F_9$	$F_{10}$
0	1	1	2	3	5	8	13	21	34	55

Prove that any non-negative integer can be written as the sum of distinct, non-consecutive Fibonacci numbers. That is, if the Fibonacci number  $F_i$  appears in the sum, it appears exactly once, and its neighbors  $F_{i-1}$  and  $F_{i+1}$  do not appear at all. For example:

$$17 = F_7 + F_4 + F_2, \quad 42 = F_9 + F_6, \quad 54 = F_9 + F_7 + F_5 + F_3 + F_1.$$

- (b) The Fibonacci sequence can be extended backward to negative indices by rearranging the defining recurrence:  $F_n = F_{n+2} - F_{n+1}$ . Here are the first several negative-index Fibonacci numbers:

$F_{-10}$	$F_{-9}$	$F_{-8}$	$F_{-7}$	$F_{-6}$	$F_{-5}$	$F_{-4}$	$F_{-3}$	$F_{-2}$	$F_{-1}$
-55	34	-21	13	-8	5	-3	2	-1	1

Prove that  $F_{-n} = -F_n$  if and only if  $n$  is even.

- (c) Prove that *any* integer—positive, negative, or zero—can be written as the sum of distinct, non-consecutive Fibonacci numbers *with negative indices*. For example:

$$17 = F_{-7} + F_{-5} + F_{-2}, \quad -42 = F_{-10} + F_{-7}, \quad 54 = F_{-9} + F_{-7} + F_{-5} + F_{-3} + F_{-1}.$$

[Hint: Zero is both non-negative and even. Don't use weak induction!]

# CS 573: Graduate Algorithms, Fall 2008

## Homework 1

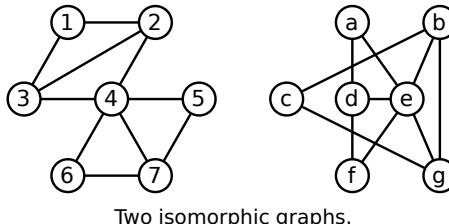
Due at 11:59:59pm, Wednesday, September 17, 2008

---

For this and all future homeworks, groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and alias (if any) of every group member on the first page of your submission.

---

1. Two graphs are said to be **isomorphic** if one can be transformed into the other just by relabeling the vertices. For example, the graphs shown below are isomorphic; the left graph can be transformed into the right graph by the relabeling  $(1, 2, 3, 4, 5, 6, 7) \mapsto (c, g, b, e, a, f, d)$ .



Consider the following related decision problems:

- **GRAPHISOMORPHISM:** Given two graphs  $G$  and  $H$ , determine whether  $G$  and  $H$  are isomorphic.
  - **EVENGRAPHISOMORPHISM:** Given two graphs  $G$  and  $H$ , such that every vertex in  $G$  and  $H$  has even degree, determine whether  $G$  and  $H$  are isomorphic.
  - **SUBGRAPHISOMORPHISM:** Given two graphs  $G$  and  $H$ , determine whether  $G$  is isomorphic to a subgraph of  $H$ .
- (a) Describe a polynomial-time reduction from **EVENGRAPHISOMORPHISM** to **GRAPHISOMORPHISM**.
  - (b) Describe a polynomial-time reduction from **GRAPHISOMORPHISM** to **EVENGRAPHISOMORPHISM**.
  - (c) Describe a polynomial-time reduction from **GRAPHISOMORPHISM** to **SUBGRAPHISOMORPHISM**.
  - (d) Prove that **SUBGRAPHISOMORPHISM** is NP-complete.
  - (e) What can you conclude about the NP-hardness of **GRAPHISOMORPHISM**? Justify your answer.

[Hint: These are all easy!]

2. (a) A *tonian path* in a graph  $G$  is a path that goes through at least half of the vertices of  $G$ . Show that determining whether a graph has a tonian path is NP-complete.  
(b) A *tonian cycle* in a graph  $G$  is a cycle that goes through at least half of the vertices of  $G$ . Show that determining whether a graph has a tonian cycle is NP-complete. [Hint: Use part (a).]
3. The following variant of 3SAT is called either EXACT3SAT or 1IN3SAT, depending on who you ask.

Given a boolean formula in conjunctive normal form with 3 literals per clause, is there an assignment that makes **exactly one** literal in each clause TRUE?

Prove that this problem is NP-complete.

4. Suppose you are given a magic black box that can solve the MAXCLIQUE problem *in polynomial time*. That is, given an arbitrary graph  $G$  as input, the magic black box computes the number of vertices in the largest complete subgraph of  $G$ . Describe and analyze a **polynomial-time** algorithm that computes, given an arbitrary graph  $G$ , a complete subgraph of  $G$  of maximum size, using this magic black box as a subroutine.
  5. A boolean formula in *exclusive-or conjunctive normal form* (XCNF) is a conjunction (AND) of several *clauses*, each of which is the *exclusive-or* of several literals. The XCNF-SAT problem asks whether a given XCNF boolean formula is satisfiable. Either describe a polynomial-time algorithm for XCNF-SAT or prove that it is NP-complete.
- \*6. [Extra credit] Describe and analyze an algorithm to solve 3SAT in  $O(\phi^n \text{poly}(n))$  time, where  $\phi = (1 + \sqrt{5})/2 \approx 1.618034$ . [Hint: Prove that in each recursive call, either you have just eliminated a pure literal, or the formula has a clause with at most two literals. What recurrence leads to this running time?]

---

<sup>0</sup>In class, I asserted that Gaussian elimination was probably discovered by Gauss, in violation of Stigler's Law of Eponymy. In fact, a method very similar to Gaussian elimination appears in the Chinese treatise *Nine Chapters on the Mathematical Art*, believed to have been finalized before 100AD, although some material may predate emperor Qin Shi Huang's infamous 'burning of the books and burial of the scholars' in 213BC. The great Chinese mathematician Liu Hui, in his 3rd-century commentary on *Nine Chapters*, compares two variants of the method and counts the number of arithmetic operations used by each, with the explicit goal of finding the more efficient method. This is arguably the earliest recorded analysis of any algorithm.

# CS 573: Graduate Algorithms, Fall 2008

## Homework 2

Due at 11:59:59pm, Wednesday, October 1, 2008

- 
- For this and all future homeworks, groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and alias (if any) of every group member on the first page of your submission.
  - We will use the following point breakdown to grade dynamic programming algorithms: 60% for a correct recurrence (including base cases), 20% for correct running time analysis of the memoized recurrence, 10% for correctly transforming the memoized recursion into an iterative algorithm.
  - A greedy algorithm *must* be accompanied by a proof of correctness in order to receive *any* credit.
- 

1. (a) Let  $X[1..m]$  and  $Y[1..n]$  be two arbitrary arrays of numbers. A **common supersequence** of  $X$  and  $Y$  is another sequence that contains both  $X$  and  $Y$  as subsequences. Describe and analyze an efficient algorithm to compute the function  $scs(X, Y)$ , which gives the length of the shortest common supersequence of  $X$  and  $Y$ .  
(b) Call a sequence  $X[1..n]$  of numbers **oscillating** if  $X[i] < X[i + 1]$  for all even  $i$ , and  $X[i] > X[i + 1]$  for all odd  $i$ . Describe and analyze an efficient algorithm to compute the function  $los(X)$ , which gives the length of the longest oscillating subsequence of an arbitrary array  $X$  of integers.  
(c) Call a sequence  $X[1..n]$  of numbers **accelerating** if  $2 \cdot X[i] < X[i - 1] + X[i + 1]$  for all  $i$ . Describe and analyze an efficient algorithm to compute the function  $lxs(X)$ , which gives the length of the longest accelerating subsequence of an arbitrary array  $X$  of integers.
2. A **palindrome** is a string that reads the same forwards and backwards, like x, pop, noon, redivider, or amanaplanacatahamayakayamahatacanalpanama. Any string can be broken into sequence of palindromes. For example, the string bubbasesabana ('Bubba sees a banana.') can be broken into palindromes in several different ways; for example:

$$\begin{aligned} &\text{bub} + \text{baseesab} + \text{anana} \\ &\text{b} + \text{u} + \text{bb} + \text{a} + \text{sees} + \text{aba} + \text{nan} + \text{a} \\ &\text{b} + \text{u} + \text{bb} + \text{a} + \text{sees} + \text{a} + \text{b} + \text{anana} \\ &\text{b} + \text{u} + \text{b} + \text{b} + \text{a} + \text{s} + \text{e} + \text{e} + \text{s} + \text{a} + \text{b} + \text{a} + \text{n} + \text{a} + \text{n} + \text{a} \end{aligned}$$

Describe and analyze an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string bubbasesabana, your algorithm would return the integer 3.

3. Describe and analyze an algorithm to solve the traveling salesman problem in  $O(2^n \text{poly}(n))$  time. Given an undirected  $n$ -vertex graph  $G$  with weighted edges, your algorithm should return the weight of the lightest Hamiltonian cycle in  $G$ , or  $\infty$  if  $G$  has no Hamiltonian cycles. [Hint: The obvious recursive algorithm takes  $O(n!)$  time.]

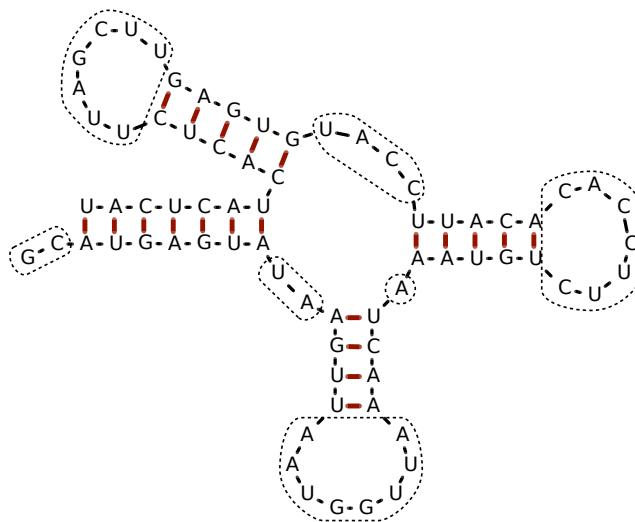
4. Ribonucleic acid (RNA) molecules are long chains of millions of nucleotides or *bases* of four different types: adenine (A), cytosine (C), guanine (G), and uracil (U). The *sequence* of an RNA molecule is a string  $b[1..n]$ , where each character  $b[i] \in \{A, C, G, U\}$  corresponds to a base. In addition to the chemical bonds between adjacent bases in the sequence, hydrogen bonds can form between certain pairs of bases. The set of bonded base pairs is called the *secondary structure* of the RNA molecule.

We say that two base pairs  $(i, j)$  and  $(i', j')$  with  $i < j$  and  $i' < j'$  **overlap** if  $i < i' < j < j'$  or  $i' < i < j' < j$ . In practice, most base pairs are non-overlapping. Overlapping base pairs create so-called *pseudoknots* in the secondary structure, which are essential for some RNA functions, but are more difficult to predict.

Suppose we want to predict the best possible secondary structure for a given RNA sequence. We will adopt a drastically simplified model of secondary structure:

- Each base can be paired with at most one other base.
- Only A-U pairs and C-G pairs can bond.
- Pairs of the form  $(i, i + 1)$  and  $(i, i + 2)$  cannot bond.
- Overlapping base pairs cannot bond.

The last restriction allows us to visualize RNA secondary structure as a sort of fat tree.

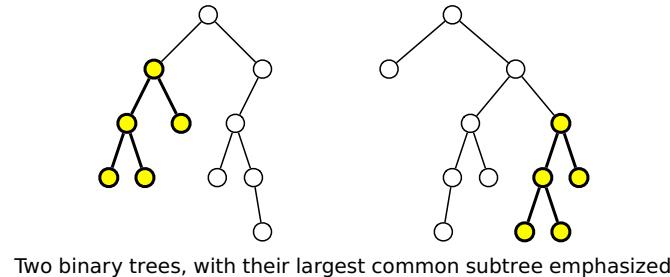


Example RNA secondary structure with 21 base pairs, indicated by heavy red lines.  
Gaps are indicated by dotted curves. This structure has score  $2^2 + 2^2 + 8^2 + 1^2 + 7^2 + 4^2 + 7^2 = 187$

- Describe and analyze an algorithm that computes the maximum possible *number* of base pairs in a secondary structure for a given RNA sequence.
- A *gap* in a secondary structure is a maximal substring of unpaired bases. Large gaps lead to chemical instabilities, so secondary structures with smaller gaps are more likely. To account for this preference, let's define the *score* of a secondary structure to be the sum of the *squares* of the gap lengths.<sup>1</sup> Describe and analyze an algorithm that computes the minimum possible score of a secondary structure for a given RNA sequence.

<sup>1</sup>This score function has absolutely no connection to reality; I just made it up. Real RNA structure prediction requires much more complicated scoring functions.

5. A *subtree* of a (rooted, ordered) binary tree  $T$  consists of a node and all its descendants. Design and analyze an efficient algorithm to compute the *largest common subtree* of two given binary trees  $T_1$  and  $T_2$ ; this is the largest subtree of  $T_1$  that is isomorphic to a subtree in  $T_2$ . The contents of the nodes are irrelevant; we are only interested in matching the underlying combinatorial structure.



- \*6. [Extra credit] Let  $D[1..n]$  be an array of digits, each an integer between 0 and 9. A *digital subsequence* of  $D$  is an sequence of positive integers composed in the usual way from disjoint substrings of  $D$ . For example, 3, 4, 5, 6, 23, 38, 62, 64, 83, 279 is an increasing digital subsequence of the first several digits of  $\pi$ :

3, 1, 4, 1, 5, 9, 6, 2, 3, 4, 3, 8, 4, 6, 2, 6, 4, 3, 3, 8, 3, 2, 7, 9

The *length* of a digital subsequence is the number of integers it contains, *not* the number of digits; the previous example has length 10.

Describe and analyze an efficient algorithm to compute the longest increasing digital subsequence of  $D$ . [Hint: Be careful about your computational assumptions. How long does it take to compare two  $k$ -digit numbers?]

# CS 573: Graduate Algorithms, Fall 2008

## Homework 3

Due at 11:59:59pm, Wednesday, October 22, 2008

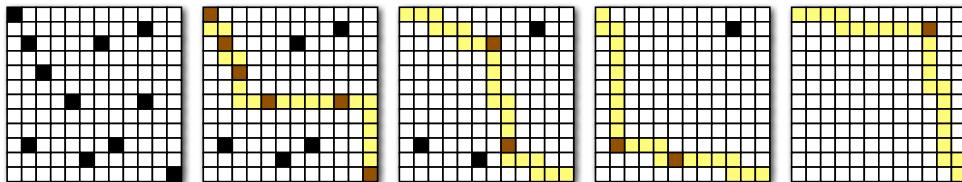
- 
- Groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and the HW0 alias (if any) of every group member on the first page of your submission.
- 

- Consider an  $n \times n$  grid, some of whose cells are marked. A *monotone* path through the grid starts at the top-left cell, moves only right or down at each step, and ends at the bottom-right cell. We want to compute the minimum number of monotone paths that cover all marked cells. The input to our problem is an array  $M[1..n, 1..n]$  of booleans, where  $M[i, j] = \text{TRUE}$  if and only if cell  $(i, j)$  is marked.

One of your friends suggests the following greedy strategy:

- Find (somehow) one “good” path  $\pi$  that covers the maximum number of marked cells.
- Unmark the cells covered by  $\pi$ .
- If any cells are still marked, recursively cover them.

Does this greedy strategy always compute an optimal solution? If yes, give a proof. If no, give a counterexample.



Greedily covering the marked cells in a grid with four monotone paths.

- Let  $X$  be a set of  $n$  intervals on the real line. A subset of intervals  $Y \subseteq X$  is called a *tiling path* if the intervals in  $Y$  cover the intervals in  $X$ , that is, any real value that is contained in some interval in  $X$  is also contained in some interval in  $Y$ . The *size* of a tiling path is just the number of intervals.

Describe and analyze an algorithm to compute the smallest tiling path of  $X$  as quickly as possible. Assume that your input consists of two arrays  $X_L[1..n]$  and  $X_R[1..n]$ , representing the left and right endpoints of the intervals in  $X$ . If you use a greedy algorithm, you must prove that it is correct.



A set of intervals. The seven shaded intervals form a tiling path.

3. Given a graph  $G$  with edge weights and an integer  $k$ , suppose we wish to partition the vertices of  $G$  into  $k$  subsets  $S_1, S_2, \dots, S_k$  so that the sum of the weights of the edges that cross the partition (*i.e.*, that have endpoints in different subsets) is as large as possible.
- Describe an efficient  $(1 - 1/k)$ -approximation algorithm for this problem. [*Hint: Solve the special case  $k = 2$  first.*]
  - Now suppose we wish to minimize the sum of the weights of edges that do *not* cross the partition. What approximation ratio does your algorithm from part (a) achieve for this new problem? Justify your answer.
4. Consider the following heuristic for constructing a vertex cover of a connected graph  $G$ : ***Return the set of all non-leaf nodes of any depth-first spanning tree.*** (Recall that a depth-first spanning tree is a *rooted* tree; the root is not considered a leaf, even if it has only one neighbor in the tree.)
- Prove that this heuristic returns a vertex cover of  $G$ .
  - Prove that this heuristic returns a 2-approximation to the minimum vertex cover of  $G$ .
  - Prove that for any  $\varepsilon > 0$ , there is a graph for which this heuristic returns a vertex cover of size at least  $(2 - \varepsilon) \cdot OPT$ .
5. Consider the following greedy approximation algorithm to find a vertex cover in a graph:

```

GREEDYVERTEXCOVER( $G$ ):
   $C \leftarrow \emptyset$ 
  while  $G$  has at least one edge
     $v \leftarrow$  vertex in  $G$  with maximum degree
     $G \leftarrow G \setminus v$ 
     $C \leftarrow C \cup v$ 
  return  $C$ 

```

In class we proved that the approximation ratio of this algorithm is  $O(\log n)$ ; your task is to prove a matching lower bound. Specifically, for any positive integer  $n$ , describe an  $n$ -vertex graph  $G$  such that  $\text{GREEDYVERTEXCOVER}(G)$  returns a vertex cover that is  $\Omega(\log n)$  times larger than optimal. [*Hint:  $H_n = \Omega(\log n)$ .*]

- \*6. [*Extra credit*] Consider the greedy algorithm for metric TSP: Start at an arbitrary vertex  $u$ , and at each step, travel to the closest unvisited vertex.
- Prove that this greedy algorithm is an  $O(\log n)$ -approximation algorithm, where  $n$  is the number of vertices. [*Hint: Show that the  $k$ th least expensive edge in the tour output by the greedy algorithm has weight at most  $OPT/(n - k + 1)$ ; try  $k = 1$  and  $k = 2$  first.*]
  - Prove that the greedy algorithm for metric TSP is no better than an  $O(\log n)$ -approximation. That is, describe an infinite family of weighted graphs that satisfy the triangle inequality, such that the greedy algorithm returns a cycle whose length is  $\Omega(\log n)$  times the optimal TSP tour.

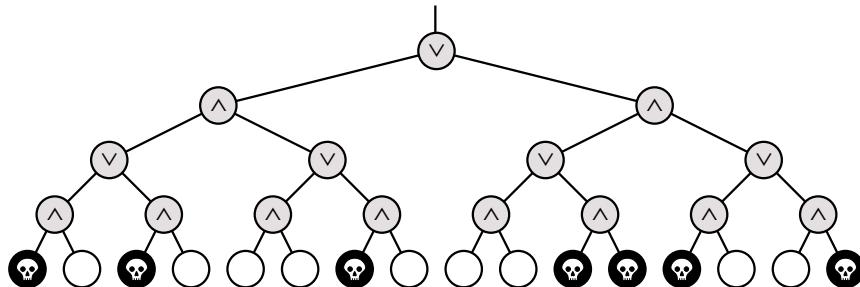
# CS 573: Graduate Algorithms, Fall 2008

## Homework 4

Due at 11:59:59pm, Wednesday, October 31, 2008

- 
- Groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and the HW0 alias (if any) of every group member on the first page of your submission.
  - Unless a problem explicitly states otherwise, you can assume the existence of a function  $\text{RANDOM}(k)$ , which returns an integer uniformly distributed in the range  $\{1, 2, \dots, k\}$  in  $O(1)$  time; the argument  $k$  must be a positive integer. For example,  $\text{RANDOM}(2)$  simulates a fair coin flip, and  $\text{RANDOM}(1)$  always returns 1.
- 

1. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with  $4^n$  leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after  $2n$  moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.



You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are AND gates. Each gate gets its input from its children and passes its output to its parent. White and black leaves stand represent TRUE and FALSE inputs, respectively. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

- (a) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy!]*
- (b) Unfortunately, Death won't let you even look at every node in the tree. Describe and analyze a randomized algorithm that determines whether you can win in  $O(3^n)$  expected time. *[Hint: Consider the case  $n = 1$ .]*
- (c) **[Extra credit]** Describe and analyze a randomized algorithm that determines whether you can win in  $O(c^n)$  expected time, for some constant  $c < 3$ . *[Hint: You may not need to change your algorithm at all.]*

2. Consider the following randomized algorithm for choosing the largest bolt. Draw a bolt uniformly at random from the set of  $n$  bolts, and draw a nut uniformly at random from the set of  $n$  nuts. If the bolt is smaller than the nut, discard the bolt, draw a new bolt uniformly at random from the unchosen bolts, and repeat. Otherwise, discard the nut, draw a new nut uniformly at random from the unchosen nuts, and repeat. Stop either when every nut has been discarded, or every bolt except the one in your hand has been discarded.

What is the *exact* expected number of nut-bolt tests performed by this algorithm? Prove your answer is correct. [Hint: What is the expected number of unchosen nuts and bolts when the algorithm terminates?]

3. (a) Prove that the expected number of proper descendants of any node in a treap is *exactly* equal to the expected depth of that node.
- (b) Why doesn't the Chernoff-bound argument for depth imply that, with high probability, *every* node in a treap has  $O(\log n)$  descendants? The conclusion is obviously bogus—every  $n$ -node treap has one node with exactly  $n$  descendants!—but what is the flaw in the argument?
- (c) What is the expected number of leaves in an  $n$ -node treap? [Hint: What is the probability that in an  $n$ -node treap, the node with  $k$ th smallest search key is a leaf?]
4. The following randomized algorithm, sometimes called “one-armed quicksort”, selects the  $r$ th smallest element in an unsorted array  $A[1..n]$ . For example, to find the smallest element, you would call  $\text{RANDOMSELECT}(A, 1)$ ; to find the median element, you would call  $\text{RANDOMSELECT}(A, \lfloor n/2 \rfloor)$ . The subroutine  $\text{PARTITION}(A[1..n], p)$  splits the array into three parts by comparing the pivot element  $A[p]$  to every other element of the array, using  $n - 1$  comparisons altogether, and returns the new index of the pivot element.

```
RANDOMSELECT( $A[1..n]$ ,  $r$ ) :
   $k \leftarrow \text{PARTITION}(A[1..n], \text{RANDOM}(n))$ 
  if  $r < k$ 
    return RANDOMSELECT( $A[1..k - 1]$ ,  $r$ )
  else if  $r > k$ 
    return RANDOMSELECT( $A[k + 1..n]$ ,  $r - k$ )
  else
    return  $A[k]$ 
```

- (a) State a recurrence for the expected running time of  $\text{RANDOMSELECT}$ , as a function of  $n$  and  $r$ .
- (b) What is the *exact* probability that  $\text{RANDOMSELECT}$  compares the  $i$ th smallest and  $j$ th smallest elements in the input array? The correct answer is a simple function of  $i$ ,  $j$ , and  $r$ . [Hint: Check your answer by trying a few small examples.]
- (c) Show that for any  $n$  and  $r$ , the expected running time of  $\text{RANDOMSELECT}$  is  $\Theta(n)$ . You can use either the recurrence from part (a) or the probabilities from part (b).
- \*(d) [*Extra Credit*] Find the *exact* expected number of comparisons executed by  $\text{RANDOMSELECT}$ , as a function of  $n$  and  $r$ .

5. A *meldable priority queue* stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:

- **MAKEQUEUE**: Return a new priority queue containing the empty set.
- **FINDMIN( $Q$ )**: Return the smallest element of  $Q$  (if any).
- **DELETEMIN( $Q$ )**: Remove the smallest element in  $Q$  (if any).
- **INSERT( $Q, x$ )**: Insert element  $x$  into  $Q$ , if it is not already there.
- **DECREASEKEY( $Q, x, y$ )**: Replace an element  $x \in Q$  with a smaller key  $y$ . (If  $y > x$ , the operation fails.) The input is a pointer directly to the node in  $Q$  containing  $x$ .
- **DELETE( $Q, x$ )**: Delete the element  $x \in Q$ . The input is a pointer directly to the node in  $Q$  containing  $x$ .
- **MELD( $Q_1, Q_2$ )**: Return a new priority queue containing all the elements of  $Q_1$  and  $Q_2$ ; this operation destroys  $Q_1$  and  $Q_2$ .

A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm:

```

MELD( $Q_1, Q_2$ ):
    if  $Q_1$  is empty return  $Q_2$ 
    if  $Q_2$  is empty return  $Q_1$ 
    if  $key(Q_1) > key(Q_2)$ 
        swap  $Q_1 \leftrightarrow Q_2$ 
    with probability 1/2
         $left(Q_1) \leftarrow \text{MELD}(left(Q_1), Q_2)$ 
    else
         $right(Q_1) \leftarrow \text{MELD}(right(Q_1), Q_2)$ 
    return  $Q_1$ 
```

- (a) Prove that for *any* heap-ordered binary trees  $Q_1$  and  $Q_2$  (*not* just those constructed by the operations listed above), the expected running time of  $\text{MELD}(Q_1, Q_2)$  is  $O(\log n)$ , where  $n$  is the total number of nodes in both trees. [Hint: How long is a random root-to-leaf path in an  $n$ -node binary tree if each left/right choice is made with equal probability?]
- (b) Prove that  $\text{MELD}(Q_1, Q_2)$  runs in  $O(\log n)$  time with high probability. [Hint: You don't need Chernoff bounds, but you might use the identity  $\binom{ck}{k} \leq (ce)^k$ .]
- (c) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and  $O(1)$  additional time. (This implies that every operation takes  $O(\log n)$  time with high probability.)

\*6. [*Extra credit*] In the usual theoretical presentation of treaps, the priorities are random real numbers chosen uniformly from the interval  $[0, 1]$ , but in practice, computers only have access to random *bits*. This problem asks you to analyze a modification of treaps that takes this limitation into account.

Suppose the priority of a node  $v$  is abstractly represented as an infinite sequence  $\pi_v[1.. \infty]$  of random bits, which is interpreted as the rational number

$$\text{priority}(v) = \sum_{i=1}^{\infty} \pi_v[i] \cdot 2^{-i}.$$

However, only a finite number  $\ell_v$  of these bits are actually known at any given time. When a node  $v$  is first created, *none* of the priority bits are known:  $\ell_v = 0$ . We generate (or ‘reveal’) new random bits only when they are necessary to compare priorities. The following algorithm compares the priorities of any two nodes in  $O(1)$  expected time:

```
LARGERPRIORITY( $v, w$ ):  
for  $i \leftarrow 1$  to  $\infty$   
  if  $i > \ell_v$   
     $\ell_v \leftarrow i$ ;  $\pi_v[i] \leftarrow \text{RANDOMBIT}$   
  if  $i > \ell_w$   
     $\ell_w \leftarrow i$ ;  $\pi_w[i] \leftarrow \text{RANDOMBIT}$   
  if  $\pi_v[i] > \pi_w[i]$   
    return  $v$   
  else if  $\pi_v[i] < \pi_w[i]$   
    return  $w$ 
```

Suppose we insert  $n$  items one at a time into an initially empty treap. Let  $L = \sum_v \ell_v$  denote the total number of random bits generated by calls to LARGERPRIORITY during these insertions.

- (a) Prove that  $E[L] = \Theta(n)$ .
- (b) Prove that  $E[\ell_v] = \Theta(1)$  for any node  $v$ . [Hint: This is equivalent to part (a). Why?]
- (c) Prove that  $E[\ell_{\text{root}}] = \Theta(\log n)$ . [Hint: Why doesn’t this contradict part (b)?]

# CS 573: Graduate Algorithms, Fall 2008

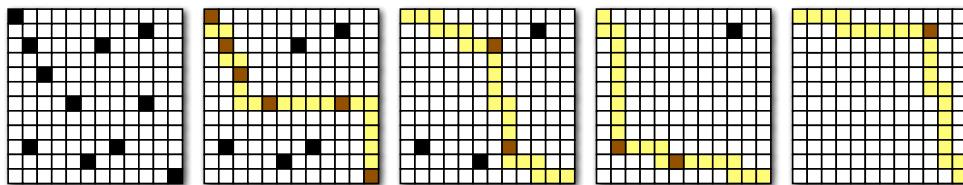
## Homework 5

Due at 11:59:59pm, Wednesday, November 19, 2008

- Groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name, NetID, and the HW0 alias (if any) of every group member on the first page of your submission.

- Recall the following problem from Homework 3: You are given an  $n \times n$  grid, some of whose cells are marked; the grid is represented by an array  $M[1..n, 1..n]$  of booleans, where  $M[i, j] = \text{TRUE}$  if and only if cell  $(i, j)$  is marked. A *monotone* path through the grid starts at the top-left cell, moves only right or down at each step, and ends at the bottom-right cell.

Describe and analyze an efficient algorithm to compute the smallest set of monotone paths that covers every marked cell.



Greedyly covering the marked cells in a grid with four monotone paths.

- Suppose we are given a directed graph  $G = (V, E)$ , two vertices  $s$  and  $t$ , and a capacity function  $c: V \rightarrow \mathbb{R}^+$ . A flow  $f$  is *feasible* if the total flow into every vertex  $v$  is at most  $c(v)$ :

$$\sum_u f(u \rightarrow v) \leq c(v) \quad \text{for every vertex } v.$$

Describe and analyze an efficient algorithm to compute a feasible flow of maximum value.

- Suppose we are given an array  $A[1..m][1..n]$  of non-negative real numbers. We want to *round*  $A$  to an integer matrix, by replacing each entry  $x$  in  $A$  with either  $\lfloor x \rfloor$  or  $\lceil x \rceil$ , without changing the sum of entries in any row or column of  $A$ . For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

Describe an efficient algorithm that either rounds  $A$  in this fashion, or reports correctly that no such rounding is possible.

4. *Ad-hoc networks* are made up of cheap, low-powered wireless devices. In principle<sup>1</sup>, these networks can be used on battlefields, in regions that have recently suffered from natural disasters, and in other situations where people might want to monitor conditions in hard-to-reach areas. The idea is that a large collection of cheap, simple devices could be dropped into the area from an airplane (for instance), and then they would somehow automatically configure themselves into an efficiently functioning wireless network.

The devices can communicate only within a limited range. We assume all the devices are identical; there is a distance  $D$  such that two devices can communicate if and only if the distance between them is at most  $D$ .

We would like our ad-hoc network to be reliable, but because the devices are cheap and low-powered, they frequently fail. If a device detects that it is likely to fail, it should transmit the information it has to some other *backup* device within its communication range. To improve reliability, we require each device  $x$  to have  $k$  potential backup devices, all within distance  $D$  of  $x$ ; we call these  $k$  devices the *backup set* of  $x$ . Also, we do not want any device to be in the backup set of too many other devices; otherwise, a single failure might affect a large fraction of the network.

So suppose we are given the communication radius  $D$ , parameters  $b$  and  $k$ , and an array  $d[1..n, 1..n]$  of distances, where  $d[i, j]$  is the distance between device  $i$  and device  $j$ . Describe an algorithm that either computes a backup set of size  $k$  for each of the  $n$  devices, such that that no device appears in more than  $b$  backup sets, or reports (correctly) that no good collection of backup sets exists.

5. Let  $G = (V, E)$  be a directed graph where for each vertex  $v$ , the in-degree and out-degree of  $v$  are equal. Let  $u$  and  $v$  be two vertices  $G$ , and suppose  $G$  contains  $k$  edge-disjoint paths from  $u$  to  $v$ . Under these conditions, must  $G$  also contain  $k$  edge-disjoint paths from  $v$  to  $u$ ? Give a proof or a counterexample with explanation.

- \*6. [Extra credit] A *rooted tree* is a directed acyclic graph, in which every vertex has exactly one incoming edge, except for the *root*, which has no incoming edges. Equivalently, a rooted tree consists of a root vertex, which has edges pointing to the roots of zero or more smaller rooted trees. Describe a polynomial-time algorithm to compute, given two rooted trees  $A$  and  $B$ , the largest common rooted subtree of  $A$  and  $B$ .

[Hint: Let  $LCS(u, v)$  denote the largest common subtree whose root in  $A$  is  $u$  and whose root in  $B$  is  $v$ . Your algorithm should compute  $LCS(u, v)$  for all vertices  $u$  and  $v$  using dynamic programming. This would be easy if every vertex had  $O(1)$  children, and still straightforward if the children of each node were ordered from left to right and the common subtree had to respect that ordering. But for unordered trees with large degree, you need another trick to combine recursive subproblems efficiently. Don't waste your time trying to reduce the polynomial running time.]

---

<sup>1</sup>but not really in practice

# CS 573: Graduate Algorithms, Fall 2008

## Homework 6

Practice only

- 
- This homework is only for practice; do not submit solutions. At least one (sub)problem (or something *very* similar) will appear on the final exam.
- 

1. An *integer program* is a linear program with the additional constraint that the variables must take only integer values.
  - (a) Prove that deciding whether an integer program has a feasible solution is NP-complete.
  - (b) Prove that finding the optimal solution to an integer program is NP-hard.

[*Hint: Almost any NP-hard decision problem can be formulated as an integer program. Pick your favorite.*]

2. Describe precisely how to dualize a linear program written in general form:

$$\begin{aligned} & \text{maximize} \sum_{j=1}^d c_j x_j \\ & \text{subject to} \sum_{j=1}^d a_{ij} x_j \leq b_i \quad \text{for each } i = 1..p \\ & \quad \sum_{j=1}^d a_{ij} x_j = b_i \quad \text{for each } i = p+1..p+q \\ & \quad \sum_{j=1}^d a_{ij} x_j \geq b_i \quad \text{for each } i = p+q+1..n \end{aligned}$$

Keep the number of dual variables as small as possible. The dual of the dual of any linear program should be *syntactically identical* to the original linear program.

3. Suppose you have a subroutine that can solve linear programs in polynomial time, but only if they are both feasible and bounded. Describe an algorithm that solves *arbitrary* linear programs in polynomial time, using this subroutine as a black box. Your algorithm should return an optimal solution if one exists; if no optimum exists, your algorithm should report that the input instance is UNBOUNDED or INFEASIBLE, whichever is appropriate. [Hint: Add one constraint to guarantee boundedness; add one variable to guarantee feasibility.]

4. Suppose you are given a set  $P$  of  $n$  points in some high-dimensional space  $\mathbb{R}^d$ , each labeled either *black* or *white*. A *linear classifier* is a  $d$ -dimensional vector  $c$  with the following properties:

- If  $p$  is a black point, then  $p \cdot c > 0$ .
- If  $p$  is a white point, then  $p \cdot c < 0$ .

Describe an efficient algorithm to find a linear classifier for the given data points, or correctly report that none exists. [*Hint: This is almost trivial, but not quite.*]

Lots more linear programming problems can be found at <http://www.ee.ucla.edu/ee236a/homework/problems.pdf>. Enjoy!

You have 120 minutes to answer all five questions.  
**Write your answers in the separate answer booklet.**  
 Please turn in your question sheet and your cheat sheet with your answers.

1. You and your eight-year-old nephew Elmo decide to play a simple card game. At the beginning of the game, several cards are dealt face up in a long row. Then you and Elmo take turns removing either the leftmost or rightmost card from the row, until all the cards are gone. Each card is worth a different number of points. The player that collects the most points wins the game.

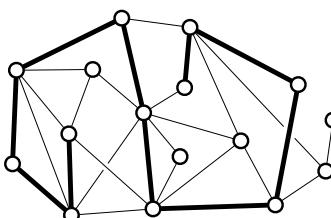
Like most eight-year-olds who haven't studied algorithms, Elmo follows the obvious greedy strategy every time he plays: ***Elmo always takes the card with the higher point value.*** Your task is to find a strategy that will beat Elmo whenever possible. (It might seem mean to beat up on a little kid like this, but Elmo absolutely *hates* it when grown-ups let him win.)

- (a) Describe an initial sequence of cards that allows you to win against Elmo, no matter who moves first, but *only* if you do *not* follow Elmo's greedy strategy.
- (b) Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that you can collect playing against Elmo.

Here is a sample game, where both you and Elmo are using the greedy strategy. Elmo wins 8–7. You cannot win this particular game, no matter what strategy you use.

Initial cards	2	4	5	1	3
Elmo takes the 3	2	4	5	1	\$
You take the 2	\$	4	5	1	
Elmo takes the 4		\$	5	1	
You take the 5			\$	1	
Elmo takes the 1				\$	

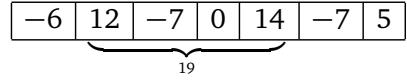
2. **Prove** that the following problem is NP-hard: Given an undirected graph  $G$ , find the longest path in  $G$  whose length is a multiple of 5.



This graph has a path of length 10, but no path of length 15.

3. Suppose you are given an array  $A[1..n]$  of integers. Describe and analyze an algorithm that finds the largest sum of elements in a contiguous subarray  $A[i..j]$ .

For example, if the array  $A$  contains the numbers  $[-6, 12, -7, 0, 14, -7, 5]$ , your algorithm should return the number 19:



4. A *shuffle* of two strings  $X$  and  $Y$  is formed by interspersing the characters into a new string, keeping the characters of  $X$  and  $Y$  in the same order. For example, ‘bananaanananas’ is a shuffle of ‘banana’ and ‘ananas’ in several different ways:

banana<sub>a</sub>nanas      ban<sub>a</sub>na<sub>a</sub>na<sub>n</sub>as      b<sub>a</sub>n<sub>a</sub><sub>a</sub>na<sub>a</sub>s

The strings ‘prodgyrnamatammiiincg’ and ‘dyprongarmammicing’ are both shuffles of ‘dynamic’ and ‘programming’:

pro<sup>d</sup>g<sub>y</sub>r<sup>n</sup>am<sub>m</sub>atamm<sub>i</sub><sup>i</sup>n<sup>c</sup>g      dy<sub>p</sub>ro<sup>n</sup>g<sup>a</sup>r<sup>m</sup>am<sup>m</sup>ic<sub>i</sub>ng

Given three strings  $A[1..m]$ ,  $B[1..n]$ , and  $C[1..m+n]$ , describe and analyze an algorithm to determine whether  $C$  is a shuffle of  $A$  and  $B$ .

5. Suppose you are given two sorted arrays  $A[1..m]$  and  $B[1..n]$  and an integer  $k$ . Describe an algorithm to find the  $k$ th smallest element in the union of  $A$  and  $B$  in  $\Theta(\log(m+n))$  time. For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \quad B[1..5] = [2, 5, 8, 17, 19] \quad k = 6$$

your algorithm should return 8. You can assume that the arrays contain no duplicates. An algorithm that works only in the special case  $n = m = k$  is worth 7 points.

[Hint: What can you learn from comparing one element of  $A$  to one element of  $B$ ?]

You have 120 minutes to answer all five questions.  
**Write your answers in the separate answer booklet.**  
Please turn in your question sheet and your cheat sheet with your answers.

1. Consider the following modification of the ‘dumb’ 2-approximation algorithm for minimum vertex cover that we saw in class. The only change is that we output a set of edges instead of a set of vertices.

```
APPROXMINMAXMATCHING( $G$ ):  

 $M \leftarrow \emptyset$   

while  $G$  has at least one edge  

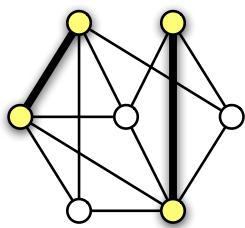
    let  $(u, v)$  be any edge in  $G$   

    remove  $u$  and  $v$  (and their incident edges) from  $G$   

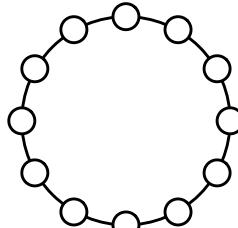
    add  $(u, v)$  to  $M$   

return  $M$ 
```

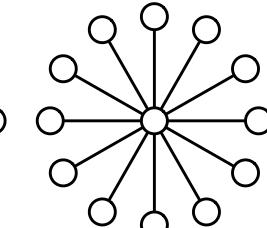
- (a) **Prove** that this algorithm computes a *matching*—no two edges in  $M$  share a common vertex.
- (b) **Prove** that  $M$  is a *maximal matching*— $M$  is not a proper subgraph of another matching in  $G$ .
- (c) **Prove** that  $M$  contains at most twice as many edges as the *smallest maximal matching* in  $G$ .



The smallest maximal matching in a graph.



A cycle and a star.



2. Consider the following heuristic for computing a small vertex cover of a graph.

- Assign a random *priority* to each vertex, chosen independently and uniformly from the real interval  $[0, 1]$  (just like treaps).
- Mark every vertex that does *not* have larger priority than *all* of its neighbors.

For any graph  $G$ , let  $OPT(G)$  denote the size of the smallest vertex cover of  $G$ , and let  $M(G)$  denote the number of nodes marked by this algorithm.

- (a) **Prove** that the set of vertices marked by this heuristic is *always* a vertex cover.
- (b) Suppose the input graph  $G$  is a *cycle*, that is, a connected graph where every vertex has degree 2. What is the expected value of  $M(G)/OPT(G)$ ? **Prove** your answer is correct.
- (c) Suppose the input graph  $G$  is a *star*, that is, a tree with one central vertex of degree  $n - 1$ . What is the expected value of  $M(G)/OPT(G)$ ? **Prove** your answer is correct.

3. Suppose we want to write an efficient function  $\text{SHUFFLE}(A[1..n])$  that randomly permutes the array  $A$ , so that each of the  $n!$  permutations is equally likely.

- (a) *Prove* that the following  $\text{SHUFFLE}$  algorithm is *not* correct. [Hint: There is a two-line proof.]

```

SHUFFLE( $A[1..n]$ ):
  for  $i = 1$  to  $n$ 
    swap  $A[i] \leftrightarrow A[\text{RANDOM}(n)]$ 
```

- (b) Describe and analyze a correct  $\text{SHUFFLE}$  algorithm whose expected running time is  $O(n)$ .

Your algorithm may call the function  $\text{RANDOM}(k)$ , which returns an integer uniformly distributed in the range  $\{1, 2, \dots, k\}$  in  $O(1)$  time. For example,  $\text{RANDOM}(2)$  simulates a fair coin flip, and  $\text{RANDOM}(1)$  *always* returns 1.

4. Let  $\Phi$  be a legal input for 3SAT—a boolean formula in conjunctive normal form, with exactly three literals in each clause. Recall that an assignment of boolean values to the variables in  $\Phi$  **satisfies** a clause if at least one of its literals is TRUE. The **maximum satisfiability problem**, sometimes called MAX3SAT, asks for the maximum number of clauses that can be simultaneously satisfied by a single assignment. Solving MAXSAT exactly is clearly also NP-hard; this problem asks about approximation algorithms.

- (a) Let  $\text{MaxSat}(\Phi)$  denote the maximum number of clauses that can be simultaneously satisfied by one variable assignment. Suppose we randomly assign each variable in  $\Phi$  to be TRUE or FALSE, each with equal probability. *Prove* that the expected number of satisfied clauses is at least  $\frac{7}{8}\text{MaxSat}(\Phi)$ .
- (b) Let  $\text{MinUnsat}(\Phi)$  denote the *minimum* number of clauses that can be simultaneously *unsatisfied* by a single assignment. *Prove* that it is NP-hard to approximate  $\text{MinUnsat}(\Phi)$  within a factor of  $10^{10^{100}}$ .

5. Consider the following randomized algorithm for generating biased random bits. The subroutine FAIRCOIN returns either 0 or 1 with equal probability; the random bits returned by FAIRCOIN are mutually independent.

```

ONEINTHREE:
  if FAIRCOIN = 0
    return 0
  else
    return 1 - ONEINTHREE
```

- (a) *Prove* that ONEINTHREE returns 1 with probability 1/3.
- (b) What is the *exact* expected number of times that this algorithm calls FAIRCOIN? *Prove* your answer is correct.
- (c) Now suppose you are given a subroutine ONEINTHREE that generates a random bit that is equal to 1 with probability 1/3. Describe a FAIRCOIN algorithm that returns either 0 or 1 with equal probability, using ONEINTHREE as a subroutine. **Your only source of randomness is ONEINTHREE; in particular, you may not use the RANDOM function from problem 3.**
- (d) What is the *exact* expected number of times that your FAIRCOIN algorithm calls ONEINTHREE? *Prove* your answer is correct.

You have 180 minutes to answer all seven questions.  
**Write your answers in the separate answer booklet.**  
You can keep everything except your answer booklet when you leave.

1. An *integer program* is a linear program with the additional constraint that the variables must take only integer values. **Prove** that deciding whether an integer program has a feasible solution is NP-complete. [Hint: Almost any NP-hard decision problem can be formulated as an integer program. Pick your favorite.]
2. Recall that a *priority search tree* is a binary tree in which every node has both a *search key* and a *priority*, arranged so that the tree is simultaneously a binary search tree for the keys and a min-heap for the priorities. A *heater* is a priority search tree in which the priorities are given by the user, and the search keys are distributed uniformly and independently at random in the real interval  $[0, 1]$ . Intuitively, a heater is the ‘opposite’ of a treap.

The following problems consider an  $n$ -node heater  $T$  whose node priorities are the integers from 1 to  $n$ . We identify nodes in  $T$  by their priorities; thus, ‘node 5’ means the node in  $T$  with priority 5. The min-heap property implies that node 1 is the root of  $T$ . Finally, let  $i$  and  $j$  be integers with  $1 \leq i < j \leq n$ .

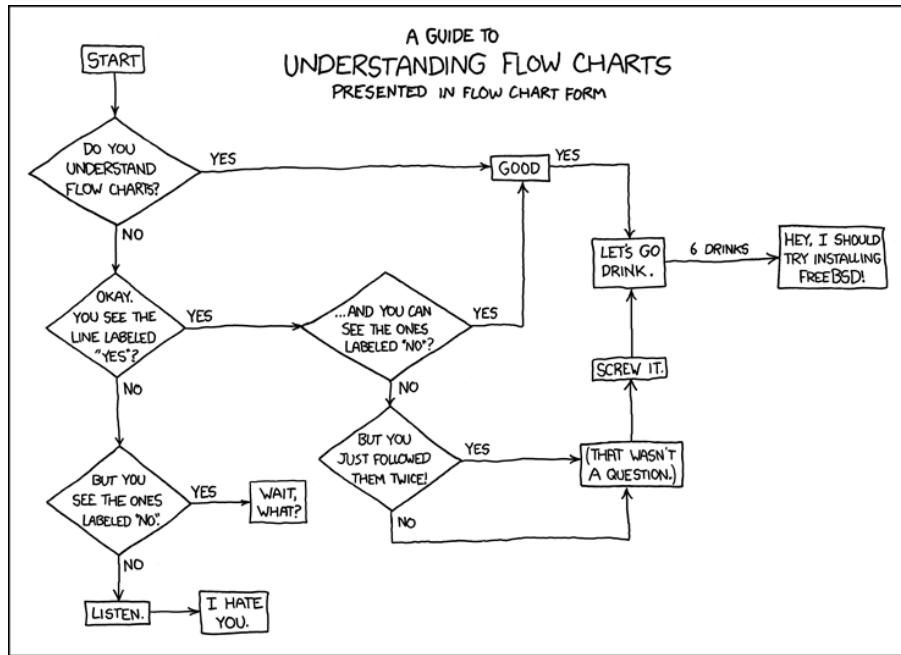
- (a) **Prove** that in a random permutation of the  $(i + 1)$ -element set  $\{1, 2, \dots, i, j\}$ , elements  $i$  and  $j$  are adjacent with probability  $2/(i + 1)$ .
  - (b) **Prove** that node  $i$  is an ancestor of node  $j$  with probability  $2/(i + 1)$ . [Hint: Use part (a)!]
  - (c) What is the probability that node  $i$  is a descendant of node  $j$ ? [Hint: Don’t use part (a)!]
  - (d) What is the exact expected depth of node  $j$ ?
- 
3. The UIUC Faculty Senate has decided to convene a committee to determine whether Chief Illiniwek should become the official *maseet symbol* of the University of Illinois Global Campus. Exactly one faculty member must be chosen from each academic department to serve on this committee. Some faculty members have appointments in multiple departments, but each committee member will represent only one department. For example, if Prof. Blagojevich is affiliated with both the Department of Corruption and the Department of Stupidity, and he is chosen as the Stupidity representative, then someone else must represent Corruption. Finally, University policy requires that any committee on virtual *maseots symbols* must contain the same number of assistant professors, associate professors, and full professors. Fortunately, the number of departments is a multiple of 3.
- Describe an efficient algorithm to select the membership of the Global Illiniwek Committee. Your input is a list of all UIUC faculty members, their ranks (assistant, associate, or full), and their departmental affiliation(s). There are  $n$  faculty members and  $3k$  departments.
- 
4. Let  $\alpha(G)$  denote the number of vertices in the largest independent set in a graph  $G$ . **Prove** that the following problem is NP-hard: Given a graph  $G$ , return *any* integer between  $\alpha(G) - 31337$  and  $\alpha(G) + 31337$ .

5. Let  $G = (V, E)$  be a directed graph with capacities  $c: E \rightarrow \mathbb{R}^+$ , a source vertex  $s$ , and a target vertex  $t$ . Suppose someone hands you a function  $f: E \rightarrow \mathbb{R}$ . Describe and analyze a fast algorithm to determine whether  $f$  is a maximum  $(s, t)$ -flow in  $G$ .
6. For some strange reason, you decide to ride your bicycle 3688 miles from Urbana to Wasilla, Alaska, to join in the annual Wasilla Mining Festival and Helicopter Wolf Hunt. The festival starts exactly 32 days from now, so you need to bike an average of 109 miles each day. Because you are a poor starving student, you can only afford to sleep at campgrounds, which are unfortunately *not* spaced exactly 109 miles apart. So some days you will have to ride more than average, and other days less, but you would like to keep the variation as small as possible. You settle on a formal scoring system to help decide where to sleep; if you ride  $x$  miles in one day, your score for that day is  $(109 - x)^2$ . What is the minimum possible total score for all 32 days?

More generally, suppose you have  $D$  days to travel  $DP$  miles, there are  $n$  campgrounds along your route, and your score for traveling  $x$  miles in one day is  $(x - P)^2$ . You are given a sorted array  $dist[1..n]$  of real numbers, where  $dist[i]$  is the distance from your starting location to the  $i$ th campground; it may help to also set  $dist[0] = 0$  and  $dist[n + 1] = DP$ . Describe and analyze a fast algorithm to compute the minimum possible score for your trip. The running time of your algorithm should depend on the integers  $D$  and  $n$ , but not on the real number  $P$ .

7. Describe and analyze efficient algorithms for the following problems.

- Given a set of  $n$  integers, does it contain elements  $a$  and  $b$  such that  $a + b = 0$ ?
- Given a set of  $n$  integers, does it contain elements  $a$ ,  $b$ , and  $c$  such that  $a + b = c$ ?



— Randall Munroe, *xkcd*, December 17, 2008 (<http://xkcd.com/518/>)

# CS 473: Undergraduate Algorithms, Spring 2009

## Homework 0

Due in class at 11:00am, Tuesday, January 27, 2009

---

- This homework tests your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, graphs, and most importantly, induction—to help you identify gaps in your background knowledge. **You are responsible for filling those gaps.** The early chapters of any algorithms textbook should be sufficient review, but you may also want consult your favorite discrete mathematics and data structures textbooks. If you need help, please ask in office hours and/or on the course newsgroup.
- Each student must submit individual solutions for this homework. For all future homeworks, groups of up to three students may submit a single, common solution.
- Please carefully read the course policies linked from the course web site. If you have *any* questions, please ask during lecture or office hours, or post your question to the course newsgroup. In particular:
  - Submit five separately stapled solutions, one for each numbered problem, with your name and NetID clearly printed on each page. Please do not staple everything together.
  - You may use any source at your disposal—paper, electronic, or human—but you **must** write your solutions in your own words, and you **must** cite every source that you use.
  - Unless explicitly stated otherwise, **every** homework problem requires a proof.
  - Answering “I don’t know” to any homework or exam problem (except for extra credit problems) is worth 25% partial credit.
  - Algorithms or proofs containing phrases like “and so on” or “repeat this process for all  $n$ ”, instead of an explicit loop, recursion, or induction, will receive 0 points.

**Write the sentence “I understand the course policies.” at the top of your solution to problem 1.**

---

1. Professor George O’Jungle has a 27-node binary tree, in which every node is labeled with a unique letter of the Roman alphabet or the character &. Preorder and postorder traversals of the tree visit the nodes in the following order:
  - Preorder: I Q J H L E M V O T S B R G Y Z K C A & F P N U D W X
  - Postorder: H E M L J V Q S G Y R Z B T C P U D N F W & X A K O I
  - (a) List the nodes in George’s tree in the order visited by an inorder traversal.
  - (b) Draw George’s tree.

2. (a) [5 pts] Solve the following recurrences. State tight asymptotic bounds for each function in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ . Assume reasonable but nontrivial base cases. If your solution requires a particular base case, say so. ***Do not submit proofs***—just a list of five functions—but you should do them anyway, just for practice.

$$A(n) = 10A(n/5) + n$$

$$B(n) = 2B\left(\left\lceil \frac{n+3}{4} \right\rceil\right) + 5n^{6/7} - 8\sqrt{\frac{n}{\log n}} + 9\left\lfloor \log^{10} n \right\rfloor - 11$$

$$C(n) = 3C(n/2) + C(n/3) + 5C(n/6) + n^2$$

$$D(n) = \max_{0 < k < n} (D(k) + D(n - k) + n)$$

$$E(n) = \frac{E(n-1)E(n-3)}{E(n-2)}$$

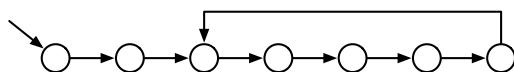
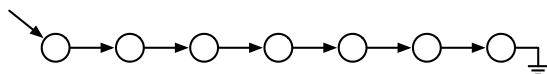
[Hint: Write out the first 20 terms.]

- (b) [5 pts] Sort the following functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. ***Do not submit proofs***—just a sorted list of 16 functions—but you should do them anyway, just for practice.

Write  $f(n) \ll g(n)$  to indicate that  $f(n) = o(g(n))$ , and write  $f(n) \equiv g(n)$  to mean  $f(n) = \Theta(g(n))$ . We use the notation  $\lg n = \log_2 n$ .

$n$	$\lg n$	$\sqrt{n}$	$3^n$
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$3^{\sqrt{n}}$	$\sqrt{3^n}$
$3^{\lg n}$	$\lg(3^n)$	$3^{\lg \sqrt{n}}$	$3^{\sqrt{\lg n}}$
$\sqrt{3^{\lg n}}$	$\lg(3^{\sqrt{n}})$	$\lg \sqrt{3^n}$	$\sqrt{\lg(3^n)}$

3. Suppose you are given a pointer to the head of singly linked list. Normally, each node in the list has a pointer to the next element, and the last node's pointer is NULL. Unfortunately, your list might have been corrupted (by a bug in *somebody else's code*, of course), so that some node's pointer leads back to an earlier node in the list.



Top: A standard singly-linked list. Bottom: A corrupted singly-linked list.

Describe an algorithm<sup>1</sup> that determines whether the linked list is corrupted or not. Your algorithm must not modify the list. For full credit, your algorithm should run in  $O(n)$  time, where  $n$  is the number of nodes in the list, and use  $O(1)$  extra space (not counting the list itself).

<sup>1</sup>Since you understand the course policies, you know what this phrase means. Right?

4. (a) Prove that any integer (positive, negative, or zero) can be written in the form  $\sum_i \pm 3^i$ , where the exponents  $i$  are distinct non-negative integers. For example:

$$42 = 3^4 - 3^3 - 3^2 - 3^1$$

$$25 = 3^3 - 3^1 + 3^0$$

$$17 = 3^3 - 3^2 - 3^0$$

- (b) Prove that any integer (positive, negative, or zero) can be written in the form  $\sum_i (-2)^i$ , where the exponents  $i$  are distinct non-negative integers. For example:

$$42 = (-2)^6 + (-2)^5 + (-2)^4 + (-2)^0$$

$$25 = (-2)^6 + (-2)^5 + (-2)^3 + (-2)^0$$

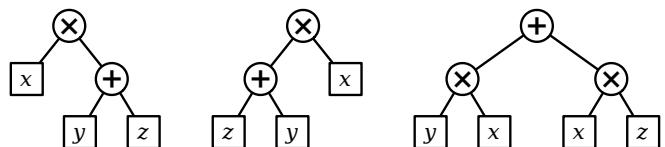
$$17 = (-2)^4 + (-2)^0$$

[Hint: Don't use weak induction. In fact, **never** use weak induction.]

5. An **arithmetic expression tree** is a binary tree where every leaf is labeled with a variable, every internal node is labeled with an arithmetic operation, and every internal node has exactly two children. For this problem, assume that the only allowed operations are  $+$  and  $\times$ . Different leaves may or may not represent different variables.

Every arithmetic expression tree represents a function, transforming input values for the leaf variables into an output value for the root, by following two simple rules: (1) The value of any  $+$ -node is the sum of the values of its children. (2) The value of any  $\times$ -node is the product of the values of its children.

Two arithmetic expression trees are **equivalent** if they represent the same function; that is, the same input values for the leaf variables always leads to the same output value at both roots. An arithmetic expression tree is in **normal form** if the parent of every  $+$ -node (if any) is another  $+$ -node.



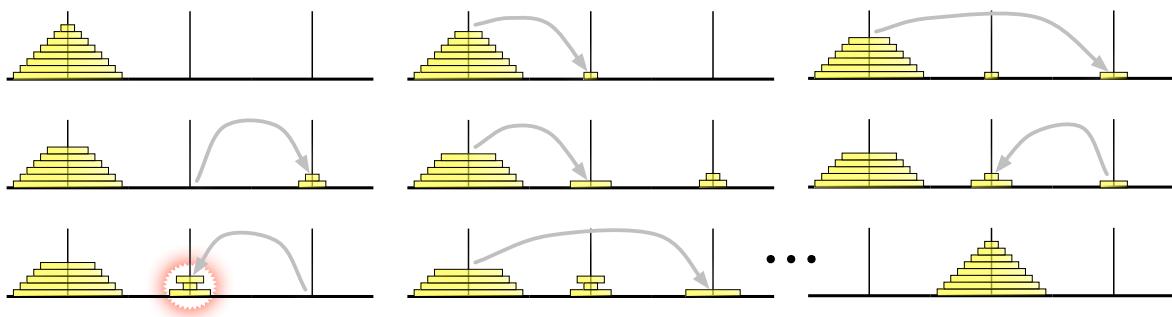
Three equivalent expression trees. Only the third is in normal form.

Prove that for any arithmetic expression tree, there is an equivalent arithmetic expression tree in normal form.

- \*6. [Extra credit] You may be familiar with the story behind the famous Tower of Hanoi puzzle:

At the great temple of Benares, there is a brass plate on which three vertical diamond shafts are fixed. On the shafts are mounted  $n$  golden disks of decreasing size. At the time of creation, the god Brahma placed all of the disks on one pin, in order of size with the largest at the bottom. The Hindu priests unceasingly transfer the disks from peg to peg, one at a time, never placing a larger disk on a smaller one. When all of the disks have been transferred to the last pin, the universe will end.

Recently the temple at Benares was relocated to southern California, where the monks are considerably more laid back about their job. At the “Towers of Hollywood”, the golden disks have been replaced with painted plywood, and the diamond shafts have been replaced with Plexiglas. More importantly, the restriction on the order of the disks has been relaxed. While the disks are being moved, the bottom disk on any pin must be the *largest* disk on that pin, but disks further up in the stack can be in any order. However, after all the disks have been moved, they must be in sorted order again.



The Towers of Hollywood. The sixth move leaves the disks out of order.

Describe an algorithm that moves a stack of  $n$  disks from one pin to the another using the smallest possible number of moves. Exactly how many moves does your algorithm perform? [Hint: The Hollywood monks can bring about the end of the universe considerably faster than their Benaresian counterparts.]

# CS 473: Undergraduate Algorithms, Spring 2009

## Homework 1

Due Tuesday, February 3, 2009 at 11:59:59pm.

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.

- The traditional Devonian/Cornish drinking song "The Barley Mow" has the following pseudolyrics, where  $\text{container}[i]$  is the name of a container that holds  $2^i$  ounces of beer. One version of the song uses the following containers: nippertkin, gill pot, half-pint, pint, quart, pottle, gallon, half-anker, anker, firkin, half-barrel, barrel, hogshead, pipe, well, river, and ocean. (Every container in this list is twice as big as its predecessor, except that a firkin is actually 2.25 ankers, and the last three units are just silly.)

**BARLEYMOW( $n$ ):**

```

"Here's a health to the barley-mow, my brave boys,"
"Here's a health to the barley-mow!"

"We'll drink it out of the jolly brown bowl,"
"Here's a health to the barley-mow!"
"Here's a health to the barley-mow, my brave boys,"
"Here's a health to the barley-mow!"

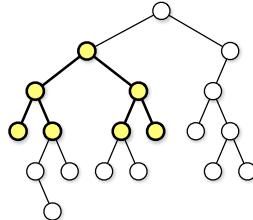
for i ← 1 to n
    "We'll drink it out of the container[i], boys,"
    "Here's a health to the barley-mow!"
    for j ← i downto 1
        "The container[j],"
        "And the jolly brown bowl!"
        "Here's a health to the barley-mow!"
        "Here's a health to the barley-mow, my brave boys,"
        "Here's a health to the barley-mow!"

```

- Suppose each container name  $\text{container}[i]$  is a single word, and you can sing four words a second. How long would it take you to sing BARLEYMOW( $n$ )? (Give a tight asymptotic bound.) [Hint: Is 'barley-mow' one word or two? Does it matter?]
- If you want to sing this song for  $n > 20$ , you'll have to make up your own container names. To avoid repetition, these names will get progressively longer as  $n$  increases<sup>1</sup>. Suppose  $\text{container}[n]$  has  $\Theta(\log n)$  syllables, and you can sing six syllables per second. Now how long would it take you to sing BARLEYMOW( $n$ )? (Give a tight asymptotic bound.)
- Suppose each time you mention the name of a container, you actually drink the corresponding amount of beer: one ounce for the jolly brown bowl, and  $2^i$  ounces for each  $\text{container}[i]$ . Assuming for purposes of this problem that you are at least 21 years old, exactly how many ounces of beer would you drink if you sang BARLEYMOW( $n$ )? (Give an exact answer, not just an asymptotic bound.)

<sup>1</sup>"We'll drink it out of the hemisemidemiyottapint, boys!"

2. For this problem, a *subtree* of a binary tree means any connected subgraph; a binary tree is *complete* if every leaf has exactly the same depth. Describe and analyze a recursive algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return the root and the depth of this subtree.



The largest complete subtree of this binary tree has depth 2.

3. (a) Describe and analyze a recursive algorithm to reconstruct a binary tree, given its preorder and postorder node sequences (as in Homework 0, problem 1).  
(b) Describe and analyze a recursive algorithm to reconstruct a binary tree, given its preorder and *inorder* node sequences.

# CS 473: Undergraduate Algorithms, Spring 2009

## Homework 10

Due Tuesday, May 5, 2009 at 11:59:59pm

---

- Groups of up to three students may submit a single, common solution. Please clearly write every group member's name and NetID on every page of your submission.
  - ***This homework is optional.*** If you submit solutions, they will be graded, and your overall homework grade will be the average of ten homeworks (Homeworks 0–10, dropping the lowest). If you do not submit solutions, your overall homework grade will be the average of *nine* homeworks (Homeworks 0–9, dropping the lowest).
- 

1. Suppose you are given a magic black box that can determine ***in polynomial time***, given an arbitrary graph  $G$ , the number of vertices in the largest complete subgraph of  $G$ . Describe and analyze a ***polynomial-time*** algorithm that computes, given an arbitrary graph  $G$ , a complete subgraph of  $G$  of maximum size, using this magic black box as a subroutine.
2. PLANARCIRCUITSAT is a special case of CIRCUITSAT where the input circuit is drawn ‘nicely’ in the plane — no two wires cross, no two gates touch, and each wire touches only the gates it connects. (Not every circuit can be drawn this way!) As in the general CIRCUITSAT problem, we want to determine if there is an input that makes the circuit output TRUE?

Prove that PLANARCIRCUITSAT is NP-complete. [*Hint: XOR.*]

3. For each problem below, either describe a polynomial-time algorithm or prove that the problem is NP-complete.
  - (a) A ***double-Eulerian*** circuit in an undirected graph  $G$  is a closed walk that traverses every edge in  $G$  exactly twice. Given a graph  $G$ , does  $G$  have a ***double-Eulerian*** circuit?
  - (b) A ***double-Hamiltonian*** circuit in an undirected graph  $G$  is a closed walk that visits every vertex in  $G$  exactly twice. Given a graph  $G$ , does  $G$  have a ***double-Hamiltonian*** circuit?

# CS 473: Undergraduate Algorithms, Spring 2009

## Homework 2

Written solutions due Tuesday, February 10, 2009 at 11:59:59pm.

---

- Roughly 1/3 of the students will give oral presentations of their solutions to the TAs. ***Please check Compass to check whether you are supposed give an oral presentation for this homework.*** Please see the course web page for further details.
  - Groups of up to three students may submit a common solution. Please clearly write every group member's name and NetID on every page of your submission.
  - Please start your solution to each numbered problem on a new sheet of paper. Please *don't* staple solutions for different problems together.
  - ***For this homework only:*** These homework problems ask you to describe recursive backtracking algorithms for various problems. ***Don't*** use memoization or dynamic programming to make your algorithms more efficient; you'll get to do that on HW3. ***Don't*** analyze the running times of your algorithms. The ***only*** things you should submit for each problem are (1) a description of your recursive algorithm, and (2) a brief justification for its correctness.
- 

1. A ***basic arithmetic expression*** is composed of characters from the set  $\{1, +, \times\}$  and parentheses. Almost every integer can be represented by more than one basic arithmetic expression. For example, all of the following basic arithmetic expression represent the integer 14:

$$\begin{aligned}
 & 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 \\
 & ((1 + 1) \times (1 + 1 + 1 + 1 + 1)) + ((1 + 1) \times (1 + 1)) \\
 & (1 + 1) \times (1 + 1 + 1 + 1 + 1 + 1 + 1) \\
 & (1 + 1) \times (((1 + 1 + 1) \times (1 + 1)) + 1)
 \end{aligned}$$

Describe a recursive algorithm to compute, given an integer  $n$  as input, the minimum number of 1's in a basic arithmetic expression whose value is  $n$ . The number of parentheses doesn't matter, just the number of 1's. For example, when  $n = 14$ , your algorithm should return 8, for the final expression above.

2. A sequence  $A = \langle a_1, a_2, \dots, a_n \rangle$  is ***bitonic*** if there is an index  $i$  with  $1 < i < n$ , such that the prefix  $\langle a_1, a_2, \dots, a_i \rangle$  is strictly increasing and the suffix  $\langle a_i, a_{i+1}, \dots, a_n \rangle$  is strictly decreasing. In particular, a bitonic sequence must contain at least three elements.

Describe a recursive algorithm to compute, given a sequence  $A$ , the length of the longest bitonic subsequence of  $A$ .

3. A *palindrome* is a string that reads the same forwards and backwards, like x, pop, noon, redivider, or amanaplanacatahamayakayamahatacanalpanama. Any string can be broken into sequence of palindromes. For example, the string bubbbaseesabana ('Bubba sees a banana.') can be broken into palindromes in several different ways; for example:

bub + baseesab + anana  
b + u + bb + a + sees + aba + nan + a  
b + u + bb + a + sees + a + b + anana  
b + u + b + b + a + s + e + e + s + a + b + a + n + a + n + a

Describe a recursive algorithm to compute the minimum number of palindromes that make up a given input string. For example, given the input string bubbbaseesabana, your algorithm would return the integer 3.

# CS 473: Undergraduate Algorithms, Spring 2009

## Homework 3

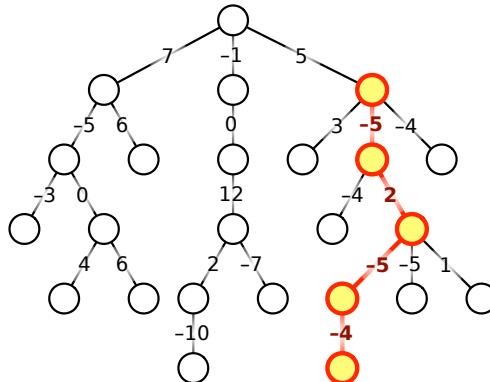
Written solutions due Tuesday, February 17, 2009 at 11:59:59pm.

---

1. Redo Homework 2, but now with dynamic programming!
  - (a) Describe and analyze an efficient algorithm to compute the minimum number of 1's in a basic arithmetic expression whose value is a given positive integer.
  - (b) Describe and analyze an efficient algorithm to compute the length of the longest bitonic subsequence of a given input sequence.
  - (c) Describe and analyze an efficient algorithm to compute the minimum number of palindromes that make up a given input string.

Please see Homework 2 for more detailed descriptions of each problem. ***Solutions for Homework 2 will be posted Friday, after the HW2 oral presentations.*** You may (and should!) use anything from those solutions without justification.

2. Let  $T$  be a rooted tree with integer weights on its edges, which could be positive, negative, or zero. Design an algorithm to find the minimum-length path from a node in  $T$  down to one of its descendants. The length of a path is the sum of the weights of its edges. For example, given the tree shown below, your algorithm should return the number  $-12$ . For full credit, your algorithm should run in  $O(n)$  time.



The minimum-weight downward path in this tree has weight  $-12$ .

3. Describe and analyze an efficient algorithm to compute the longest common subsequence of *three* given strings. For example, given the input strings EPIDEMIOLOGIST, REFRIGERATION, and SUPERCALIFRAGILISTICEXPIALODOCIOUS, your algorithm should return the number 5, because the longest common subsequence is EIEIO.

# CS 473: Undergraduate Algorithms, Spring 2009

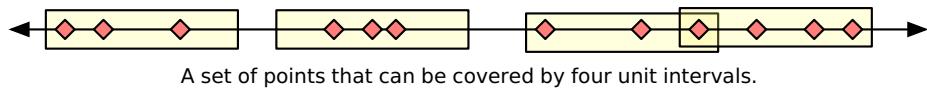
## Homework 3½

Practice only

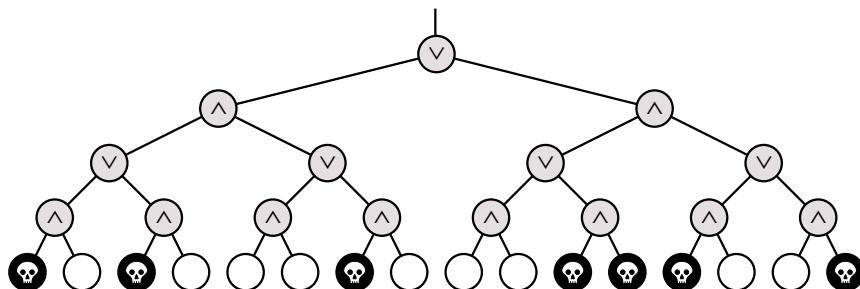
---

- After graduating from UIUC, you are hired by a mobile phone company to plan the placement of new cell towers along a long, straight, nearly-deserted highway out west. Each cell tower can transmit the same fixed distance from its location. Federal law requires that any building along the highway must be within the broadcast range of at least one tower. On the other hand, your company wants to build as few towers as possible. Given the locations of the buildings, where should you build the towers?

More formally, suppose you are given a set  $X = \{x_1, x_2, \dots, x_n\}$  of points on the real number line. Describe an algorithm to compute the minimum number of intervals of length 1 that can cover all the points in  $X$ . For full credit, your algorithm should run in  $O(n \log n)$  time.



- The *left spine* of a binary tree is a path starting at the root and following only left-child pointers down to a leaf. What is the expected number of nodes in the left spine of an  $n$ -node treap?
  - What is the expected number of leaves in an  $n$ -node treap? [Hint: What is the probability that in an  $n$ -node treap, the node with  $k$ th smallest search key is a leaf?]
  - Prove that the expected number of proper descendants of any node in a treap is exactly equal to the expected depth of that node.
- Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with  $4^n$  leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after  $2n$  moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.



You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are AND gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

- (a) Describe and analyze a deterministic algorithm to determine whether or not you can win. [*Hint: This is easy!*]
- (b) Unfortunately, Death won't give you enough time to look at every node in the tree. Describe a randomized algorithm that determines whether you can win in  $O(3^n)$  expected time. [*Hint: Consider the case  $n = 1$ .*]
- \*(c) Describe a randomized algorithm that determines whether you can win in  $O(c^n)$  expected time, for some constant  $c < 3$ . [*Hint: You may not need to change your algorithm from part (b) at all!*]

# CS 473: Undergraduate Algorithms, Spring 2009

## Homework 3

Written solutions due Tuesday, March 2, 2009 at 11:59:59pm.

---

1. A *meldable priority queue* stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:

- **MAKEQUEUE**: Return a new priority queue containing the empty set.
- **FINDMIN( $Q$ )**: Return the smallest element of  $Q$  (if any).
- **DELETEMIN( $Q$ )**: Remove the smallest element in  $Q$  (if any).
- **INSERT( $Q, x$ )**: Insert element  $x$  into  $Q$ , if it is not already there.
- **DECREASEKEY( $Q, x, y$ )**: Replace an element  $x \in Q$  with a smaller key  $y$ . (If  $y > x$ , the operation fails.) The input is a pointer directly to the node in  $Q$  containing  $x$ .
- **DELETE( $Q, x$ )**: Delete the element  $x \in Q$ . The input is a pointer directly to the node in  $Q$  containing  $x$ .
- **MELD( $Q_1, Q_2$ )**: Return a new priority queue containing all the elements of  $Q_1$  and  $Q_2$ ; this operation destroys  $Q_1$  and  $Q_2$ .

A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm:

```

MELD( $Q_1, Q_2$ ):
    if  $Q_1$  is empty return  $Q_2$ 
    if  $Q_2$  is empty return  $Q_1$ 
    if  $key(Q_1) > key(Q_2)$ 
        swap  $Q_1 \leftrightarrow Q_2$ 
    with probability  $1/2$ 
         $left(Q_1) \leftarrow \text{MELD}(left(Q_1), Q_2)$ 
    else
         $right(Q_1) \leftarrow \text{MELD}(right(Q_1), Q_2)$ 
    return  $Q_1$ 

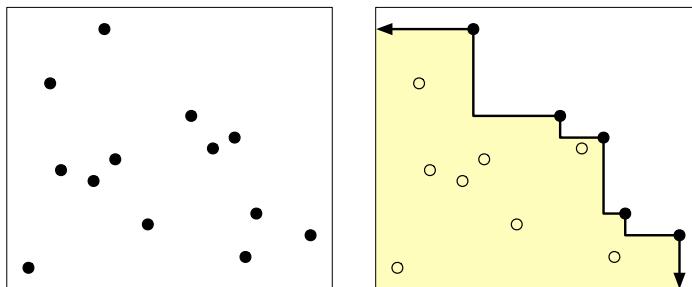
```

- (a) Prove that for *any* heap-ordered binary trees  $Q_1$  and  $Q_2$  (*not just those constructed by the operations listed above*), the expected running time of  $\text{MELD}(Q_1, Q_2)$  is  $O(\log n)$ , where  $n$  is the total number of nodes in both trees. [Hint: How long is a random root-to-leaf path in an  $n$ -node binary tree if each left/right choice is made with equal probability?]
- (b) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and  $O(1)$  additional time. (This implies that every operation takes  $O(\log n)$  expected time.)

2. Recall that a *priority search tree* is a binary tree in which every node has both a *search key* and a *priority*, arranged so that the tree is simultaneously a binary search tree for the keys and a min-heap for the priorities. A *heater* is a priority search tree in which the priorities are given by the user, and the search keys are distributed uniformly and independently at random in the real interval  $[0, 1]$ . Intuitively, a heater is the ‘opposite’ of a treap.

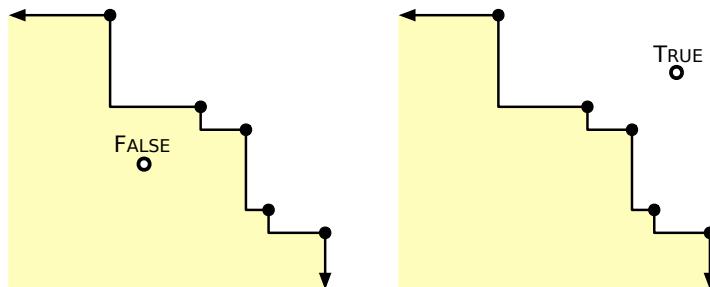
The following problems consider an  $n$ -node heater  $T$  whose node priorities are the integers from 1 to  $n$ . We identify nodes in  $T$  by their priorities; thus, ‘node 5’ means the node in  $T$  with priority 5. The min-heap property implies that node 1 is the root of  $T$ . Finally, let  $i$  and  $j$  be integers with  $1 \leq i < j \leq n$ .

- (a) **Prove** that in a random permutation of the  $(i + 1)$ -element set  $\{1, 2, \dots, i, j\}$ , elements  $i$  and  $j$  are adjacent with probability  $2/(i + 1)$ .
  - (b) **Prove** that node  $i$  is an ancestor of node  $j$  with probability  $2/(i + 1)$ . [Hint: Use part (a)!]
  - (c) What is the probability that node  $i$  is a descendant of node  $j$ ? [Hint: Don’t use part (a)!]
  - (d) What is the exact expected depth of node  $j$ ?
3. Let  $P$  be a set of  $n$  points in the plane. The *staircase* of  $P$  is the set of all points in the plane that have at least one point in  $P$  both above and to the right.



A set of points in the plane and its staircase (shaded).

- (a) Describe an algorithm to compute the staircase of a set of  $n$  points in  $O(n \log n)$  time.
- (b) Describe and analyze a data structure that stores the staircase of a set of points, and an algorithm ABOVE?( $x, y$ ) that returns TRUE if the point  $(x, y)$  is above the staircase, or FALSE otherwise. Your data structure should use  $O(n)$  space, and your ABOVE? algorithm should run in  $O(\log n)$  time.



Two staircase queries.

# CS 473: Undergraduate Algorithms, Spring 2009

## Homework 5

Written solutions due Tuesday, March 9, 2009 at 11:59:59pm.

---

1. Remember the difference between stacks and queues? Good.
  - (a) Describe how to implement a queue using two stacks and  $O(1)$  additional memory, so that the amortized time for any enqueue or dequeue operation is  $O(1)$ . The *only* access you have to the stacks is through the standard methods PUSH and POP.
  - (b) A *quack* is an abstract data type that combines properties of both stacks and queues. It can be viewed as a list of elements written left to right such that three operations are possible:
    - **Push:** add a new item to the left end of the list;
    - **Pop:** remove the item on the left end of the list;
    - **Pull:** remove the item on the right end of the list.
 Implement a quack using *three* stacks and  $O(1)$  additional memory, so that the amortized time for any push, pop, or pull operation is  $O(1)$ . Again, you are *only* allowed to access the stacks through the standard methods PUSH and POP.
  
2. In a *dirty* binary search tree, each node is labeled either *clean* or *dirty*. The lazy deletion scheme used for scapegoat trees requires us to *purge* the search tree, keeping all the clean nodes and deleting all the dirty nodes, as soon as half the nodes become dirty. In addition, the purged tree should be perfectly balanced.
 

Describe and analyze an algorithm to purge an *arbitrary*  $n$ -node dirty binary search tree in  $O(n)$  time, using at most  $O(\log n)$  space (in addition to the tree itself). Don't forget to include the recursion stack in your space bound. An algorithm that uses  $\Theta(n)$  additional space in the worst case is worth half credit.
  
3. Some applications of binary search trees attach a *secondary data structure* to each node in the tree, to allow for more complicated searches. Maintaining these secondary structures usually complicates algorithms for keeping the top-level search tree balanced.
 

Let  $T$  be an arbitrary binary tree. Suppose every node  $v$  in  $T$  stores a secondary structure of size  $O(\text{size}(v))$ , which can be built in  $O(\text{size}(v))$  time, where  $\text{size}(v)$  denotes the number of descendants of  $v$ . Performing a rotation at any node  $v$  now requires  $O(\text{size}(v))$  time, because we have to rebuild one of the secondary structures.

  - (a) [1 pt] Overall, how much space does this data structure use *in the worst case*?
  - (b) [1 pt] How much space does this structure use if the primary search tree  $T$  is perfectly balanced?
  - (c) [2 pts] Suppose  $T$  is a splay tree. Prove that the *amortized* cost of a splay (and therefore of a search, insertion, or deletion) is  $\Omega(n)$ . [*Hint: This is easy!*]

- (d) [3 pts] Now suppose  $T$  is a scapegoat tree, and that rebuilding the subtree rooted at  $v$  requires  $\Theta(\text{size}(v) \log \text{size}(v))$  time (because we also have to rebuild the secondary structures at every descendant of  $v$ ). What is the *amortized* cost of inserting a new element into  $T$ ?
- (e) [3 pts] Finally, suppose  $T$  is a treap. What's the worst-case *expected* time for inserting a new element into  $T$ ?

# CS 473: Undergraduate Algorithms, Spring 2009

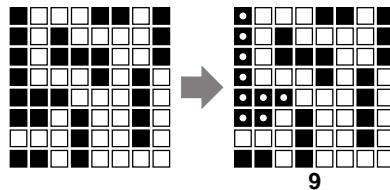
## Homework 6

Written solutions due Tuesday, March 17, 2009 at 11:59:59pm.

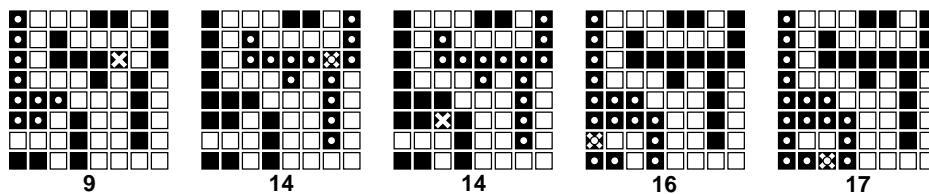
1. Let  $G$  be an undirected graph with  $n$  nodes. Suppose that  $G$  contains two nodes  $s$  and  $t$ , such that every path from  $s$  to  $t$  contains more than  $n/2$  edges.
  - (a) Prove that  $G$  must contain a vertex  $v$  that lies on *every* path from  $s$  to  $t$ .
  - (b) Describe an algorithm that finds such a vertex  $v$  in  $O(V + E)$  time.
  
2. Suppose you are given a graph  $G$  with weighted edges and a minimum spanning tree  $T$  of  $G$ .
  - (a) Describe an algorithm to update the minimum spanning tree when the weight of a single edge  $e$  is decreased.
  - (b) Describe an algorithm to update the minimum spanning tree when the weight of a single edge  $e$  is increased.

In both cases, the input to your algorithm is the edge  $e$  and its new weight; your algorithms should modify  $T$  so that it is still a minimum spanning tree. [Hint: Consider the cases  $e \in T$  and  $e \notin T$  separately.]

3. (a) Describe and analyze an algorithm to compute the size of the largest connected component of black pixels in an  $n \times n$  bitmap  $B[1..n, 1..n]$ .  
 For example, given the bitmap below as input, your algorithm should return the number 9, because the largest connected black component (marked with white dots on the right) contains nine pixels.



- (b) Design and analyze an algorithm  $\text{BLACKEN}(i, j)$  that colors the pixel  $B[i, j]$  black and returns the size of the largest black component in the bitmap. For full credit, the *amortized* running time of your algorithm (starting with an all-white bitmap) must be as small as possible.  
 For example, at each step in the sequence below, we blacken the pixel marked with an X. The largest black component is marked with white dots; the number underneath shows the correct output of the  $\text{BLACKEN}$  algorithm.



- (c) What is the *worst-case* running time of your  $\text{BLACKEN}$  algorithm?

# CS 473: Undergraduate Algorithms, Spring 2009

## Homework 6½

Practice only—do not submit solutions

---

1. In class last Tuesday, we discussed Ford's generic shortest-path algorithm—relax arbitrary tense edges until no edge is tense. This problem asks you to fill in part of the proof that this algorithm is correct.

- (a) Prove that after *every* call to RELAX, for every vertex  $v$ , either  $dist(v) = \infty$  or  $dist(v)$  is the total weight of some path from  $s$  to  $v$ .
  - (b) Prove that for every vertex  $v$ , when the generic algorithm halts, either  $pred(v) = \text{NULL}$  and  $dist(v) = \infty$  or  $dist(v)$  is the total weight of the predecessor chain ending at  $v$ :

$$s \rightarrow \dots \rightarrow pred(pred(v)) \rightarrow pred(v) \rightarrow v.$$

2. Describe a modification of Shimbels shortest-path algorithm that actually computes a negative-weight cycle if any such cycle is reachable from  $s$ , or a shortest-path tree rooted at  $s$  if there is no such cycle. Your modified algorithm should still run in  $O(VE)$  time.
3. After graduating you accept a job with Aerophobes-Я-Us, the leading traveling agency for people who hate to fly. Your job is to build a system to help customers plan airplane trips from one city to another. All of your customers are afraid of flying (and by extension, airports), so any trip you plan needs to be as short as possible. You know all the departure and arrival times of all the flights on the planet.

Suppose one of your customers wants to fly from city  $X$  to city  $Y$ . Describe an algorithm to find a sequence of flights that minimizes the *total time in transit*—the length of time from the initial departure to the final arrival, including time at intermediate airports waiting for connecting flights. [Hint: Modify the input data and apply Dijkstra's algorithm.]

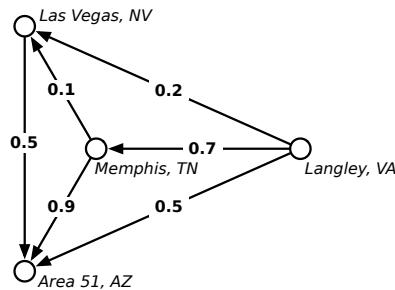
# CS 473: Undergraduate Algorithms, Spring 2009

## Homework 6 $\frac{3}{4}$

Practice only—do not submit solutions

1. Mulder and Scully have computed, for every road in the United States, the exact probability that someone driving on that road *won't* be abducted by aliens. Agent Mulder needs to drive from Langley, Virginia to Area 51, Nevada. What route should he take so that he has the least chance of being abducted?

More formally, you are given a directed graph  $G = (V, E)$ , where every edge  $e$  has an independent safety probability  $p(e)$ . The *safety* of a path is the product of the safety probabilities of its edges. Design and analyze an algorithm to determine the safest path from a given start vertex  $s$  to a given target vertex  $t$ .



For example, with the probabilities shown above, if Mulder tries to drive directly from Langley to Area 51, he has a 50% chance of getting there without being abducted. If he stops in Memphis, he has a  $0.7 \times 0.9 = 63\%$  chance of arriving safely. If he stops first in Memphis and then in Las Vegas, he has a  $1 - 0.7 \times 0.1 \times 0.5 = 96.5\%$  chance of being abducted! (That's how they got Elvis, you know.)

2. Let  $G = (V, E)$  be a directed graph with weighted edges; edge weights could be positive, negative, or zero. Suppose the vertices of  $G$  are partitioned into  $k$  disjoint subsets  $V_1, V_2, \dots, V_k$ ; that is, every vertex of  $G$  belongs to exactly one subset  $V_i$ . For each  $i$  and  $j$ , let  $\delta(i, j)$  denote the minimum shortest-path distance between any vertex in  $V_i$  and any vertex in  $V_j$ :

$$\delta(i, j) = \min\{\text{dist}(u, v) \mid u \in V_i \text{ and } v \in V_j\}.$$

Describe an algorithm to compute  $\delta(i, j)$  for all  $i$  and  $j$  in time  $O(VE + kE \log E)$ . The output from your algorithm is a  $k \times k$  array.

3. Recall<sup>1</sup> that a deterministic finite automaton (DFA) is formally defined as a tuple  $M = (\Sigma, Q, q_0, F, \delta)$ , where the finite set  $\Sigma$  is the input alphabet, the finite set  $Q$  is the set of states,  $q_0 \in Q$  is the start state,  $F \subseteq Q$  is the set of final (accepting) states, and  $\delta : Q \times \Sigma \rightarrow Q$  is the transition function. Equivalently, a DFA is a directed (multi-)graph with labeled edges, such that each symbol in  $\Sigma$  is the label of exactly one edge leaving any vertex. There is a special ‘start’ vertex  $q_0$ , and a subset of the vertices are marked as ‘accepting states’. Any string in  $\Sigma^*$  describes a unique walk starting at  $q_0$ .

Stephen Kleene<sup>2</sup> proved that the language accepted by any DFA is identical to the language described by some regular expression. This problem asks you to develop a variant of the Floyd-Warshall all-pairs shortest path algorithm that computes a regular expression that is equivalent to the language accepted by a given DFA.

Suppose the input DFA  $M$  has  $n$  states, numbered from 1 to  $n$ , where (without loss of generality) the start state is state 1. Let  $L(i, j, r)$  denote the set of all words that describe walks in  $M$  from state  $i$  to state  $j$ , where every intermediate state lies in the subset  $\{1, 2, \dots, r\}$ ; thus, the language accepted by the DFA is exactly

$$\bigcup_{q \in F} L(1, q, n).$$

Let  $R(i, j, r)$  be a regular expression that describes the language  $L(i, j, r)$ .

- (a) What is the regular expression  $R(i, j, 0)$ ?
- (b) Write a recurrence for the regular expression  $R(i, j, r)$  in terms of regular expressions of the form  $R(i', j', r - 1)$ .
- (c) Describe a polynomial-time algorithm to compute  $R(i, j, n)$  for all states  $i$  and  $j$ . (Assume that you can concatenate two regular expressions in  $O(1)$  time.)

---

<sup>1</sup>No, really, you saw this in CS 273/373.

<sup>2</sup>Pronounced ‘clay knee’, not ‘clean’ or ‘clean-ee’ or ‘clay-nuh’ or ‘dimaggio’.

# CS 473: Undergraduate Algorithms, Spring 2009

## Homework 7

Due Tuesday, April 14, 2009 at 11:59:59pm.

---

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.
- 
1. A graph is *bipartite* if its vertices can be colored black or white such that every edge joins vertices of two different colors. A graph is *d-regular* if every vertex has degree  $d$ . A *matching* in a graph is a subset of the edges with no common endpoints; a matching is *perfect* if it touches every vertex.
    - (a) Prove that every regular bipartite graph contains a perfect matching.
    - (b) Prove that every  $d$ -regular bipartite graph is the union of  $d$  perfect matchings.
  2. Let  $G = (V, E)$  be a directed graph where for each vertex  $v$ , the in-degree of  $v$  and out-degree of  $v$  are equal. Let  $u$  and  $v$  be two vertices  $G$ , and suppose  $G$  contains  $k$  edge-disjoint paths from  $u$  to  $v$ . Under these conditions, must  $G$  also contain  $k$  edge-disjoint paths from  $v$  to  $u$ ? Give a proof or a counterexample with explanation.
  3. A flow  $f$  is called *acyclic* if the subgraph of directed edges with positive flow contains no directed cycles. A flow is *positive* if its value is greater than 0.
    - (a) A *path flow* assigns positive values only to the edges of one simple directed path from  $s$  to  $t$ . Prove that every positive acyclic flow can be written as the sum of a finite number of path flows.
    - (b) Describe a flow in a directed graph that *cannot* be written as the sum of path flows.
    - (c) A *cycle flow* assigns positive values only to the edges of one simple directed cycle. Prove that every flow can be written as the sum of a finite number of path flows and cycle flows.
    - (d) Prove that for any flow  $f$ , there is an acyclic flow with the same value as  $f$ . (In particular, this implies that some maximum flow is acyclic.)

# CS 473: Undergraduate Algorithms, Spring 2009

## Homework 8

Due Tuesday, April 21, 2009 at 11:59:59pm.

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.

- A *cycle cover* of a given directed graph  $G = (V, E)$  is a set of vertex-disjoint cycles that cover all the vertices. Describe and analyze an efficient algorithm to find a cycle cover for a given graph, or correctly report that none exists. [*Hint: Use bipartite matching!*]
- Suppose we are given an array  $A[1..m][1..n]$  of non-negative real numbers. We want to *round*  $A$  to an integer matrix, by replacing each entry  $x$  in  $A$  with either  $\lfloor x \rfloor$  or  $\lceil x \rceil$ , without changing the sum of entries in any row or column of  $A$ . For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

Describe an efficient algorithm that either rounds  $A$  in this fashion, or reports correctly that no such rounding is possible.

- Ad-hoc networks* are made up of cheap, low-powered wireless devices. In principle<sup>1</sup>, these networks can be used on battlefields, in regions that have recently suffered from natural disasters, and in other hard-to-reach areas. The idea is that several simple devices could be distributed randomly in the area of interest (for example, dropped from an airplane), and then they would somehow automatically configure themselves into an efficiently functioning wireless network.

The devices can communicate only within a limited range. We assume all the devices are identical; there is a distance  $D$  such that two devices can communicate if and only if the distance between them is at most  $D$ .

We would like our ad-hoc network to be reliable, but because the devices are cheap and low-powered, they frequently fail. If a device detects that it is likely to fail, it should transmit all its information to some other *backup* device within its communication range. To improve reliability, we require each device  $x$  to have  $k$  potential backup devices, all within distance  $D$  of  $x$ ; we call these  $k$  devices the **backup set** of  $x$ . Also, we do not want any device to be in the backup set of too many other devices; otherwise, a single failure might affect a large fraction of the network.

Suppose we are given the communication distance  $D$ , parameters  $b$  and  $k$ , and an array  $d[1..n, 1..n]$  of distances, where  $d[i, j]$  is the distance between device  $i$  and device  $j$ . Describe and analyze an algorithm that either computes a backup set of size  $k$  for each of the  $n$  devices, such that no device appears in more than  $b$  backup sets, or correctly reports that no good collection of backup sets exists.

<sup>1</sup>but not so much in practice

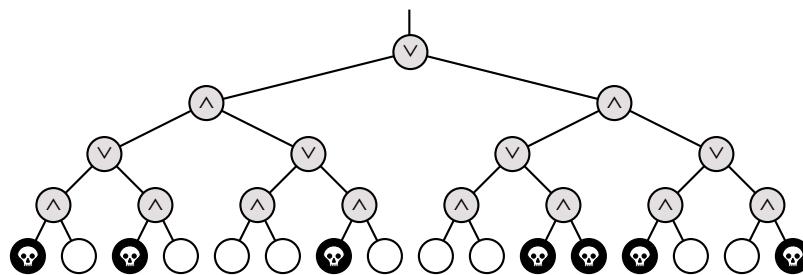
# CS 473: Undergraduate Algorithms, Spring 2009

## Homework 9

Due Tuesday, April 28, 2009 at 11:59:59pm.

- Groups of up to three students may submit a single, common solution for this and all future homeworks. Please clearly write every group member's name and NetID on every page of your submission.

- Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with  $4^n$  leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after  $2n$  moves, when the token lands on a leaf. If the final leaf is black, you die; if it is white, you will live forever. You move first, so Death gets the last turn.



You can decide whether it is worth playing or not as follows. Imagine that the nodes at even levels (where it is your turn) are OR gates, the nodes at odd levels (where it is Death's turn) are AND gates. Each gate gets its input from its children and passes its output to its parent. White and black leaves stand represent TRUE and FALSE inputs, respectively. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

- Describe and analyze a deterministic algorithm to determine whether or not you can win.  
*[Hint: This is easy.]*
  - Prove that *every* deterministic algorithm must examine *every* leaf of the tree in the worst case. Since there are  $4^n$  leaves, this implies that any deterministic algorithm must take  $\Omega(4^n)$  time in the worst case. Use an adversary argument; in other words, assume that Death cheats.
  - [Extra credit]** Describe a *randomized* algorithm that runs in  $O(3^n)$  expected time.
- We say that an array  $A[1..n]$  is *k-sorted* if it can be divided into  $k$  blocks, each of size  $n/k$ , such that the elements in each block are larger than the elements in earlier blocks, and smaller than elements in later blocks. The elements within each block need not be sorted.

For example, the following array is 4-sorted:

1	2	4	3		7	6	8	5		10	11	9	12		15	13	16	14
---	---	---	---	--	---	---	---	---	--	----	----	---	----	--	----	----	----	----

- (a) Describe an algorithm that  $k$ -sorts an arbitrary array in time  $O(n \log k)$ .
- (b) Prove that any comparison-based  $k$ -sorting algorithm requires  $\Omega(n \log k)$  comparisons in the worst case.
- (c) Describe an algorithm that completely sorts an already  $k$ -sorted array in time  $O(n \log(n/k))$ .
- (d) Prove that any comparison-based algorithm to completely sort a  $k$ -sorted array requires  $\Omega(n \log(n/k))$  comparisons in the worst case.

In all cases, you can assume that  $n/k$  is an integer.

3. UIUC has just finished constructing the new Reingold Building, the tallest dormitory on campus. In order to determine how much insurance to buy, the university administration needs to determine the highest safe floor in the building. A floor is considered *safe* if a ~~drunk~~ student *an egg* can fall from a window on that floor and land without breaking; if the egg breaks, the floor is considered *unsafe*. Any floor that is higher than an unsafe floor is also considered unsafe. The only way to determine whether a floor is safe is to drop an egg from a window on that floor.

You would like to find the lowest unsafe floor  $L$  by performing as few tests as possible; unfortunately, you have only a very limited supply of eggs.

- (a) Prove that if you have only one egg, you can find the lowest unsafe floor with  $L$  tests. [*Hint: Yes, this is trivial.*]
- (b) Prove that if you have only one egg, you must perform at least  $L$  tests in the worst case. In other words, prove that your algorithm from part (a) is optimal. [*Hint: Use an adversary argument.*]
- (c) Describe an algorithm to find the lowest unsafe floor using *two* eggs and only  $O(\sqrt{L})$  tests. [*Hint: Ideally, each egg should be dropped the same number of times. How many floors can you test with  $n$  drops?*]
- (d) Prove that if you start with two eggs, you must perform at least  $\Omega(\sqrt{L})$  tests in the worst case. In other words, prove that your algorithm from part (c) is optimal.
- \*(e) [**Extra credit!**] Describe an algorithm to find the lowest unsafe floor using  $k$  eggs, using as few tests as possible, and prove your algorithm is optimal for all values of  $k$ .

# CS 473: Undergraduate Algorithms, Spring 2009

## Head Banging Session 0

January 20 and 21, 2009

---

1. Solve the following recurrences. If base cases are provided, find an *exact* closed-form solution. Otherwise, find a solution of the form  $\Theta(f(n))$  for some function  $f$ .

- **Warmup:** You should be able to solve these almost as fast as you can write down the answers.

- $A(n) = A(n - 1) + 1$ , where  $A(0) = 0$ .
- $B(n) = B(n - 5) + 2$ , where  $B(0) = 17$ .
- $C(n) = C(n - 1) + n^2$
- $D(n) = 3D(n/2) + n^2$
- $E(n) = 4E(n/2) + n^2$
- $F(n) = 5F(n/2) + n^2$

- **Real practice:**

- $A(n) = A(n/3) + 3A(n/5) + A(n/15) + n$
- $B(n) = \min_{0 < k < n} (B(k) + B(n - k) + n)$
- $C(n) = \max_{n/4 < k < 3n/4} (C(k) + C(n - k) + n)$
- $D(n) = \max_{0 < k < n} (D(k) + D(n - k) + k(n - k))$ , where  $D(1) = 0$
- $E(n) = 2E(n - 1) + E(n - 2)$ , where  $E(0) = 1$  and  $E(1) = 2$
- $F(n) = \frac{1}{F(n - 1)F(n - 2)}$ , where  $F(0) = 1$  and  $F(2) = 2$
- \* $G(n) = nG(\sqrt{n}) + n^2$

2. The *Fibonacci numbers*  $F_n$  are defined recursively as follows:  $F_0 = 0$ ,  $F_1 = 1$ , and  $F_n = F_{n-1} + F_{n-2}$  for every integer  $n \geq 2$ . The first few Fibonacci numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ....

Prove that any non-negative integer can be written as the sum of distinct *non-consecutive* Fibonacci numbers. That is, if any Fibonacci number  $F_n$  appears in the sum, then its neighbors  $F_{n-1}$  and  $F_{n+1}$  do not. For example:

$$\begin{aligned} 88 &= 55 + 21 + 8 + 3 + 1 &= F_{10} + F_8 + F_6 + F_4 + F_2 \\ 42 &= 34 + 8 &= F_9 + F_6 \\ 17 &= 13 + 3 + 1 &= F_7 + F_4 + F_2 \end{aligned}$$

3. Whenever groups of pigeons gather, they instinctively establish a *pecking order*. For any pair of pigeons, one pigeon always pecks the other, driving it away from food or potential mates. The same pair of pigeons always chooses the same pecking order, even after years of separation, no matter what other pigeons are around. Surprisingly, the overall pecking order can contain cycles—for example, pigeon A pecks pigeon B, which pecks pigeon C, which pecks pigeon A.

Prove that any finite set of pigeons can be arranged in a row from left to right so that every pigeon pecks the pigeon immediately to its left. Pretty please.

1. An *inversion* in an array  $A[1..n]$  is a pair of indices  $(i, j)$  such that  $i < j$  and  $A[i] > A[j]$ . The number of inversions in an  $n$ -element array is between 0 (if the array is sorted) and  $\binom{n}{2}$  (if the array is sorted backward).

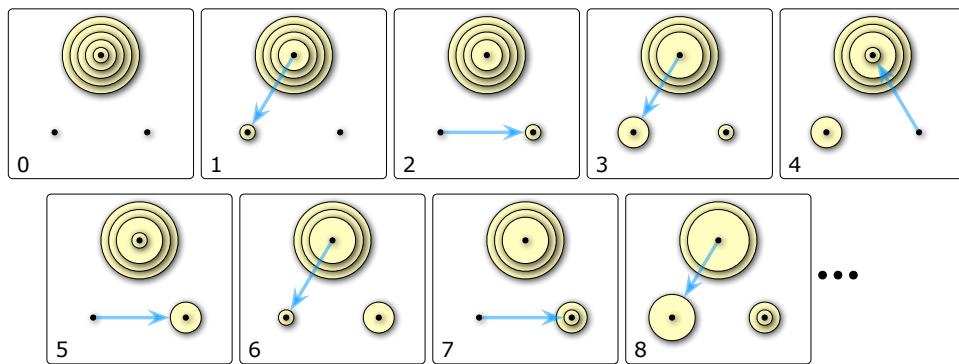
Describe and analyze an algorithm to count the number of inversions in an  $n$ -element array in  $O(n \log n)$  time.

2. (a) Prove that the following algorithm actually sorts its input.

```

STOOGESORT( $A[0..n-1]$ ) :
  if  $n = 2$  and  $A[0] > A[1]$ 
    swap  $A[0] \leftrightarrow A[1]$ 
  else if  $n > 2$ 
     $m = \lceil 2n/3 \rceil$ 
    STOOGESORT( $A[0..m-1]$ )
    STOOGESORT( $A[n-m..n-1]$ )
    STOOGESORT( $A[0..m-1]$ )
  
```

- (b) Would STOOGESORT still sort correctly if we replaced  $m = \lceil 2n/3 \rceil$  with  $m = \lfloor 2n/3 \rfloor$ ? Justify your answer.
- (c) State a recurrence (including base case(s)) for the number of comparisons executed by STOOGESORT.
- (d) Solve this recurrence. [Hint: Ignore the ceiling.]
- (e) **To think about on your own:** Prove that the number of swaps executed by STOOGESORT is at most  $\binom{n}{2}$ .
3. Consider the following restricted variants of the Tower of Hanoi puzzle. In each problem, the needles are numbered 0, 1, and 2, and your task is to move a stack of  $n$  disks from needle 1 to needle 2.
- (a) Suppose you are forbidden to move any disk directly between needle 1 and needle 2; every move must involve needle 0. Describe an algorithm to solve this version of the puzzle using as few moves as possible. Exactly how many moves does your algorithm make?
- (b) Suppose you are only allowed to move disks from needle 0 to needle 2, from needle 2 to needle 1, or from needle 1 to needle 0. Equivalently, Suppose the needles are arranged in a circle and numbered in clockwise order, and you are only allowed to move disks counterclockwise. Describe an algorithm to solve this version of the puzzle using as few moves as possible. Exactly how many moves does your algorithm make?



The first eight moves in a counterclockwise Towers of Hanoi solution

- ★(c) Finally, suppose you are forbidden to move any disk directly from needle 1 to 2, but any other move is allowed. Describe an algorithm to solve this version of the puzzle using as few moves as possible. *Exactly* how many moves does your algorithm make?

*[Hint: This version is considerably harder than the other two.]*

## CS 473: Undergraduate Algorithms, Spring 2009

### HBS 10

---

1. Consider the following problem, called *BOX-DEPTH*: Given a set of  $n$  axis-aligned rectangles in the plane, how big is the largest subset of these rectangles that contain a common point?
  - (a) Describe a polynomial-time reduction from *BOX-DEPTH* to *MAX-CLIQUE*.
  - (b) Describe and analyze a polynomial-time algorithm for *BOX-DEPTH*. [Hint:  $O(n^3)$  time should be easy, but  $O(n \log n)$  time is possible.]
  - (c) Why don't these two results imply that  $P = NP$ ?
2. Suppose you are given a magic black box that can determine in polynomial time, given an arbitrary weighted graph  $G$ , the length of the shortest Hamiltonian cycle in  $G$ . Describe and analyze a polynomial-time algorithm that computes, given an arbitrary weighted graph  $G$ , the shortest Hamiltonian cycle in  $G$ , using this magic black box as a subroutine.
3. Prove that the following problems are NP-complete.
  - (a) Given an undirected graph  $G$ , does  $G$  have a spanning tree in which every node has degree at most 17?
  - (b) Given an undirected graph  $G$ , does  $G$  have a spanning tree with at most 42 leaves?

## CS 473: Undergraduate Algorithms, Spring 2009

### HBS 11

---

1. You step in a party with a camera in your hand. Each person attending the party has some friends there. You want to have exactly one picture of each person in your camera. You want to use the following protocol to collect photos. At each step, the person that has the camera in his hand takes a picture of one of his/her friends and pass the camera to him/her. Of course, you only like the solution if it finishes when the camera is in your hand. Given the friendship matrix of the people in the party, design a polynomial algorithm that decides whether this is possible, or prove that this decision problem is NP-hard.
2. A boolean formula is in disjunctive normal form (DNF) if it is a disjunctions (OR) of several clauses, each of which is the conjunction (AND) of several literals, each of which is either a variable or its negation. For example:

$$(a \wedge b \wedge c) \vee (\bar{a} \wedge b) \vee (\bar{c} \wedge x)$$

Given a DNF formula give a polynomial algorithm to check whether it is satisfiable or not. Why this does not imply  $P = NP$ .

3. Prove that the following problems are NP-complete.
  - (a) Given an undirected graph  $G$ , does  $G$  have a spanning tree in which every node has degree at most 17?
  - (b) Given an undirected graph  $G$ , does  $G$  have a spanning tree with at most 42 leaves?

## CS 473: Undergraduate Algorithms, Spring 2009

### HBS 2

---

1. Consider two horizontal lines  $l_1$  and  $l_2$  in the plane. There are  $n$  points on  $l_1$  with  $x$ -coordinates  $A = a_1, a_2, \dots, a_n$  and there are  $n$  points on  $l_2$  with  $x$ -coordinates  $B = b_1, b_2, \dots, b_n$ . Design an algorithm to compute, given  $A$  and  $B$ , a largest set  $S$  of non-intersecting line segments subject to the following restrictions:
  - (a) Any segment in  $S$  connects  $a_i$  to  $b_i$  for some  $i (1 \leq i \leq n)$ .
  - (b) Any two segments in  $S$  do not intersect.
2. Consider a  $2^n \times 2^n$  chess board with one (arbitrarily chosen) square removed. Prove that any such chessboard can be tiled without gaps or overlaps by L-shaped pieces of 3 squares each. Can you give an algorithm to do the tiling?
3. Given a string of letters  $Y = y_1 y_2 \dots y_n$ , a segmentation of  $Y$  is a partition of its letters into contiguous blocks of letters (also called words). Each word has a quality that can be computed by a given oracle (e.g. you can call `quality("meet")` to get the quality of the word "meet"). The quality of a segmentation is equal to the sum over the qualities of its words. Each call to the oracle takes linear time in terms of the argument; that is `quality( $S$ )` takes  $O(|S|)$ .

Using the given oracle, give an algorithm that takes a string  $Y$  and computes a segmentation of maximum total quality.

## CS 473: Undergraduate Algorithms, Spring 2009

### HBS 3

---

1. Change your recursive solutions for the following problems to efficient algorithms (Hint: use dynamic programming!).
  - (a) Consider two horizontal lines  $l_1$  and  $l_2$  in the plane. There are  $n$  points on  $l_1$  with  $x$ -coordinates  $A = a_1, a_2, \dots, a_n$  and there are  $n$  points on  $l_2$  with  $x$ -coordinates  $B = b_1, b_2, \dots, b_n$ . Design an algorithm to compute, given  $A$  and  $B$ , a largest set  $S$  of non-intersecting line segments subject to the following restrictions:
    - i. Any segment in  $S$  connects  $a_i$  to  $b_i$  for some  $i$  ( $1 \leq i \leq n$ ).
    - ii. Any two segments in  $S$  do not intersect.
  - (b) Given a string of letters  $Y = y_1y_2\dots y_n$ , a segmentation of  $Y$  is a partition of its letters into contiguous blocks of letters (also called words). Each word has a quality that can be computed by a given oracle (e.g. you can call *quality("meet")* to get the quality of the word "meet"). The quality of a segmentation is equal to the sum over the qualities of its words. Each call to the oracle takes linear time in terms of the argument; that is *quality*( $S$ ) takes  $O(|S|)$ .Using the given oracle, give an algorithm that takes a string  $Y$  and computes a segmentation of maximum total quality.
2. Give a polynomial time algorithm which given two strings  $A$  and  $B$  returns the longest sequence  $S$  that is a subsequence of  $A$  and  $B$ .
3. Consider a rooted tree  $T$ . Assume the root has a message to send to all nodes. At the beginning only the root has the message. If a node has the message, it can forward it to one of its children at each time step. Design an algorithm to find the minimum number of time steps required for the message to be delivered to all nodes.

## CS 473: Undergraduate Algorithms, Spring 2009

### HBS 3.5

---

1. Say you are given  $n$  jobs to run on a machine. Each job has a start time and an end time. If a job is chosen to be run, then it must start at its start time and end at its end time. Your goal is to accept as many jobs as possible, regardless of the job lengths, subject to the constraint that the processor can run at most one job at any given point in time. Provide an algorithm to do this with a running time that is polynomial in  $n$ . You may assume for simplicity that no two jobs have the same start or end times, but the start time and end time of two jobs can overlap.
2. Describe and analyze an algorithm that chooses one element uniformly at random from a data stream, without knowing the length of the stream in advance. Your algorithm should spend  $O(1)$  time per stream element and use  $O(1)$  space (not counting the stream itself).
3. Design and analyze an algorithm that return a permutation of the integers  $\{1, 2, \dots, n\}$  chosen uniformly at random.

## CS 473: Undergraduate Algorithms, Spring 2009

### HBS 4

---

1. Let  $x$  and  $y$  be two elements of a set  $S$  whose ranks differ by exactly  $r$ . Prove that in a treap for  $S$ , the expected length of the unique path from  $x$  to  $y$  is  $O(\log r)$
2. Consider the problem of making change for  $n$  cents using the least number of coins.
  - (a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.
  - (b) Suppose that the available coins have the values  $c^0, c^1, \dots, c^k$  for some integers  $c > 1$  and  $k \geq 1$ . Show that the greedy algorithm always yields an optimal solution.
  - (c) Give a set of 4 coin values for which the greedy algorithm does not yield an optimal solution, show why.
  - (d) Give a dynamic programming algorithm that yields an optimal solution for an arbitrary set of coin values.
3. A heater is a sort of dual treap, in which the priorities of the nodes are given, but their search keys are generated independently and uniformly from the unit interval  $[0,1]$ . You can assume all priorities and keys are distinct. Describe algorithms to perform the operations INSERT and DELETEMIN in a heater. What are the expected worst-case running times of your algorithms? In particular, can you express the expected running time of INSERT in terms of the priority rank of the newly inserted item?

## CS 473: Undergraduate Algorithms, Spring 2009

### HBS 5

---

1. Recall that the staircase of a set of points consists of the points with no other point both above and to the right. Describe a method to maintain the staircase as new points are added to the set. Specifically, describe and analyze a data structure that stores the staircase of a set of points, and an algorithm  $INSERT(x, y)$  that adds the point  $(x, y)$  to the set and returns  $TRUE$  or  $FALSE$  to indicate whether the staircase has changed. Your data structure should use  $O(n)$  space, and your  $INSERT$  algorithm should run in  $O(\log n)$  amortized time.
2. In some applications, we do not know in advance how much space we will require. So, we start the program by allocating a (dynamic) table of some fixed size. Later, as new objects are inserted, we may have to allocate a larger table and copy the previous table to it. So, we may need more than  $O(1)$  time for copying. In addition, we want to keep the table size small enough, avoiding a very large table to keep only few items. One way to manage a dynamic table is by the following rules:
  - (a) Double the size of the table if an item is inserted into a full table
  - (b) Halve the table size if a deletion causes the table to become less than 1/4 full

Show that, in such a dynamic table we only need  $O(1)$  amortized time, per operation.

3. Consider a stack data structure with the following operations:

- $PUSH(x)$ : adds the element  $x$  to the top of the stack
- $POP$ : removes and returns the element that is currently on top of the stack (if the stack is non-empty)
- $SEARCH(x)$ : repeatedly removes the element on top of the stack until  $x$  is found or the stack becomes empty

What is the amortized cost of an operation?

## CS 473: Undergraduate Algorithms, Spring 2009

### HBS 6

---

1. Let  $G$  be a connected graph and let  $v$  be a vertex in  $G$ . Show that  $T$  is both a DFS tree and a BFS tree rooted at  $v$ , then  $G = T$ .
2. An Euler tour of a graph  $G$  is a walk that starts from a vertex  $v$ , visits every edge of  $G$  exactly once and gets back to  $v$ . Prove that  $G$  has an Euler tour if and only if all the vertices of  $G$  has even degrees. Can you give an efficient algorithm to find an Euler tour of such a graph.
3. You are helping a group of ethnographers analyze some oral history data they have collected by interviewing members of a village to learn about the lives of people lived there over the last two hundred years. From the interviews, you have learned about a set of people, all now deceased, whom we will denote  $P_1, P_2, \dots, P_n$ . The ethnographers have collected several facts about the lifespans of these people, of one of the following forms:
  - (a)  $P_i$  died before  $P_j$  was born.
  - (b)  $P_i$  and  $P_j$  were both alive at some moment.

Naturally, the ethnographers are not sure that their facts are correct; memories are not so good, and all this information was passed down by word of mouth. So they'd like you to determine whether the data they have collected is at least internally consistent, in the sense that there could have existed a set of people for which all the facts they have learned simultaneously hold.

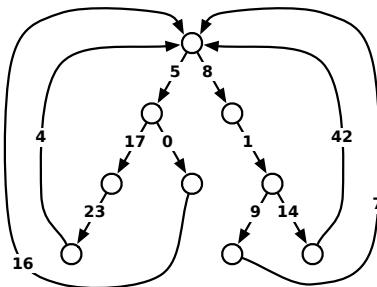
Describe and analyze an algorithm to answer the ethnographers' problem. Your algorithm should either output possible dates of birth and death that are consistent with all the stated facts, or it should report correctly that no such dates exist.

# CS 473: Undergraduate Algorithms, Spring 2009

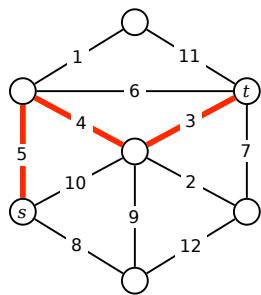
## HBS 6.5

---

1. (a) Describe and analyze an algorithm to find the *second smallest spanning tree* of a given graph  $G$ , that is, the spanning tree of  $G$  with smallest total weight except for the minimum spanning tree.
- \*(b) Describe and analyze an efficient algorithm to compute, given a weighted undirected graph  $G$  and an integer  $k$ , the  $k$  smallest spanning trees of  $G$ .
2. A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has a non-negative weight.



- (a) How much time would Dijkstra's algorithm require to compute the shortest path between two vertices  $u$  and  $v$  in a looped tree with  $n$  nodes?
- (b) Describe and analyze a faster algorithm.
3. Consider a path between two vertices  $s$  and  $t$  in an undirected weighted graph  $G$ . The *bottleneck length* of this path is the maximum weight of any edge in the path. The *bottleneck distance* between  $s$  and  $t$  is the minimum bottleneck length of any path from  $s$  to  $t$ . (If there are no paths from  $s$  to  $t$ , the bottleneck distance between  $s$  and  $t$  is  $\infty$ )



The bottleneck distance between  $s$  and  $t$  is 5.

Describe and analyze an algorithm to compute the bottleneck distance between *every* pair of vertices in an arbitrary undirected weighted graph. Assume that no two edges have the same weight.

## CS 473: Undergraduate Algorithms, Spring 2009

### HBS 6.55

---

1. Suppose you are given a directed graph  $G = (V, E)$  with non-negative edge lengths;  $\ell(e)$  is the length of  $e \in E$ . You are interested in the shortest path distance between two given locations/nodes  $s$  and  $t$ . It has been noticed that the existing shortest path distance between  $s$  and  $t$  in  $G$  is not satisfactory and there is a proposal to add exactly one edge to the graph to improve the situation. The candidate edges from which one has to be chosen is given by  $E' = \{e_1, e_2, \dots, e_k\}$  and you can assume that  $E \cup E' = \emptyset$ . The length of the  $e_i$  is  $\alpha_i \geq 0$ . Your goal is figure out which of these  $k$  edges will result in the most reduction in the shortest path distance from  $s$  to  $t$ . Describe an algorithm for this problem that runs in time  $O((m + n) \log n + k)$  where  $m = |E|$  and  $n = |V|$ . Note that one can easily solve this problem in  $O(k(m + n) \log n)$  by running Dijkstra's algorithm  $k$  times, one for each  $G_i$  where  $G_i$  is the graph obtained by adding  $e_i$  to  $G$ .
2. Let  $G$  be an undirected graph with non-negative edge weights. Let  $s$  and  $t$  be two vertices such that the shortest path between  $s$  and  $t$  in  $G$  contains all the vertices in the graph. For each edge  $e$ , let  $G \setminus e$  be the graph obtained from  $G$  by deleting the edge  $e$ . Design an  $O(E \log V)$  algorithm that finds the shortest path distance between  $s$  and  $t$  in  $G \setminus e$  for all  $e$ . [Note that you need to output  $E$  distances, one for each graph  $G \setminus e$ ]
3. Given a Directed Acyclic Graph (DAG) and two vertices  $s$  and  $t$  you want to determine if there is an  $s$  to  $t$  path that includes at least  $k$  vertices.

## CS 473: Undergraduate Algorithms, Spring 2009

### HBS 7

---

1. Let  $G = (V, E)$  be a directed graph with non-negative capacities. Give an efficient algorithm to check whether there is a unique max-flow on  $G$ ?
2. Let  $G = (V, E)$  be a graph and  $s, t \in V$  be two specific vertices of  $G$ . We call  $(S, T = V \setminus S)$  an  $(s, t)$ -cut if  $s \in S$  and  $t \in T$ . Moreover, it is a minimum cut if the sum of the capacities of the edges that have one endpoint in  $S$  and one endpoint in  $T$  equals the maximum  $(s, t)$ -flow. Show that, both intersection and union of two min-cuts is a min-cut itself.
3. Let  $G = (V, E)$  be a graph. For each edge  $e$  let  $d(e)$  be a demand value attached to it. A flow is feasible if it sends more than  $d(e)$  through  $e$ . Assume you have an oracle that is capable of solving the maximum flow problem. Give efficient algorithms for the following problems that call the oracle at most once.
  - (a) Find a feasible flow.
  - (b) Find a feasible flow of minimum possible value.

## CS 473: Undergraduate Algorithms, Spring 2009

### HBS 8

---

1. A box  $i$  can be specified by the values of its sides, say  $(i_1, i_2, i_3)$ . We know all the side lengths are larger than 10 and smaller than 20 (i.e.  $10 < i_1, i_2, i_3 < 20$ ). Geometrically, you know what it means for one box to nest in another: It is possible if you can rotate the smaller so that it fits inside the larger in each dimension. Of course, nesting is recursive, that is if  $i$  nests in  $j$  and  $j$  nests in  $k$  then  $i$  nests in  $k$ . After doing some nesting operations, we say a box is visible if it is not nested in any other one. Given a set of boxes (each specified by the lengths of their sides) the goal is to find a set of nesting operations to minimize the number of visible boxes. Design and analyze an efficient algorithm to do this.
  
2. Let the number of papers submitted to a conference be  $n$  and the number of available reviewers be  $m$ . Each reviewer has a list of papers that he/she can review and each paper should be reviewed by three different persons. Also, each reviewer can review at most 5 papers. Design and analyze an algorithm to make the assignment or decide no feasible assignment exists.
  
3. Back in the euphoric early days of the Web, people liked to claim that much of the enormous potential in a company like Yahoo! was in the "eyeballs" - the simple fact that it gets millions of people looking at its pages every day. And further, by convincing people to register personal data with the site, it can show each user an extremely targeted advertisement whenever he or she visits the site, in away that TV networks or magazines could not hope to match. So if the user has told Yahoo! that he is a 20-year old computer science major from Cornell University, the site can throw up a banner ad for apartments in Ithaca, NY; on the other hand, if he is a 50-year-old investment banker from Greenwich, Connecticut, the site can display a banner ad pitching Lincoln Town Cars instead.

But deciding on which ads to show to which people involves some serious computation behind the scenes. Suppose that the managers of a popular Web site have identified  $k$  distinct demographic groups  $G_1, G_2, \dots, G_k$ . (These groups can overlap; for example  $G_1$  can be equal to all residents of New York State, and  $G_2$  can be equal to all people with a degree in computer science.) The site has contracts with  $m$  different advertisers, to show a certain number of copies of their ads to users of the site. Here is what the contract with the  $i^{th}$  advertiser looks like:

- (a) For a subset  $X_i \subset \{G_1, \dots, G_k\}$  of the demographic groups, advertiser  $i$  wants its ads shown only to users who belong to at least one of the demographic groups in the set  $X_i$
- (b) For a number  $r_i$ , advertiser  $i$  wants its ads shown to at least  $r_i$  users each minute.

Now, consider the problem of designing a good advertising policy - a way to show a single ad to each user of the site. Suppose at a given minute, there are  $n$  users visiting the site. Because we have registration information on each of these users, we know that user  $j$  (for  $j = 1, 2, \dots, n$ ) belongs to a subset  $U_j \subset \{G_1, \dots, G_k\}$  of the demographic groups. The problem is: is there a way to show a single ad to each user so that the site's contracts with each of the  $m$  advertisers is satisfied for this minute? (That is, for each  $i = 1, 2, \dots, m$ , at least  $r_i$  of the  $n$  users, each belonging to at least one demographic group in  $X_i$ , are shown an ad provided by advertiser  $i$ .)

Give an efficient algorithm to decide if this is possible, and if so, to actually choose an ad to show each user.

## CS 473: Undergraduate Algorithms, Spring 2009

### HBS 9

---

1. Prove that any algorithm to merge two sorted arrays, each of size  $n$ , requires at least  $2n - 1$  comparisons.
2. Suppose you want to determine the largest number in an  $n$ -element set  $X = \{x_1, x_2, \dots, x_n\}$ , where each element  $x_i$  is an integer between 1 and  $2^m - 1$ . Describe an algorithm that solves this problem in  $O(n + m)$  steps, where at each step, your algorithm compares one of the elements  $x_i$  with a constant. In particular, your algorithm must never actually compare two elements of  $X$ !  
*[Hint: Construct and maintain a nested set of ‘pinning intervals’ for the numbers that you have not yet removed from consideration, where each interval but the largest is either the upper half or lower half of the next larger block.]*
3. Let  $P$  be a set of  $n$  points in the plane. The staircase of  $P$  is the set of all points in the plane that have at least one point in  $P$  both above and to the right. Prove that computing the staircase requires at least  $\Omega(n \log n)$  comparisons in two ways,
  - (a) Reduction from sorting.
  - (b) Directly.

You have 90 minutes to answer four of the five questions.  
**Write your answers in the separate answer booklet.**  
 You may take the question sheet with you when you leave.

1. Each of these ten questions has one of the following five answers:

$$\text{A: } \Theta(1) \quad \text{B: } \Theta(\log n) \quad \text{C: } \Theta(n) \quad \text{D: } \Theta(n \log n) \quad \text{E: } \Theta(n^2)$$

Choose the correct answer for each question. Each correct answer is worth +1 point; each incorrect answer is worth  $-1/2$  point; each "I don't know" is worth  $+1/4$  point. Your score will be rounded to the nearest *non-negative* integer.

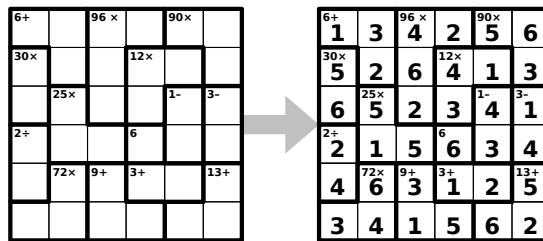
- (a) What is  $\sum_{i=1}^n \frac{n}{i}$ ?
- (b) What is  $\sqrt{\sum_{i=1}^n i}$  ?
- (c) How many digits are required to write  $3^n$  in decimal?
- (d) What is the solution to the recurrence  $D(n) = D(n/\pi) + \sqrt{2}$ ?
- (e) What is the solution to the recurrence  $E(n) = E(n - \sqrt{2}) + \pi$ ?
- (f) What is the solution to the recurrence  $F(n) = 4F(n/2) + 3n$ ?
- (g) What is the worst-case time to search for an item in a binary search tree?
- (h) What is the worst-case running time of quicksort?
- (i) Let  $H[1..n, 1..n]$  be a fixed array of numbers. Consider the following recursive function:

$$Glub(i, j) = \begin{cases} 0 & \text{if } i = 0 \\ \infty & \text{if } i > n \text{ or } j = 0 \\ \max \{Glub(i - 1, j), H[i, j] + Glub(i + 1, j - 1)\} & \text{otherwise} \end{cases}$$

How long does it take to compute  $Glub(n, n)$  using dynamic programming?

- (j) What is the running time of the fastest possible algorithm to solve KenKen puzzles?

A KenKen puzzle is a  $6 \times 6$  grid, divided into regions called *cages*. Each cage is labeled with a numerical *value* and an arithmetic *operation*:  $+$ ,  $-$ ,  $\times$ , or  $\div$ . (The operation can be omitted if the cage consists of a single cell.) The goal is to place an integer between 1 and 6 in each grid cell, so that no number appears twice in any row or column, and the numbers inside each cage can be combined using *only* that cage's operation to obtain that cage's value. The solution is guaranteed to be unique.



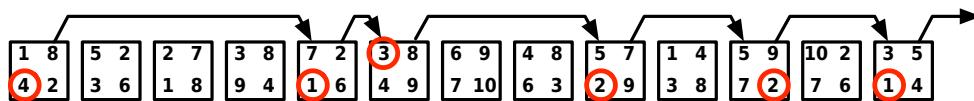
6+		96 ×		90 ×	
30x			12x		
	25x			1-	3-
2+			6		
	72x	9+	3+		13+

1	3	4	2	5	6
5	2	6	4	1	3
6	5	2	3	4	1
2	1	5	6	3	4
4	6	3	1	2	5
3	4	1	5	6	2

A Kenken puzzle and its solution

2. (a) Suppose  $A[1..n]$  is an array of  $n$  distinct integers, sorted so that  $A[1] < A[2] < \dots < A[n]$ . Each integer  $A[i]$  could be positive, negative, or zero. Describe an efficient algorithm that either computes an index  $i$  such that  $A[i] = i$  or correctly reports that no such index exists. An algorithm that runs in  $\Theta(n)$  time is worth at most 3 points.
- (b) Now suppose  $A[1..n]$  is a sorted array of  $n$  distinct **positive** integers. Describe an even faster algorithm that either computes an index  $i$  such that  $A[i] = i$  or correctly reports that no such index exists. [Hint: This is **really** easy!]
3. *Moby Selene* is a solitaire game played on a row of  $n$  squares. Each square contains four positive integers. The player begins by placing a token on the leftmost square. On each move, the player chooses one of the numbers on the token's current square, and then moves the token that number of squares to the right. The game ends when the token moves past the rightmost square. The object of the game is to make as many moves as possible before the game ends.



A Moby Selene puzzle that allows six moves. (This is **not** the longest legal sequence of moves.)

- (a) **Prove** that the obvious greedy strategy (always choose the smallest number) does not give the largest possible number of moves for every Moby Selene puzzle.
- (b) Describe and analyze an efficient algorithm to find the largest possible number of legal moves for a given Moby Selene puzzle.
4. Consider the following algorithm for finding the largest element in an unsorted array:

```
RANDOMMAX( $A[1..n]$ ):
   $max \leftarrow \infty$ 
  for  $i \leftarrow 1$  to  $n$  in random order
    if  $A[i] > max$ 
       $max \leftarrow A[i]$  (*)
```

return  $max$

- (a) In the worst case, how many times does RANDOMMAX execute line (\*)?
- (b) What is the **exact** probability that line (\*) is executed during the last iteration of the for loop?
- (c) What is the **exact** expected number of executions of line (\*)? (A correct  $\Theta()$  bound is worth half credit.)
5. *This question is taken directly from HBS 0.* Whenever groups of pigeons gather, they instinctively establish a *pecking order*. For any pair of pigeons, one pigeon always pecks the other, driving it away from food or potential mates. The same pair of pigeons always chooses the same pecking order, even after years of separation, no matter what other pigeons are around. Surprisingly, the overall pecking order can contain cycles—for example, pigeon A pecks pigeon B, which pecks pigeon C, which pecks pigeon A.

**Prove** that any finite set of pigeons can be arranged in a row from left to right so that every pigeon pecks the pigeon immediately to its left. Pretty please.

You have 90 minutes to answer four of the five questions.  
**Write your answers in the separate answer booklet.**  
You may take the question sheet with you when you leave.

1. Recall that a *tree* is a connected graph with no cycles. A graph is *bipartite* if we can color its vertices black and white, so that every edge connects a white vertex to a black vertex.
  - (a) **Prove** that every tree is bipartite.
  - (b) Describe and analyze a fast algorithm to determine whether a given graph is bipartite.
2. Describe and analyze an algorithm  $\text{SHUFFLE}(A[1..n])$  that randomly permutes the input array  $A$ , so that each of the  $n!$  possible permutations is equally likely. You can assume the existence of a subroutine  $\text{RANDOM}(k)$  that returns a random integer chosen uniformly between 1 and  $k$  in  $O(1)$  time. For full credit, your  $\text{SHUFFLE}$  algorithm should run in  $O(n)$  time. [Hint: This problem appeared in HBS 3½.]
3. Let  $G$  be an undirected graph with weighted edges.
  - (a) Describe and analyze an algorithm to compute the *maximum weight spanning tree* of  $G$ .
  - (b) A *feedback edge set* of  $G$  is a subset  $F$  of the edges such that every cycle in  $G$  contains at least one edge in  $F$ . In other words, removing every edge in  $F$  makes  $G$  acyclic. Describe and analyze a fast algorithm to compute the minimum weight feedback edge set of  $G$ .

[Hint: Don't reinvent the wheel!]
4. Let  $G = (V, E)$  be a connected directed graph with non-negative edge weights, let  $s$  and  $t$  be vertices of  $G$ , and let  $H$  be a subgraph of  $G$  obtained by deleting some edges. Suppose we want to reinsert exactly one edge from  $G$  back into  $H$ , so that the shortest path from  $s$  to  $t$  in the resulting graph is as short as possible. Describe and analyze an algorithm to choose the best edge to reinsert. For full credit, your algorithm should run in  $O(E \log V)$  time. [Hint: This problem appeared in HBS 6¾.]
5. Describe and analyze an efficient data structure to support the following operations on an array  $X[1..n]$  as quickly as possible. Initially,  $X[i] = 0$  for all  $i$ .
  - Given an index  $i$  such that  $X[i] = 0$ , set  $X[i]$  to 1.
  - Given an index  $i$ , return  $X[i]$ .
  - Given an index  $i$ , return the smallest index  $j \geq i$  such that  $X[j] = 0$ , or report that no such index exists.

For full credit, the first two operations should run in *worst-case constant* time, and the amortized cost of the third operation should be as small as possible.

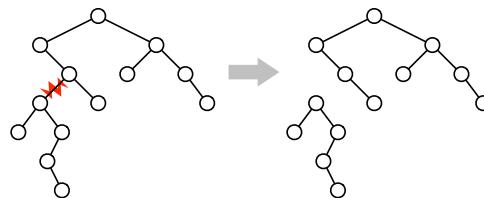
You have 180 minutes to answer six of the seven questions.  
 Write your answers in the separate answer booklet.  
 You may take the question sheet with you when you leave.

1. SUBSETSUM and PARTITION are two closely related NP-hard problems, defined as follows.

**SUBSETSUM:** Given a set  $X$  of positive integers and a positive integer  $k$ , does  $X$  have a subset whose elements sum up to  $k$ ?

**PARTITION:** Given a set  $Y$  of positive integers, can  $Y$  be partitioned into two subsets whose sums are equal?

- (a) [2 pts] **Prove** that PARTITION and SUBSETSUM are both in NP.
  - (b) [1 pt] Suppose you already know that SUBSETSUM is NP-hard. Which of the following arguments could you use to prove that PARTITION is NP-hard? **You do not need to justify your answer** — just answer ① or ②.
    - ① Given a set  $X$  and an integer  $k$ , construct a set  $Y$  in polynomial time, such that  $\text{PARTITION}(Y)$  is true if and only if  $\text{SUBSETSUM}(X, k)$  is true.
    - ② Given a set  $Y$ , construct a set  $X$  and an integer  $k$  in polynomial time, such that  $\text{PARTITION}(Y)$  is true if and only if  $\text{SUBSETSUM}(X, k)$  is true.
  - (c) [3 pts] Describe and analyze a polynomial-time reduction from PARTITION to SUBSETSUM. **You do not need to prove that your reduction is correct.**
  - (d) [4 pts] Describe and analyze a polynomial-time reduction from SUBSETSUM to PARTITION. **You do not need to prove that your reduction is correct.**
2. (a) [4 pts] For any node  $v$  in a binary tree, let  $\text{size}(v)$  denote the number of nodes in the subtree rooted at  $v$ . Let  $k$  be an arbitrary positive number. **Prove** that every binary tree with at least  $k$  nodes contains a node  $v$  such that  $k \leq \text{size}(v) \leq 2k$ .
- (b) [2 pts] Removing any edge from an  $n$ -node binary tree  $T$  separates it into two smaller binary trees. An edge is called a **balanced separator** if both of these subtrees have at least  $n/3$  nodes (and therefore at most  $2n/3$  nodes). **Prove** that every binary tree with more than one node has a balanced separator. [Hint: Use part (a).]
- (c) [4 pts] Describe and analyze an algorithm to find a balanced separator in a given binary tree. [Hint: Use part (a).]



Removing a balanced separator from a binary tree.

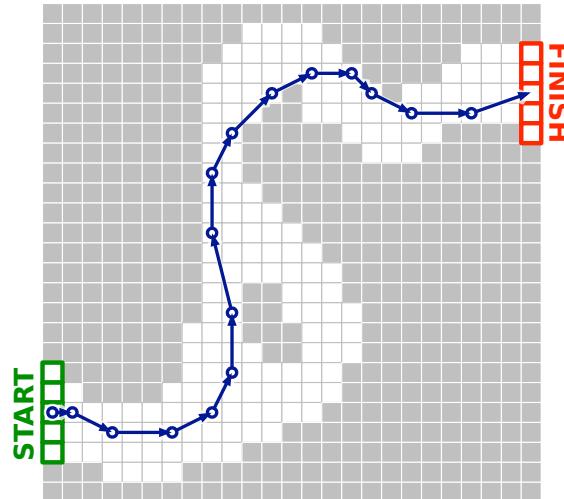
3. **Racetrack** (also known as *Graph Racers* and *Vector Rally*) is a two-player paper-and-pencil racing game that Jeff played on the bus in 5th grade.<sup>1</sup> The game is played with a track drawn on a sheet of graph paper. The players alternately choose a sequence of grid points that represent the motion of a car around the track, subject to certain constraints explained below.

Each car has a *position* and a *velocity*, both with integer *x*- and *y*-coordinates. The initial position is a point on the starting line, chosen by the player; the initial velocity is always  $(0, 0)$ . At each step, the player optionally increments or decrements either or both coordinates of the car's velocity; in other words, each component of the velocity can change by at most 1 in a single step. The car's new position is then determined by adding the new velocity to the car's previous position. The new position must be inside the track; otherwise, the car crashes and that player loses the race. The race ends when the first car reaches a position on the finish line.

Suppose the racetrack is represented by an  $n \times n$  array of bits, where each 0 bit represents a grid point inside the track, each 1 bit represents a grid point outside the track, the 'starting line' is the first column, and the 'finish line' is the last column.

Describe and analyze an algorithm to find the minimum number of steps required to move a car from the starting line to the finish line of a given racetrack. [Hint: Build a graph. What are the vertices? What are the edges? What problem is this?]

velocity	position
$(0, 0)$	$(1, 5)$
$(1, 0)$	$(2, 5)$
$(2, -1)$	$(4, 4)$
$(3, 0)$	$(7, 4)$
$(2, 1)$	$(9, 5)$
$(1, 2)$	$(10, 7)$
$(0, 3)$	$(10, 10)$
$(-1, 4)$	$(9, 14)$
$(0, 3)$	$(9, 17)$
$(1, 2)$	$(10, 19)$
$(2, 2)$	$(12, 21)$
$(2, 1)$	$(14, 22)$
$(2, 0)$	$(16, 22)$
$(1, -1)$	$(17, 21)$
$(2, -1)$	$(19, 20)$
$(3, 0)$	$(22, 20)$
$(3, 1)$	$(25, 21)$



A 16-step Racetrack run, on a  $25 \times 25$  track.

4. A palindrome is any string that is exactly the same as its reversal, like I, or DEED, or RACECAR, or AMANAPLANACATACANALPANAMA. Describe and analyze an algorithm to find the length of the longest *subsequence* of a given string that is also a palindrome.

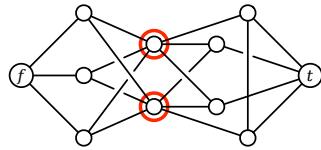
For example, the longest palindrome subsequence of MAHDYNAMICPROGRAMZLETMESHOWYOUTHEM is MHYMRORMYHM, so given that string as input, your algorithm should output the number 11.

<sup>1</sup>The actual game is a bit more complicated than the version described here.

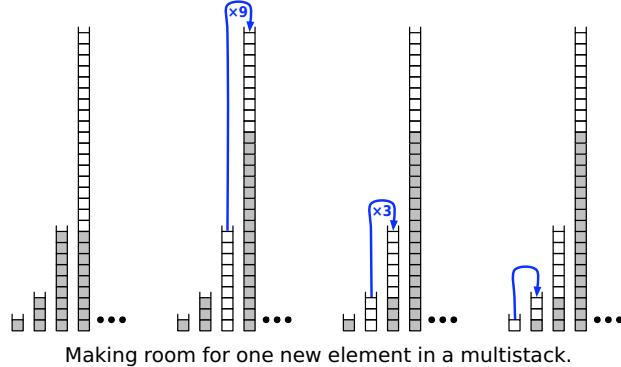
5. The Island of Sodor is home to a large number of towns and villages, connected by an extensive rail network. Recently, several cases of a deadly contagious disease (either swine flu or zombies; reports are unclear) have been reported in the village of Ffarquhar. The controller of the Sodor railway plans to close down certain railway stations to prevent the disease from spreading to Tidmouth, his home town. No trains can pass through a closed station. To minimize expense (and public notice), he wants to close down as few stations as possible. However, he cannot close the Ffarquhar station, because that would expose him to the disease, and he cannot close the Tidmouth station, because then he couldn't visit his favorite pub.

Describe and analyze an algorithm to find the minimum number of stations that must be closed to block all rail travel from Ffarquhar to Tidmouth. The Sodor rail network is represented by an undirected graph, with a vertex for each station and an edge for each rail connection between two stations. Two special vertices  $f$  and  $t$  represent the stations in Ffarquhar and Tidmouth.

For example, given the following input graph, your algorithm should return the number 2.



6. A *mystack* consists of an infinite series of stacks  $S_0, S_1, S_2, \dots$ , where the  $i$ th stack  $S_i$  can hold up to  $3^i$  elements. Whenever a user attempts to push an element onto any full stack  $S_i$ , we first pop all the elements off  $S_i$  and push them onto stack  $S_{i+1}$  to make room. (Thus, if  $S_{i+1}$  is already full, we first recursively move all its members to  $S_{i+2}$ .) Moving a single element from one stack to the next takes  $O(1)$  time.



- (a) In the worst case, how long does it take to push one more element onto a mystack containing  $n$  elements?
- (b) **Prove** that the amortized cost of a push operation is  $O(\log n)$ , where  $n$  is the maximum number of elements in the mystack.
7. Recall the problem 3COLOR: Given a graph, can we color each vertex with one of 3 colors, so that every edge touches two different colors? We proved in class that 3COLOR is NP-hard.

Now consider the related problem 12COLOR: Given a graph, can we color each vertex with one of twelve colors, so that every edge touches two different colors? **Prove** that 12COLOR is NP-hard.

**You may assume the following problems are NP-hard:**

**CIRCUITSAT:** Given a boolean circuit, are there any input values that make the circuit output True?

**PLANAR CIRCUITSAT:** Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output True?

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

**MAXINDEPENDENTSET:** Given an undirected graph  $G$ , what is the size of the largest subset of vertices in  $G$  that have no edges among them?

**MAXCLIQUE:** Given an undirected graph  $G$ , what is the size of the largest complete subgraph of  $G$ ?

**MINVERTEXCOVER:** Given an undirected graph  $G$ , what is the size of the smallest subset of vertices that touch every edge in  $G$ ?

**MINSETCOVER:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subcollection whose union is  $S$ ?

**MINHITTINGSET:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subset of  $S$  that intersects every subset  $S_i$ ?

**3COLOR:** Given an undirected graph  $G$ , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

**HAMILTONIANCYCLE:** Given a graph  $G$ , can is there a cycle in  $G$  that visits every vertex once?

**HAMILTONIANPATH:** Given a graph  $G$ , can is there a path in  $G$  that visits every vertex once?

**DOUBLEHAMILTONIANCYCLE:** Given a graph  $G$ , can is there a closed walk in  $G$  that visits every vertex twice?

**DOUBLEHAMILTONIANPATH:** Given a graph  $G$ , can is there an open walk in  $G$  that visits every vertex twice?

**MINDEGREESPANNINGTREE:** Given an undirected graph  $G$ , what is the minimum degree of any spanning tree of  $G$ ?

**MINLEAVESPANNINGTREE:** Given an undirected graph  $G$ , what is the minimum number of leaves in any spanning tree of  $G$ ?

**TRAVELINGSalesman:** Given a graph  $G$  with weighted edges, what is the minimum cost of any Hamiltonian path/cycle in  $G$ ?

**LONGESTPATH:** Given a graph  $G$  with weighted edges and two vertices  $s$  and  $t$ , what is the length of the longest *simple* path from  $s$  to  $t$  in  $G$ ?

**SUBSETSUM:** Given a set  $X$  of positive integers and an integer  $k$ , does  $X$  have a subset whose elements sum to  $k$ ?

**PARTITION:** Given a set  $X$  of positive integers, can  $X$  be partitioned into two subsets with the same sum?

**3PARTITION:** Given a set  $X$  of  $n$  positive integers, can  $X$  be partitioned into  $n/3$  three-element subsets, all with the same sum?

**MINESWEEPER:** Given a Minesweeper configuration and a particular square  $x$ , is it safe to click on  $x$ ?

**TETRIS:** Given a sequence of  $N$  Tetris pieces and a partially filled  $n \times k$  board, is it possible to play every piece in the sequence without overflowing the board?

**SUDOKU:** Given an  $n \times n$  Sudoku puzzle, does it have a solution?

**KENKEN:** Given an  $n \times n$  Ken-Ken puzzle, does it have a solution?

# CS 473: Undergraduate Algorithms, Spring 2010

## Homework 0

Due Tuesday, January 26, 2009 in class

---

- This homework tests your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, graphs, and most importantly, induction—to help you identify gaps in your background knowledge. **You are responsible for filling those gaps.** The early chapters of any algorithms textbook should be sufficient review, but you may also want consult your favorite discrete mathematics and data structures textbooks. If you need help, please ask in office hours and/or on the course newsgroup.
- Each student must submit individual solutions for these homework problems. For all future homeworks, groups of up to three students may submit (or present) a single group solution for each problem.
- Please carefully read the course policies linked from the course web site. If you have *any* questions, please ask during lecture or office hours, or post your question to the course newsgroup. In particular:
  - Submit five separately stapled solutions, one for each numbered problem, with your name and NetID clearly printed on each page. Please do *not* staple everything together.
  - You may use any source at your disposal—paper, electronic, or human—but you **must** write your solutions in your own words, and you **must** cite every source that you use.
  - Unless explicitly stated otherwise, **every** homework problem requires a proof.
  - Answering “I don’t know” to any homework or exam problem (except for extra credit problems) is worth 25% partial credit.
  - Algorithms or proofs containing phrases like “and so on” or “repeat this process for all  $n$ ”, instead of an explicit loop, recursion, or induction, will receive 0 points.

1. (a) ***Write the sentence “I understand the course policies.”***
- (b) [5 pts] Solve the following recurrences. State tight asymptotic bounds for each function in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ . Assume reasonable but nontrivial base cases if none are given. ***Do not submit proofs***—just a list of five functions—but you should do them anyway, just for practice.
- $A(n) = 3A(n - 1) + 1$
  - $B(n) = B(n - 5) + 2n - 3$
  - $C(n) = 4C(n/2) + \sqrt{n}$
  - $D(n) = 3D(n/3) + n^2$
  - $E(n) = E(n - 1)^2 - E(n - 2)^2$ , where  $E(0) = 0$  and  $E(1) = 1$     [Hint: This is easy!]
- (c) [5 pts] Sort the following functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. ***Do not submit proofs***—just a sorted list of 16 functions—but you should do them anyway, just for practice.

Write  $f(n) \ll g(n)$  to indicate that  $f(n) = o(g(n))$ , and write  $f(n) \equiv g(n)$  to mean  $f(n) = \Theta(g(n))$ . We use the notation  $\lg n = \log_2 n$ .

$n$	$\lg n$	$\sqrt{n}$	$5^n$
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$5^{\sqrt{n}}$	$\sqrt{5^n}$
$5^{\lg n}$	$\lg(5^n)$	$5^{\lg \sqrt{n}}$	$5^{\sqrt{\lg n}}$
$\sqrt{5^{\lg n}}$	$\lg(5^{\sqrt{n}})$	$\lg \sqrt{5^n}$	$\sqrt{\lg(5^n)}$

2. [CS 225 Spring 2009] Suppose we build up a binary search tree by inserting elements one at a time from the set  $\{1, 2, 3, \dots, n\}$ , starting with the empty tree. The structure of the resulting binary search tree depends on the order that these elements are inserted; every insertion order leads to a different  $n$ -node binary search tree.

Recall that the *depth* of a leaf  $\ell$  in a binary search tree is the number of *edges* between  $\ell$  and the root, and the depth of a binary tree is the maximum depth of its leaves.

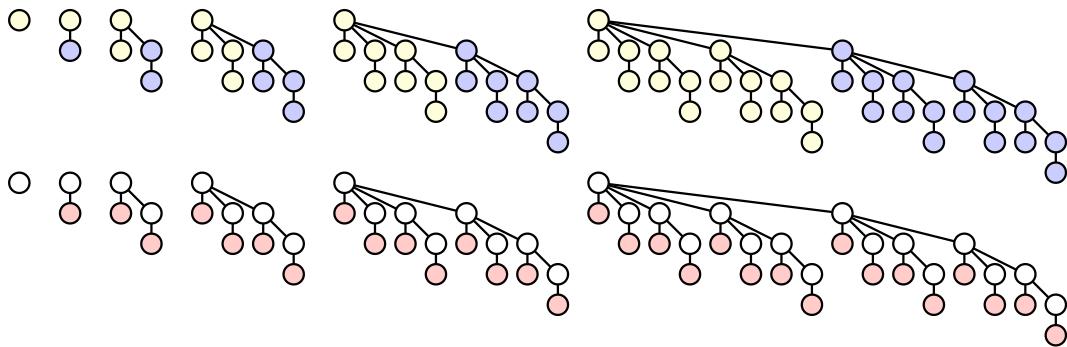
- (a) What is the maximum possible depth of an  $n$ -node binary search tree? Give an *exact* answer, and prove that it is correct.
- (b) *Exactly* how many different insertion orders result in an  $n$ -node binary search tree with maximum possible depth? Prove your answer is correct. [Hint: Set up and solve a recurrence. Don't forget to prove that recurrence counts what you want it to count.]

3. [CS 173 Spring 2009] A *binomial tree of order k* is defined recursively as follows:

- A binomial tree of order 0 is a single node.
- For all  $k > 0$ , a binomial tree of order  $k$  consists of two binomial trees of order  $k - 1$ , with the root of one tree connected as a new child of the root of the other. (See the figure below.)

Prove the following claims:

- For all non-negative integers  $k$ , a binomial tree of order  $k$  has exactly  $2^k$  nodes.
- For all positive integers  $k$ , attaching a leaf to every node in a binomial tree of order  $k - 1$  results in a binomial tree of order  $k$ .
- For all non-negative integers  $k$  and  $d$ , a binomial tree of order  $k$  has exactly  $\binom{k}{d}$  nodes with depth  $d$ .



Binomial trees of order 0 through 5.

Top row: the recursive definition. Bottom row: the property claimed in part (b).

4. [CS 373 Fall 2009] For any language  $L \in \Sigma^*$ , let

$$\text{Rotate}(L) := \left\{ w \in \Sigma^* \mid w = xy \text{ and } yx \in L \text{ for some strings } x, y \in \Sigma^* \right\}$$

For example,  $\text{Rotate}(\{\text{0OK!}, \text{0OKOK}\}) = \{\text{0OK!}, \text{OK!0}, \text{K!OO}, \text{!OOK}, \text{0OKOK}, \text{OKOKO}, \text{KOKOO}\}$ .

Prove that if  $L$  is a regular language, then  $\text{Rotate}(L)$  is also a regular language. [Hint: Remember the power of nondeterminism.]

5. Herr Professor Doktor Georg von den Dschungel has a 24-node binary tree, in which every node is labeled with a unique letter of the German alphabet, which is just like the English alphabet with four extra letters: Ä, Ö, Ü, and ß. (Don't confuse these with A, O, U, and B!) Preorder and postorder traversals of the tree visit the nodes in the following order:

- Preorder: B K Ü E H L Z I Ö R C ß T S O A Ä D F M N U G
- Postorder: H I Ö Z R L E C Ü S O T A ß K D M U G N F Ä B

- List the nodes in George's tree in the order visited by an inorder traversal.
- Draw George's tree.

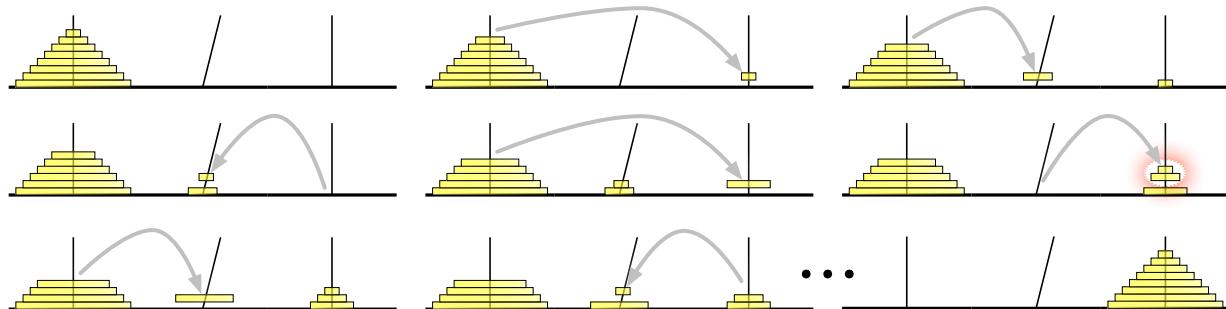
- \*6. [Extra credit] You may be familiar with the story behind the famous Tower of Hanoi puzzle, as related by Henri de Parville in 1884:

In the great temple at Benares beneath the dome which marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four discs of pure gold, the largest disc resting on the brass plate, and the others getting smaller and smaller up to the top one. This is the Tower of Bramah. Day and night unceasingly the priests transfer the discs from one diamond needle to another according to the fixed and immutable laws of Bramah, which require that the priest on duty must not move more than one disc at a time and that he must place this disc on a needle so that there is no smaller disc below it. When the sixty-four discs shall have been thus transferred from the needle on which at the creation God placed them to one of the other needles, tower, temple, and Brahmins alike will crumble into dust, and with a thunderclap the world will vanish.

A less familiar chapter in the temple's history is its brief relocation to Pisa in the early 13th century. The relocation was organized by the wealthy merchant-mathematician Leonardo Fibonacci, at the request of the Holy Roman Emperor Frederick II, who had heard reports of the temple from soldiers returning from the Crusades. The Towers of Pisa and their attendant monks became famous, helping to establish Pisa as a dominant trading center on the Italian peninsula.

Unfortunately, almost as soon as the temple was moved, one of the diamond needles began to lean to one side. To avoid the possibility of the leaning tower falling over from too much use, Fibonacci convinced the priests to adopt a more relaxed rule: ***Any number of disks on the leaning needle can be moved together to another needle in a single move.*** It was still forbidden to place a larger disk on top of a smaller disk, and disks had to be moved one at a time *onto* the leaning needle or between the two vertical needles.

Thanks to Fibonacci's new rule, the priests could bring about the end of the universe somewhat faster from Pisa than they could from Benares. Fortunately, the temple was moved from Pisa back to Benares after the newly crowned Pope Gregory IX excommunicated Frederick II, making the local priests less sympathetic to hosting foreign heretics with strange mathematical habits. Soon afterward, a bell tower was erected on the spot where the temple once stood; it too began to lean almost immediately.

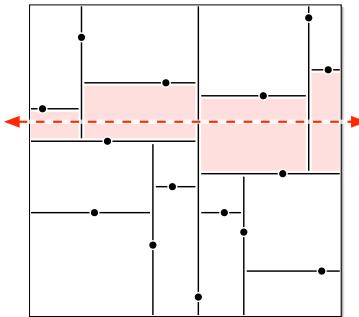


The Towers of Pisa. In the fifth move, two disks are taken off the leaning needle.

Describe an algorithm to transfer a stack of  $n$  disks from one *vertical* needle to the other *vertical* needle, using the smallest possible number of moves. *Exactly* how many moves does your algorithm perform?

- For this and all future homeworks, groups of up to three students can submit (or present) a single common solution. Please remember to write the names of all group members on every page.
- Please fill out the online input survey linked from the course web page no later than Thursday, January 28.** Among other things, this survey asks you to identify the other members of your HW1 group, so that we can partition the class into presentation clusters without breaking up your group. We will announce the presentation clusters on Friday, January 29.
- Students in **Cluster 1** will present their solutions to Jeff or one of the TAs, on Tuesday or Wednesday of the due date (February 2 or February 3), instead of submitting written solutions. **Each homework group in Cluster 1 must sign up for a 30-minute time slot no later than Monday, February 1.** Signup sheets will be posted at 3303 Siebel Center ('The Theory Lab') later this week. Please see the course web page for more details.

- Suppose we have  $n$  points scattered inside a two-dimensional box. A *kd-tree* recursively subdivides the points as follows. First we split the box into two smaller boxes with a *vertical* line, then we split each of those boxes with *horizontal* lines, and so on, always alternating between horizontal and vertical splits. Each time we split a box, the splitting line partitions the rest of the interior points *as evenly as possible* by passing through a median point in the interior of the box (*not* on its boundary). If a box doesn't contain any points, we don't split it any more; these final empty boxes are called *cells*.

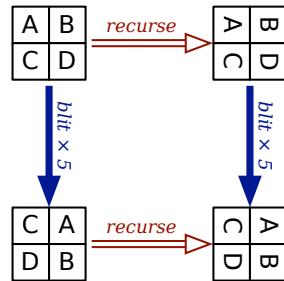


A kd-tree for 15 points. The dashed line crosses the four shaded cells.

- How many cells are there, as a function of  $n$ ? Prove your answer is correct.
- In the worst case, *exactly* how many cells can a horizontal line cross, as a function of  $n$ ? Prove your answer is correct. Assume that  $n = 2^k - 1$  for some integer  $k$ .
- Suppose we have  $n$  points stored in a kd-tree. Describe and analyze an algorithm that counts the number of points above a horizontal line (such as the dashed line in the figure) as quickly as possible. [Hint: Use part (b).]
- Describe an analyze an efficient algorithm that counts, given a kd-tree storing  $n$  points, the number of points that lie inside a rectangle  $R$  with horizontal and vertical sides. [Hint: Use part (c).]

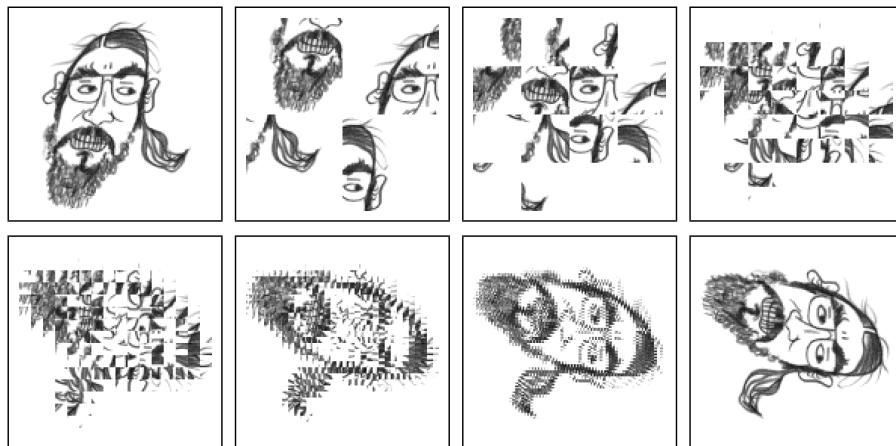
2. Most graphics hardware includes support for a low-level operation called *blit*, or **block transfer**, which quickly copies a rectangular chunk of a pixel map (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

Suppose we want to rotate an  $n \times n$  pixel map  $90^\circ$  clockwise. One way to do this, at least when  $n$  is a power of two, is to split the pixel map into four  $n/2 \times n/2$  blocks, move each block to its proper position using a sequence of five blits, and then recursively rotate each block. Alternately, we could *first* recursively rotate the blocks and *then* blit them into place.



Two algorithms for rotating a pixel map.

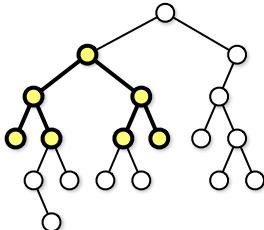
Solid arrows indicate blitting the blocks into place; hollow arrows indicate recursively rotating the blocks.



The first rotation algorithm (blit then recurse) in action.

- Prove that both versions of the algorithm are correct when  $n$  is a power of two.
- Exactly* how many blits does the algorithm perform when  $n$  is a power of two?
- Describe how to modify the algorithm so that it works for arbitrary  $n$ , not just powers of two. How many blits does your modified algorithm perform?
- What is your algorithm's running time if a  $k \times k$  blit takes  $O(k^2)$  time?
- What if a  $k \times k$  blit takes only  $O(k)$  time?

3. For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth. Describe and analyze a recursive algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return the root and the depth of this subtree.



The largest complete subtree of this binary tree has depth 2.

# CS 473: Undergraduate Algorithms, Spring 2010

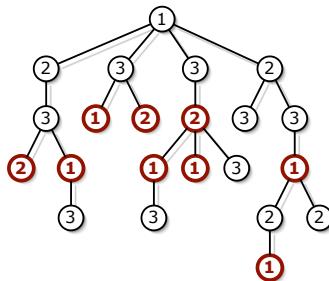
## Homework 2

Written solutions due Tuesday, February 9, 2010 at noon

- Roughly 1/3 of the students will give oral presentations of their solutions to the TAs. **You should have received an email telling you whether you are expected to present this homework.** Please see the course web page for further details.
- Groups of up to three students may submit a common solution. Please clearly write every group member's name and NetID on every page of your submission. Please start your solution to each numbered problem on a new sheet of paper. Please **don't** staple solutions for different problems together.

1. A **palindrome** is a string that reads the same forwards and backwards, like x, pop, noon, redivider, or "sator arepo tenet opera rotas", Describe and analyze an algorithm to find the length of the longest subsequence of a given string that is also a palindrome. For example, the longest palindrome subsequence of **MAHDYNAMICPROGRAMZLETMESHOWYOUTHEM** is **MHYMRORMYHM**, so given that string as input, your algorithm should output the number 11.
2. Oh, no! You have been appointed as the gift czar for Giggle, Inc.'s annual mandatory holiday party! The president of the company, who is certifiably insane, has declared that every Giggle employee must receive one of three gifts: (1) an all-expenses-paid six-week vacation anywhere in the world, (2) an all-the-pancakes-you-can-eat breakfast for two at Jumping Jack Flash's Flapjack Stack Shack, or (3) a burning paper bag full of dog poop. Corporate regulations prohibit any employee from receiving the same gift as his/her direct supervisor. Any employee who receives a better gift than his/her direct supervisor will almost certainly be fired in a fit of jealousy. How do you decide what gifts everyone gets if you want to minimize the number of people that get fired?

More formally, suppose you are given a rooted tree  $T$ , representing the company hierarchy. You want to label each node in  $T$  with an integer 1, 2, or 3, such that every node has a different label from its parent.. The **cost** of an labeling is the number of nodes that have smaller labels than their parents. Describe and analyze an algorithm to compute the minimum cost of any labeling of the given tree  $T$ . (Your algorithm does *not* have to compute the actual best labeling—just its cost.)



A tree labeling with cost 9. Bold nodes have smaller labels than their parents.

This is **not** the optimal labeling for this tree.

3. After graduating from UIUC, you have decided to join the Wall Street Bank ***Boole Long Live***. The managing director of the bank, Eloob Egroeg, is a genius mathematician who worships George Boole<sup>1</sup> every morning before leaving for the office. The first day of every hired employee is a 'solve-or-die' day where s/he has to solve one of the problems posed by Eloob within 24 hours. Those who fail to solve the problem are fired immediately!

Entering into the bank for the first time, you notice that the offices of the employees are organized in a straight row, with a large "T" or "F" written on the door of each office. Furthermore, between each adjacent pair of offices, there is a board marked by one of the symbols  $\wedge$ ,  $\vee$ , or  $\oplus$ . When you ask about these arcane symbols, Eloob confirms that  $T$  and  $F$  represent the boolean values 'true' and 'false', and the symbols on the boards represent the standard boolean operators AND, OR, and XOR. He also explains that these letters and symbols describe whether certain combinations of employees can work together successfully. At the start of any new project, Eloob hierarchically clusters his employees by adding parentheses to the sequence of symbols, to obtain an unambiguous boolean expression. The project is successful if this parenthesized boolean expression evaluates to  $T$ .

For example, if the bank has three employees, and the sequence of symbols on and between their doors is  $T \wedge F \oplus T$ , there is exactly one successful parenthesization scheme:  $(T \wedge (F \oplus T))$ . However, if the list of door symbols is  $F \wedge T \oplus F$ , there is no way to add parentheses to make the project successful.

Eloob finally poses your solve-or-die question: Describe and algorithm to decide whether a given sequence of symbols can be parenthesized so that the resulting boolean expression evaluates to  $T$ . The input to your algorithm is an array  $S[0..2n]$ , where  $S[i] \in \{T, F\}$  when  $i$  is even, and  $S[i] \in \{\vee, \wedge, \oplus\}$  when  $i$  is odd.

---

<sup>1</sup>1815-1864, The inventor of Boolean Logic

- For this and all future homeworks, groups of up to three students can submit (or present) a single common solution. Please remember to write the names of all group members on every page.
  - Students in **Cluster 3** will present their solutions to Jeff or one of the TAs, on Tuesday or Wednesday of the due date (February 16 or February 17), instead of submitting written solutions. **Each homework group in Cluster 3 must sign up for a 30-minute time slot no later than Monday, February 15.** Signup sheets will be posted at 3304 Siebel Center ('The Theory Lab') later this week. Please see the course web page for more details.
- 

1. You saw in class a correct greedy algorithm for finding the maximum number of non-conflicting courses from a given set of possible courses. This algorithm repeatedly selects the class with the earliest completion time that does not conflict with any previously selected class.

Below are four alternative greedy algorithms. For each algorithm, either prove that the algorithm constructs an optimal schedule, or give a concrete counterexample showing that the algorithm is suboptimal.

- (a) Choose the course that *ends latest*, discard all conflicting classes, and recurse.
- (b) Choose the course that *starts first*, discard all conflicting classes, and recurse.
- (c) Choose the course with *shortest duration*, discard all conflicting classes, and recurse.
- (d) Choose a course that *conflicts with the fewest other courses* (breaking ties arbitrarily), discard all conflicting classes, and recurse.

2. You have been given the task of designing an algorithm for vending machines that computes the smallest number of coins worth any given amount of money. Your supervisors at The Area 51 Soda Company are anticipating a hostile takeover of earth by an advanced alien race that uses an unknown system of currency. So your algorithm must be as general as possible so that it will work with the alien system, whatever it turns out to be.

Given a quantity of money  $x$ , and a set of coin denominations  $b_1, \dots, b_k$ , your algorithm should compute how to make change for  $x$  with the fewest number of coins. For example, if you use the US coin denominations (1¢, 5¢, 10¢, 25¢, 50¢, and 100¢), the optimal way to make 17¢ in change uses 4 coins: one dime (10¢), one nickel (5¢), and two pennies (1¢).

- (a) Show that the following greedy algorithm does *not* work for all currency systems: If  $x = 0$ , do nothing. Otherwise, find the largest denomination  $c \leq x$ , issue one  $c$ -cent coin, and recursively give  $x - c$  cents in change.
- (b) Now suppose that the system of currency you are concerned with only has coins in powers of some base  $b$ . That is, the coin denominations are  $b^0, b^1, b^2, \dots, b^k$ . Show that the greedy algorithm described in part (a) does make optimal change in this currency system.
- (c) Describe and analyze an algorithm that computes optimal change for *any* set of coin denominations. (You may assume the aliens' currency system includes a 1-cent coin, so that making change is always possible.)

3. Suppose you have just purchased a new type of hybrid car that uses fuel extremely efficiently, but can only travel 100 miles on a single battery. The car's fuel is stored in a single-use battery, which must be replaced after at most 100 miles. The actual fuel is virtually free, but the batteries are expensive and can only be installed by licensed battery-replacement technicians. Thus, even if you decide to replace your battery early, you must still pay full price for the new battery to be installed. Moreover, because these batteries are in high demand, no one can afford to own more than one battery at a time.

Suppose you are trying to get from San Francisco to New York City on the new Inter-Continental Super-Highway, which runs in a direct line between these two cities. There are several fueling stations along the way; each station charges a different price for installing a new battery. Before you start your trip, you carefully print the Wikipedia page listing the locations and prices of every fueling station on the ICSH. Given this information, how do you decide the best places to stop for fuel?

More formally, suppose you are given two arrays  $D[1..n]$  and  $C[1..n]$ , where  $D[i]$  is the distance from the start of the highway to the  $i$ th station, and  $C[i]$  is the cost to replace your battery at the  $i$ th station. Assume that your trip starts and ends at fueling stations (so  $D[1] = 0$  and  $D[n]$  is the total length of your trip), and that your car starts with an empty battery (so you must install a new battery at station 1).

- (a) Describe and analyze a greedy algorithm to find the minimum number of refueling stops needed to complete your trip. Don't forget to prove that your algorithm is correct.
- (b) But what you really want to minimize is the total *cost* of travel. Show that your greedy algorithm in part (a) does *not* produce an optimal solution when extended to this setting.
- (c) Describe a dynamic programming algorithm to compute the locations of the fuel stations you should stop at to minimize the cost of travel.

1. Suppose we want to write an efficient function  $\text{SHUFFLE}(n)$  that returns a permutation of the set  $\{1, 2, \dots, n\}$  chosen uniformly at random.

- (a) Prove that the following algorithm is *not* correct. [Hint: Consider the case  $n = 3$ .]

```
SHUFFLE( $n$ ):
  for  $i \leftarrow 1$  to  $n$ 
     $\pi[i] \leftarrow i$ 
  for  $i \leftarrow 1$  to  $n$ 
    swap  $\pi[i] \leftrightarrow \pi[\text{RANDOM}(n)]$ 
  return  $\pi[1..n]$ 
```

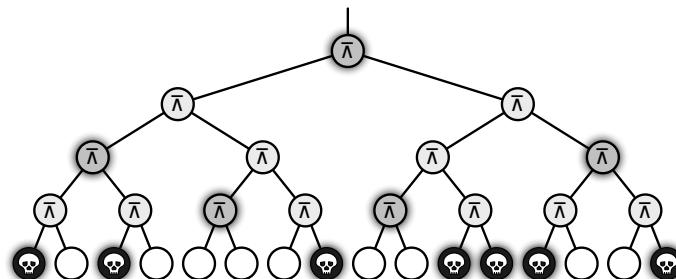
- (b) Consider the following implementation of  $\text{SHUFFLE}$ .

```
SHUFFLE( $n$ ):
  for  $i \leftarrow 1$  to  $n$ 
     $\pi[i] \leftarrow \text{NULL}$ 
  for  $i \leftarrow 1$  to  $n$ 
     $j \leftarrow \text{RANDOM}(n)$ 
    while ( $\pi[j] \neq \text{NULL}$ )
       $j \leftarrow \text{RANDOM}(n)$ 
     $\pi[j] \leftarrow i$ 
  return  $\pi[1..n]$ 
```

Prove that this algorithm is correct. What is its expected running time?

- (c) Describe and analyze an implementation of  $\text{SHUFFLE}$  that runs in  $O(n)$  time. (An algorithm that runs in  $O(n)$  *expected* time is fine, but  $O(n)$  *worst-case* time is possible.)

2. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with  $4^n$  leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after  $2n$  moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.



You can decide whether it's worth playing or not as follows. Imagine that the tree is a Boolean circuit whose inputs are specified at the leaves: white and black represent TRUE and FALSE inputs, respectively. Each internal node in the tree is a NAND gate that gets its input from its children and passes its output to its parent. (Recall that a NAND gate outputs FALSE if and only if both its inputs are TRUE.) If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead. Or maybe Battleship.

- (a) Describe and analyze a deterministic algorithm to determine whether or not you can win. [Hint: This is easy!]
- (b) Unfortunately, Death won't give you enough time to look at every node in the tree. Describe a randomized algorithm that determines whether you can win in  $O(3^n)$  expected time. [Hint: Consider the case  $n = 1$ .]
- \*(c) [Extra credit] Describe and analyze a randomized algorithm that determines whether you can win in  $O(c^n)$  expected time, for some constant  $c < 3$ . [Hint: You may not need to change your algorithm from part (b) at all!]
3. A *meldable priority queue* stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:
- MAKEQUEUE: Return a new priority queue containing the empty set.
  - FINDMIN( $Q$ ): Return the smallest element of  $Q$  (if any).
  - DELETEMIN( $Q$ ): Remove the smallest element in  $Q$  (if any).
  - INSERT( $Q, x$ ): Insert element  $x$  into  $Q$ , if it is not already there.
  - DECREASEKEY( $Q, x, y$ ): Replace an element  $x \in Q$  with a smaller key  $y$ . (If  $y > x$ , the operation fails.) The input is a pointer directly to the node in  $Q$  containing  $x$ .
  - DELETE( $Q, x$ ): Delete the element  $x \in Q$ . The input is a pointer directly to the node in  $Q$  containing  $x$ .
  - MELD( $Q_1, Q_2$ ): Return a new priority queue containing all the elements of  $Q_1$  and  $Q_2$ ; this operation destroys  $Q_1$  and  $Q_2$ .

A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm:

```

MELD( $Q_1, Q_2$ ):
  if  $Q_1$  is empty return  $Q_2$ 
  if  $Q_2$  is empty return  $Q_1$ 
  if  $key(Q_1) > key(Q_2)$ 
    swap  $Q_1 \leftrightarrow Q_2$ 
  with probability 1/2
     $left(Q_1) \leftarrow MELD(left(Q_1), Q_2)$ 
  else
     $right(Q_1) \leftarrow MELD(right(Q_1), Q_2)$ 
  return  $Q_1$ 
```

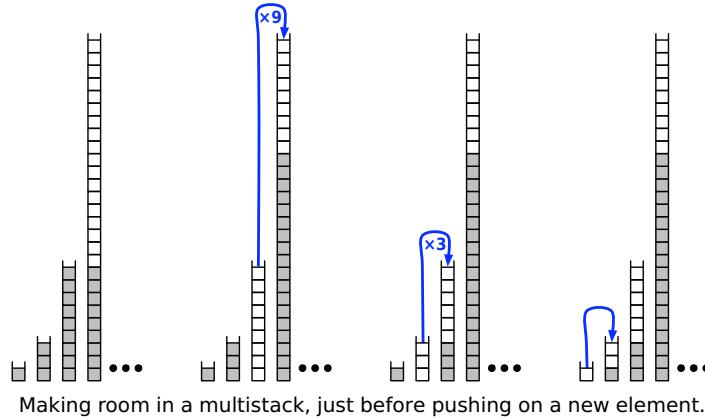
- (a) Prove that for *any* heap-ordered binary trees  $Q_1$  and  $Q_2$  (not just those constructed by the operations listed above), the expected running time of  $MELD(Q_1, Q_2)$  is  $O(\log n)$ , where  $n = |Q_1| + |Q_2|$ . [Hint: How long is a random root-to-leaf path in an  $n$ -node binary tree if each left/right choice is made with equal probability?]
- (b) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and  $O(1)$  additional time. (This implies that every operation takes  $O(\log n)$  expected time.)

1. A *multistack* consists of an infinite series of stacks  $S_0, S_1, S_2, \dots$ , where the  $i$ th stack  $S_i$  can hold up to  $3^i$  elements. The user always pushes and pops elements from the smallest stack  $S_0$ . However, before any element can be pushed onto any full stack  $S_i$ , we first pop all the elements off  $S_i$  and push them onto stack  $S_{i+1}$  to make room. (Thus, if  $S_{i+1}$  is already full, we first recursively move all its members to  $S_{i+2}$ .) Similarly, before any element can be popped from any empty stack  $S_i$ , we first pop  $3^i$  elements from  $S_{i+1}$  and push them onto  $S_i$  to make room. (Thus, if  $S_{i+1}$  is already empty, we first recursively fill it by popping elements from  $S_{i+2}$ .) Moving a single element from one stack to another takes  $O(1)$  time.

Here is pseudocode for the multistack operations `MSPUSH` and `MSPOP`. The internal stacks are managed with the subroutines `PUSH` and `POP`.

```
MPUSH( $x$ ):
   $i \leftarrow 0$ 
  while  $S_i$  is full
     $i \leftarrow i + 1$ 
  while  $i > 0$ 
     $i \leftarrow i - 1$ 
    for  $j \leftarrow 1$  to  $3^i$ 
      PUSH( $S_{i+1}$ , POP( $S_i$ )))
  PUSH( $S_0, x$ )
```

```
MPOP( $x$ ):
   $i \leftarrow 0$ 
  while  $S_i$  is empty
     $i \leftarrow i + 1$ 
  while  $i > 0$ 
     $i \leftarrow i - 1$ 
    for  $j \leftarrow 1$  to  $3^i$ 
      PUSH( $S_i$ , POP( $S_{i+1}$ )))
  return POP( $S_0$ )
```



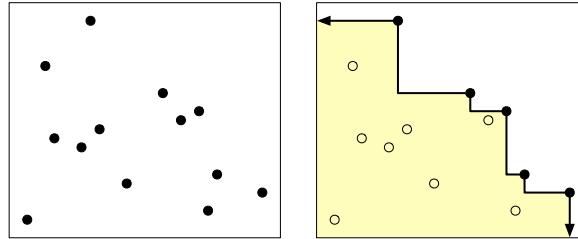
- In the worst case, how long does it take to push one more element onto a multistack containing  $n$  elements?
- Prove that if the user never pops anything from the multistack, the amortized cost of a push operation is  $O(\log n)$ , where  $n$  is the maximum number of elements in the multistack during its lifetime.
- Prove that in any intermixed sequence of pushes and pops, each push or pop operation takes  $O(\log n)$  amortized time, where  $n$  is the maximum number of elements in the multistack during its lifetime.

2. Design and analyze a simple data structure that maintains a list of integers and supports the following operations.

- $\text{CREATE}()$  creates and returns a new list
- $\text{PUSH}(L, x)$  appends  $x$  to the end of  $L$
- $\text{POP}(L)$  deletes the last entry of  $L$  and returns it
- $\text{LOOKUP}(L, k)$  returns the  $k$ th entry of  $L$

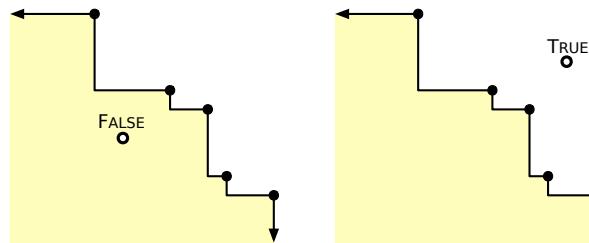
Your solution may use these primitive data structures: arrays, balanced binary search trees, heaps, queues, single or doubly linked lists, and stacks. If your algorithm uses *anything* fancier, you must give an explicit implementation. Your data structure must support all operations in amortized constant time. In addition, your data structure must support each  $\text{LOOKUP}$  in *worst-case*  $O(1)$  time. At all times, the size of your data structure must be linear in the number of objects it stores.

3. Let  $P$  be a set of  $n$  points in the plane. The *staircase* of  $P$  is the set of all points in the plane that have at least one point in  $P$  both above and to the right.



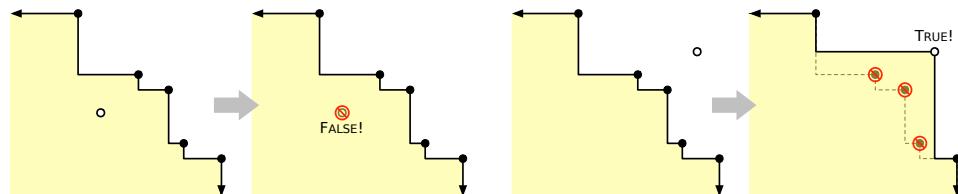
A set of points in the plane and its staircase (shaded).

- (a) Describe an algorithm to compute the staircase of a set of  $n$  points in  $O(n \log n)$  time.
- (b) Describe and analyze a data structure that stores the staircase of a set of points, and an algorithm ABOVE?( $x, y$ ) that returns TRUE if the point  $(x, y)$  is above the staircase, or FALSE otherwise. Your data structure should use  $O(n)$  space, and your ABOVE? algorithm should run in  $O(\log n)$  time.



Two staircase queries.

- (c) Describe and analyze a data structure that maintains a staircase as new points are inserted. Specifically, your data structure should support a function INSERT( $x, y$ ) that adds the point  $(x, y)$  to the underlying point set and returns TRUE or FALSE to indicate whether the staircase of the set has changed. Your data structure should use  $O(n)$  space, and your INSERT algorithm should run in  $O(\log n)$  amortized time.



Two staircase insertions.

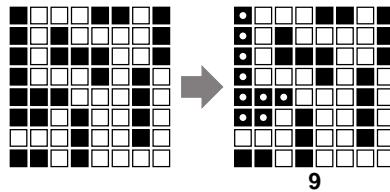
# CS 473: Undergraduate Algorithms, Spring 2010

## Homework 6

Written solutions due Tuesday, March 16, 2010 at noon

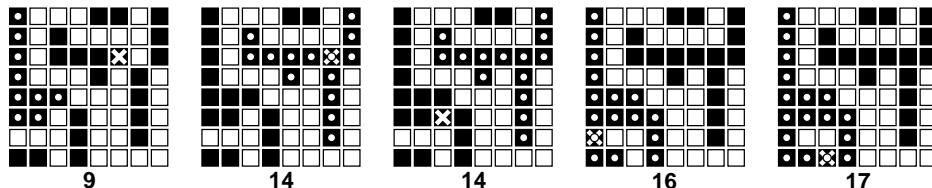
---

1. (a) Describe and analyze an algorithm to compute the size of the largest connected component of black pixels in an  $n \times n$  bitmap  $B[1..n, 1..n]$ . For example, given the bitmap below as input, your algorithm should return the number 9, because the largest connected black component (marked with white dots on the right) contains nine pixels.



- (b) Design and analyze an algorithm  $\text{BLACKEN}(i, j)$  that colors the pixel  $B[i, j]$  black and returns the size of the largest black component in the bitmap. For full credit, the *amortized* running time of your algorithm (starting with an all-white bitmap) must be as small as possible.

For example, at each step in the sequence below, we blacken the pixel marked with an X. The largest black component is marked with white dots; the number underneath shows the correct output of the  $\text{BLACKEN}$  algorithm.



- (c) What is the *worst-case* running time of your  $\text{BLACKEN}$  algorithm?
2. Suppose you are given a graph  $G$  with weighted edges and a minimum spanning tree  $T$  of  $G$ .
  - (a) Describe an algorithm to update the minimum spanning tree when the weight of a single edge  $e$  is *increased*.
  - (b) Describe an algorithm to update the minimum spanning tree when the weight of a single edge  $e$  is *decreased*.

In both cases, the input to your algorithm is the edge  $e$  and its new weight; your algorithms should modify  $T$  so that it is still a minimum spanning tree. [Hint: Consider the cases  $e \in T$  and  $e \notin T$  separately.]

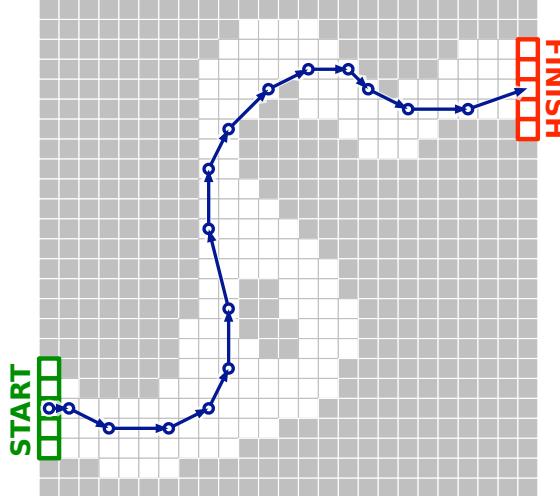
3. **Racetrack** (also known as *Graph Racers* and *Vector Rally*) is a two-player paper-and-pencil racing game of uncertain origin that Jeff played on the bus in 5th grade.<sup>1</sup> The game is played using a racetrack drawn on a sheet of graph paper. The players alternately choose a sequence of grid points that represent the motion of a car around the track, subject to certain constraints explained below.

Each car has a *position* and a *velocity*, both with integer *x*- and *y*-coordinates. The initial position is an arbitrary point on the starting line, chosen by the player; the initial velocity is always  $(0, 0)$ . At each step, the player optionally increments or decrements either or both coordinates of the car's velocity; in other words, each component of the velocity can change by at most 1 in a single step. The car's new position is then determined by adding the new velocity to the car's previous position. The new position must lie inside the track; otherwise, the car crashes and that player immediately loses the race. The first car that reaches a position on the finish line is the winner.

Suppose the racetrack is represented by an  $n \times n$  array of bits, where each 0 bit represents a grid point inside the track, each 1 bit represents a grid point outside the track, the 'starting line' is the first column, and the 'finish line' is the last column.

Describe and analyze an algorithm to find the *minimum number of steps* required to move a car from the starting line to the finish line according to these rules, given a racetrack bitmap as input. [Hint: Build a graph. What are the vertices? What are the edges? What problem is this?]

velocity	position
$(0, 0)$	$(1, 5)$
$(1, 0)$	$(2, 5)$
$(2, -1)$	$(4, 4)$
$(3, 0)$	$(7, 4)$
$(2, 1)$	$(9, 5)$
$(1, 2)$	$(10, 7)$
$(0, 3)$	$(10, 10)$
$(-1, 4)$	$(9, 14)$
$(0, 3)$	$(9, 17)$
$(1, 2)$	$(10, 19)$
$(2, 2)$	$(12, 21)$
$(2, 1)$	$(14, 22)$
$(2, 0)$	$(16, 22)$
$(1, -1)$	$(17, 21)$
$(2, -1)$	$(19, 20)$
$(3, 0)$	$(22, 20)$
$(3, 1)$	$(25, 21)$



A 16-step Racetrack run, on a  $25 \times 25$  track. This is *not* the shortest run on this track.

<sup>1</sup>The actual game Jeff played was a bit more complicated than the version described in this problem. In particular, the track was a freeform curve, and by default, the entire line segment traversed by a car in a single step had to lie entirely inside the track. If a car did run off the track, it started its next turn with velocity zero, at the legal grid point closest to where it first crossed the track boundary.

1. On an overnight camping trip in Sunnydale National Park, you are woken from a restless sleep by a scream. As you crawl out of your tent to investigate, a terrified park ranger runs out of the woods, covered in blood and clutching a crumpled piece of paper to his chest. As he reaches your tent, he gasps, “Get out...while...you...”, thrusts the paper into your hands, and falls to the ground. Checking his pulse, you discover that the ranger is stone dead.

You look down at the paper and recognize a map of the park, drawn as an undirected graph, where vertices represent landmarks in the park, and edges represent trails between those landmarks. (Trails start and end at landmarks and do not cross.) You recognize one of the vertices as your current location; several vertices on the boundary of the map are labeled EXIT.

On closer examination, you notice that someone (perhaps the poor dead park ranger) has written a real number between 0 and 1 next to each vertex and each edge. A scrawled note on the back of the map indicates that a number next to an edge is the probability of encountering a vampire along the corresponding trail, and a number next to a vertex is the probability of encountering a vampire at the corresponding landmark. (Vampires can’t stand each other’s company, so you’ll never see more than one vampire on the same trail or at the same landmark.) The note warns you that stepping off the marked trails will result in a slow and painful death.

You glance down at the corpse at your feet. Yes, his death certainly looked painful. Wait, was that a twitch? Are his teeth getting longer? After driving a tent stake through the undead ranger’s heart, you wisely decide to leave the park immediately.

Describe and analyze an efficient algorithm to find a path from your current location to an arbitrary EXIT node, such that the total *expected number* of vampires encountered along the path is as small as possible. *Be sure to account for both the vertex probabilities and the edge probabilities!*

2. In this problem we will discover how you, too, can be employed by Wall Street and cause a major economic collapse! The *arbitrage* business is a money-making scheme that takes advantage of differences in currency exchange. In particular, suppose that 1 US dollar buys 120 Japanese yen; 1 yen buys 0.01 euros; and 1 euro buys 1.2 US dollars. Then, a trader starting with \$1 can convert his money from dollars to yen, then from yen to euros, and finally from euros back to dollars, ending with \$1.44! The cycle of currencies  $\$ \rightarrow ¥ \rightarrow € \rightarrow \$$  is called an *arbitrage cycle*. Of course, finding and exploiting arbitrage cycles before the prices are corrected requires extremely fast algorithms.

Suppose  $n$  different currencies are traded in your currency market. You are given the matrix  $R[1..n, 1..n]$  of exchange rates between every pair of currencies; for each  $i$  and  $j$ , one unit of currency  $i$  can be traded for  $R[i, j]$  units of currency  $j$ . (Do *not* assume that  $R[i, j] \cdot R[j, i] = 1$ .)

- (a) Describe an algorithm that returns an array  $V[1..n]$ , where  $V[i]$  is the maximum amount of currency  $i$  that you can obtain by trading, starting with one unit of currency 1, assuming there are no arbitrage cycles.
- (b) Describe an algorithm to determine whether the given matrix of currency exchange rates creates an arbitrage cycle.
- (c) Modify your algorithm from part (b) to actually return an arbitrage cycle, if it exists.

3. Let  $G = (V, E)$  be a directed graph with weighted edges; edge weights could be positive, negative, or zero. In this problem, you will develop an algorithm to compute shortest paths between *every* pair of vertices. The output from this algorithm is a two-dimensional array  $dist[1..V, 1..V]$ , where  $dist[i, j]$  is the length of the shortest path from vertex  $i$  to vertex  $j$ .

- (a) How could we delete some node  $v$  from this graph, without changing the shortest-path distance between any other pair of nodes? Describe an algorithm that constructs a directed graph  $G' = (V', E')$  with weighted edges, where  $V' = V \setminus \{v\}$ , and the shortest-path distance between any two nodes in  $G'$  is equal to the shortest-path distance between the same two nodes in  $G$ . For full credit, your algorithm should run in  $O(V^2)$  time.
- (b) Now suppose we have already computed all shortest-path distances in  $G'$ . Describe an algorithm to compute the shortest-path distances from  $v$  to every other node, and from every other node to  $v$ , in the original graph  $G$ . For full credit, your algorithm should run in  $O(V^2)$  time.
- (c) Combine parts (a) and (b) into an algorithm that finds the shortest paths between *every* pair of vertices in the graph. For full credit, your algorithm should run in  $O(V^3)$  time.

The lecture notes (along with most algorithms textbooks and Wikipedia) describe a dynamic programming algorithm due to Floyd and Warshall that computes all shortest paths in  $O(V^3)$  time. This is *not* that algorithm.

# CS 473: Undergraduate Algorithms, Spring 2010

## Homework 8

Written solutions due Tuesday, April 20, 2010 in class.

---

1. Suppose you have already computed a maximum  $(s, t)$ -flow  $f$  in a flow network  $G$  with integer capacities. Let  $k$  be an arbitrary positive integer, and let  $e$  be an arbitrary edge in  $G$  whose capacity is at least  $k$ .
  - (a) Suppose we *increase* the capacity of  $e$  by  $k$  units. Describe and analyze an algorithm to update the maximum flow.
  - (b) Now suppose we *decrease* the capacity of  $e$  by  $k$  units. Describe and analyze an algorithm to update the maximum flow.

For full credit, both algorithms should run in  $O(Ek)$  time. [Hint: First consider the case  $k = 1$ .]

2. Suppose we are given an array  $A[1..m][1..n]$  of non-negative real numbers. We want to *round*  $A$  to an integer matrix, by replacing each entry  $x$  in  $A$  with either  $\lfloor x \rfloor$  or  $\lceil x \rceil$ , without changing the sum of entries in any row or column of  $A$ . For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

Describe an efficient algorithm that either rounds  $A$  in this fashion, or reports correctly that no such rounding is possible.

3. A *cycle cover* of a given directed graph  $G = (V, E)$  is a set of vertex-disjoint cycles that cover all the vertices. Describe and analyze an efficient algorithm to find a cycle cover for a given graph, or correctly report that none exists. [Hint: Use bipartite matching!]

1. We say that an array  $A[1..n]$  is *k-sorted* if it can be divided into  $k$  blocks, each of size  $n/k$ , such that the elements in each block are larger than the elements in earlier blocks, and smaller than elements in later blocks. The elements within each block need not be sorted.

For example, the following array is 4-sorted:

1	2	4	3	7	6	8	5	10	11	9	12	15	13	16	14
---	---	---	---	---	---	---	---	----	----	---	----	----	----	----	----

- (a) Describe an algorithm that  $k$ -sorts an arbitrary array in time  $O(n \log k)$ .
- (b) Prove that any comparison-based  $k$ -sorting algorithm requires  $\Omega(n \log k)$  comparisons in the worst case.
- (c) Describe an algorithm that completely sorts an already  $k$ -sorted array in time  $O(n \log(n/k))$ .
- (d) Prove that any comparison-based algorithm to completely sort a  $k$ -sorted array requires  $\Omega(n \log(n/k))$  comparisons in the worst case.

In all cases, you can assume that  $n/k$  is an integer and that  $n! \approx \left(\frac{n}{e}\right)^n$ .

2. Recall the nuts and bolts problem from the first randomized algorithms lecture. You are given  $n$  nuts and  $n$  bolts of different sizes. Each nut matches exactly one bolt and vice versa. The nuts and bolts are all almost exactly the same size, so we can't tell if one bolt is bigger than the other, or if one nut is bigger than the other. If we try to match a nut with a bolt, however, we will discover either that the nut is too big, the nut is too small, or the nut is just right for the bolt. The goal was to find the matching nut for every bolt.

Now consider a relaxed version of the problem where the goal is to find the matching nuts for *half* of the bolts, or equivalently, to find  $n/2$  matched nut-bolt pairs. (It doesn't matter *which*  $n/2$  nuts and bolts are matched.) Prove that any deterministic algorithm to solve this problem must perform  $\Omega(n \log n)$  nut-bolt tests in the worst case.

3. UIUC has just finished constructing the new Reingold Building, the tallest dormitory on campus. In order to determine how much insurance to buy, the university administration needs to determine the highest safe floor in the building. A floor is considered *safe* if a drunk student *an egg* can fall from a window on that floor and land without breaking; if the egg breaks, the floor is considered *unsafe*. Any floor that is higher than an unsafe floor is also considered unsafe. The only way to determine whether a floor is safe is to drop an egg from a window on that floor.

You would like to find the lowest unsafe floor  $L$  by performing as few tests as possible; unfortunately, you have only a very limited supply of eggs.

- (a) Prove that if you have only one egg, you can find the lowest unsafe floor with  $L$  tests. [Hint: Yes, this is trivial.]
- (b) Prove that if you have only one egg, you must perform at least  $L$  tests in the worst case. In other words, prove that your algorithm from part (a) is optimal. [Hint: Use an adversary argument.]
- (c) Describe an algorithm to find the lowest unsafe floor using *two* eggs and only  $O(\sqrt{L})$  tests. [Hint: Ideally, each egg should be dropped the same number of times. How many floors can you test with  $n$  drops?]
- (d) Prove that if you start with two eggs, you must perform at least  $\Omega(\sqrt{L})$  tests in the worst case. In other words, prove that your algorithm from part (c) is optimal.

This homework is practice only. However, there will be at least one NP-hardness problem on the final exam, so working through this homework is *strongly* recommended. Students/groups are welcome to submit solutions for feedback (but not credit) in class on May 4, after which we will publish official solutions.

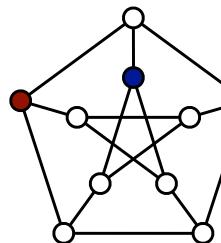
- Recall that 3SAT asks whether a given boolean formula in conjunctive normal form, with exactly three literals in each clause, is satisfiable. In class we proved that 3SAT is NP-complete, using a reduction from CIRCUITSAT.

Now consider the related problem **2SAT**: Given a boolean formula in conjunctive normal form, with exactly *two* literals in each clause, is the formula satisfiable? For example, the following boolean formula is a valid input to 2SAT:

$$(x \vee y) \wedge (y \vee \bar{z}) \wedge (\bar{x} \vee z) \wedge (\bar{w} \vee y).$$

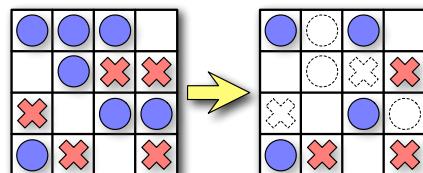
Either prove that 2SAT is NP-hard or describe a polynomial-time algorithm to solve it. [Hint: Recall that  $(x \vee y) \equiv (\bar{x} \rightarrow y)$ , and build a graph.]

- Let  $G = (V, E)$  be a graph. A *dominating set* in  $G$  is a subset  $S$  of the vertices such that every vertex in  $G$  is either in  $S$  or adjacent to a vertex in  $S$ . The DOMINATINGSET problem asks, given a graph  $G$  and an integer  $k$  as input, whether  $G$  contains a dominating set of size  $k$ . Either prove that this problem is NP-hard or describe a polynomial-time algorithm to solve it.

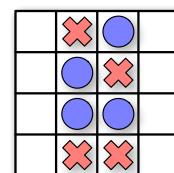


A dominating set of size 3 in the Peterson graph.

- Consider the following solitaire game. The puzzle consists of an  $n \times m$  grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) every row contains at least one stone, and (2) no column contains stones of both colors. For some initial configurations of stones, reaching this goal is impossible.



A solvable puzzle and one of its many solutions.



An unsolvable puzzle.

Prove that it is NP-hard to determine, given an initial configuration of red and blue stones, whether the puzzle can be solved.

This exam lasts 120 minutes.

**Write your answers in the separate answer booklet.**

Please return this question sheet with your answers.

1. Each of these ten questions has one of the following five answers:

$$\text{A: } \Theta(1) \quad \text{B: } \Theta(\log n) \quad \text{C: } \Theta(n) \quad \text{D: } \Theta(n \log n) \quad \text{E: } \Theta(n^2)$$

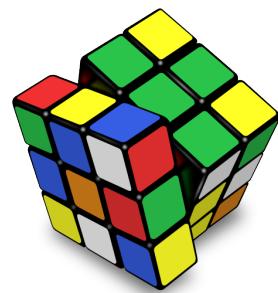
Choose the correct answer for each question. Each correct answer is worth +1 point; each incorrect answer is worth  $-1/2$  point; and each “I don’t know” is worth  $+1/4$  point. Negative scores will be recorded as 0.

- (a) What is  $\frac{3}{n} + \frac{n}{3}$ ?
- (b) What is  $\sum_{i=1}^n \frac{i}{n}$ ?
- (c) What is  $\sqrt{\sum_{i=1}^n i}$  ?
- (d) How many bits are required to write the number  $n!$  (the factorial of  $n$ ) in binary?
- (e) What is the solution to the recurrence  $E(n) = E(n-3) + 17n$ ?
- (f) What is the solution to the recurrence  $F(n) = 2F(n/4) + 6n$ ?
- (g) What is the solution to the recurrence  $G(n) = 9G(n/9) + 9n$ ?
- (h) What is the worst-case running time of quicksort?
- (i) Let  $X[1..n, 1..n]$  be a fixed array of numbers. Consider the following recursive function:

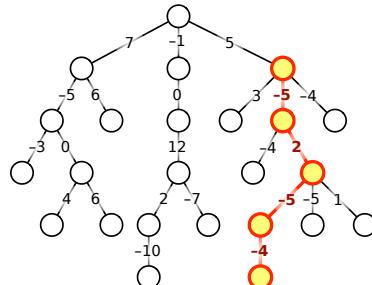
$$WTF(i, j) = \begin{cases} 0 & \text{if } \min\{i, j\} \leq 0 \\ -\infty & \text{if } \max\{i, j\} > n \\ X[i, j] + \max \left\{ \begin{array}{l} \left. \begin{array}{l} WTF(i-2, j+1) \\ WTF(i-2, j-1) \\ WTF(i-1, j-2) \\ WTF(i+1, j-2) \end{array} \right\} \text{ otherwise} \end{array} \right\} & \text{otherwise} \end{cases}$$

How long does it take to compute  $WTF(n, n)$  using dynamic programming?

- (j) The Rubik’s Cube is a mechanical puzzle invented in 1974 by Ernő Rubik, a Hungarian professor of architecture. The puzzle consists of a  $3 \times 3 \times 3$  grid of ‘cubelets’, whose faces are covered with stickers in six different colors. In the puzzle’s solved state, each face of the puzzle is one solid color. A mechanism inside the puzzle allows any face of the cube to be freely turned (as shown on the right). The puzzle can be scrambled by repeated turns. Given a scrambled Rubik’s Cube, how long does it take to find the *shortest* sequence of turns that returns the cube to its solved state?



2. Let  $T$  be a rooted tree with integer weights on its edges, which could be positive, negative, or zero. The weight of a path in  $T$  is the sum of the weights of its edges. Describe and analyze an algorithm to compute the minimum weight of any path from a node in  $T$  down to one of its descendants. It is not necessary to compute the actual minimum-weight path; just its weight. For example, given the tree shown below, your algorithm should return the number  $-12$ .



The minimum-weight downward path in this tree has weight  $-12$ .

3. Describe and analyze efficient algorithms to solve the following problems:

- (a) Given a set of  $n$  integers, does it contain two elements  $a, b$  such that  $a + b = 0$ ?
- (b) Given a set of  $n$  integers, does it contain three elements  $a, b, c$  such that  $a + b = c$ ?

4. A *common supersequence* of two strings  $A$  and  $B$  is another string that includes both the characters of  $A$  in order and the characters of  $B$  in order. Describe and analyze an algorithm to compute the length of the *shortest* common supersequence of two strings  $A[1..m]$  and  $B[1..n]$ . You do not need to compute an actual supersequence, just its length.

For example, if the input strings are ANTHROHOPOBIOLOGICAL and PRETERDIPLOMATICALLY, your algorithm should output 31, because a shortest common supersequence of those two strings is PREANTHROHODPOBIOPLOMATGICALLY.

5. [Taken directly from HBS0.] Recall that the *Fibonacci numbers*  $F_n$  are recursively defined as follows:  $F_0 = 0$ ,  $F_1 = 1$ , and  $F_n = F_{n-1} + F_{n-2}$  for every integer  $n \geq 2$ . The first few Fibonacci numbers are  $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \dots$

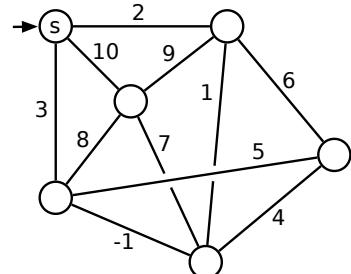
**Prove** that any non-negative integer can be written as the sum of distinct *non-consecutive* Fibonacci numbers. That is, if any Fibonacci number  $F_n$  appears in the sum, then its neighbors  $F_{n-1}$  and  $F_{n+1}$  do not. For example:

$$\begin{aligned} 88 &= 55 + 21 + 8 + 3 + 1 &= F_{10} + F_8 + F_6 + F_4 + F_2 \\ 42 &= 34 + 8 &= F_9 + F_6 \\ 17 &= 13 + 3 + 1 &= F_7 + F_4 + F_2 \end{aligned}$$

This exam lasts 120 minutes.  
**Write your answers in the separate answer booklet.**  
Please return this question sheet with your answers.

1. Find the following spanning trees for the weighted graph shown below.

- (a) A depth-first spanning tree rooted at  $s$ .
- (b) A breadth-first spanning tree rooted at  $s$ .
- (c) ~~A shortest path tree rooted at  $s$ . Oops!~~
- (d) A minimum spanning tree.



You do *not* need to justify your answers; just clearly indicate the edges of each spanning tree in your answer booklet. Yes, one of the edges has negative weight.

2. [Taken directly from HBS 6.] An *Euler tour* of a graph  $G$  is a walk that starts and ends at the same vertex and traverses every edge of  $G$  exactly once. *Prove* that a connected undirected graph  $G$  has an Euler tour if and only if every vertex in  $G$  has even degree.
3. You saw in class that the standard algorithm to INCREMENT a binary counter runs in  $O(1)$  amortized time. Now suppose we also want to support a second function called RESET, which resets all bits in the counter to zero.

Here are the INCREMENT and RESET algorithms. In addition to the array  $B[0.. \infty]$  of bits, we now also maintain the index of the most significant bit, in an integer variable  $msb$ .

```
INCREMENT( $B[0.. \infty]$ ,  $msb$ ):
   $i \leftarrow 0$ 
  while  $B[i] = 1$ 
     $B[i] \leftarrow 0$ 
     $i \leftarrow i + 1$ 
   $B[i] \leftarrow 1$ 
  if  $i > msb$ 
     $msb \leftarrow i$ 
```

```
RESET( $B[0.. \infty]$ ,  $msb$ ):
  for  $i \leftarrow 0$  to  $msb$ 
     $B[i] \leftarrow 0$ 
   $msb \leftarrow 0$ 
```

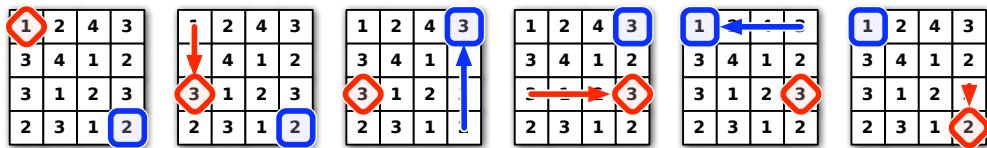
In parts (a) and (b), let  $n$  denote the number currently stored in the counter.

- (a) What is the worst-case running time of INCREMENT, as a function of  $n$ ?
- (b) What is the worst-case running time of RESET, as a function of  $n$ ?
- (c) *Prove* that in an arbitrary intermixed sequence of INCREMENT and RESET operations, the amortized time for each operation is  $O(1)$ .

4. The following puzzle was invented by the infamous Mongolian puzzle-warrior Vidrach Itky Leda in the year 1473. The puzzle consists of an  $n \times n$  grid of squares, where each square is labeled with a positive integer, and two tokens, one red and the other blue. The tokens always lie on distinct squares of the grid. The tokens start in the top left and bottom right corners of the grid; the goal of the puzzle is to swap the tokens.

In a single turn, you may move either token up, right, down, or left by a distance determined by the **other** token. For example, if the red token is on a square labeled 3, then you may move the blue token 3 steps up, 3 steps left, 3 steps right, or 3 steps down. However, you may not move a token off the grid or to the same square as the other token.

Describe and analyze an efficient algorithm that either returns the minimum number of moves required to solve a given Vidrach Itky Leda puzzle, or correctly reports that the puzzle has no solution. For example, given the puzzle below, your algorithm would return the number 5.



A five-move solution for a  $4 \times 4$  Vidrach Itky Leda puzzle.

5. Suppose you are given an array  $X[1..n]$  of real numbers chosen independently and uniformly at random from the interval  $[0, 1]$ . An array entry  $X[i]$  is called a *local maximum* if it is larger than its neighbors  $X[i - 1]$  and  $X[i + 1]$  (if they exist).

What is the *exact* expected number of local maxima in  $X$ ? **Prove** that your answer is correct. [Hint: Consider the special case  $n = 3$ .]

<b>0.7</b>	0.3	<b>1.0</b>	0.1	0.0	0.5	<b>0.6</b>	0.2	0.4	<b>0.9</b>	0.8
------------	-----	------------	-----	-----	-----	------------	-----	-----	------------	-----

A randomly filled array with 4 local maxima.

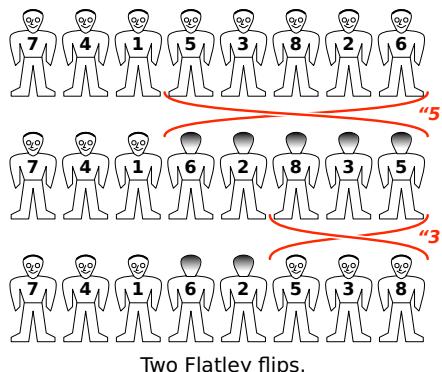
This exam lasts 180 minutes.

**Write your answers in the separate answer booklet.**

Please return this question sheet with your answers.

- Choreographer Michael Flatley has hired a new dance company to perform his latest Irish step-dancing extravaganza. At their first practice session, the new dancers line up in a row on stage and practice a movement called the *Flatley Flip*: Whenever Mr. Flatley calls out any positive integer  $k$ , the  $k$  rightmost dancers rotate 180 degrees as a group, so that their order in the line is reversed.

Each dancer wears a shirt with a positive integer printed on the front and back; different dancers have different numbers. Mr. Flatley wants to rearrange the dancers, using only a sequence of Flatley Flips, so that these numbers are sorted from left to right in increasing order.



- Describe an algorithm to sort an arbitrary row of  $n$  numbered dancers, using  $O(n)$  Flatley flips. (After sorting, the dancers may face forward, backward, or some of each.) Exactly how many flips does your algorithm perform in the worst case?<sup>1</sup>
- Describe an algorithm that sorts an arbitrary row of  $n$  numbered dancers and ensures that all dancers are facing forward, using  $O(n)$  Flatley flips. Exactly how many flips does your algorithm perform in the worst case?<sup>2</sup>
- You're in charge of choreographing a musical for your local community theater, and it's time to figure out the final pose of the big show-stopping number at the end. ("Streetcar!") You've decided that each of the  $n$  cast members in the show will be positioned in a big line when the song finishes, all with their arms extended and showing off their best spirit fingers.

The director has declared that during the final flourish, each cast member must either point both their arms up or point both their arms down; it's your job to figure out who points up and who points down. Moreover, in a fit of unchecked power, the director has also given you a list of arrangements that will upset his delicate artistic temperament. Each forbidden arrangement is a subset of cast members paired with arm positions; for example: "Marge may not point her arms up while Ned and Apu point their arms down."

Prove that finding an acceptable arrangement of arm positions is NP-hard. [Hint: Describe a reduction from 3SAT.]

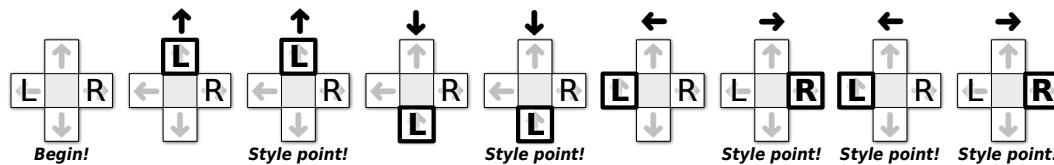
<sup>1</sup>This is really a question about networking.

<sup>2</sup>This is really a question about mutating DNA.

3. *Dance Dance Revolution* is a dance video game, first introduced in Japan by Konami in 1998. Players stand on a platform marked with four arrows, pointing forward, back, left, and right, arranged in a cross pattern. During play, the game plays a song and scrolls a sequence of  $n$  arrows ( $\leftarrow$ ,  $\uparrow$ ,  $\downarrow$ , or  $\rightarrow$ ) from the bottom to the top of the screen. At the precise moment each arrow reaches the top of the screen, the player must step on the corresponding arrow on the dance platform. (The arrows are timed so that you'll step with the beat of the song.)

You are playing a variant of this game called “Vogue Vogue Revolution”, where the goal is to play perfectly but move as little as possible. When an arrow reaches the top of the screen, if one of your feet is already on the correct arrow, you are awarded one style point for maintaining your current pose. If neither foot is on the right arrow, you must move one (and *only* one) of your feet from its current location to the correct arrow on the platform. If you ever step on the wrong arrow, or fail to step on the correct arrow, or move more than one foot at a time, all your style points are taken away and the game ends.

How should you move your feet to maximize your total number of style points? For purposes of this problem, assume you always start with your left foot on  $\leftarrow$  and your right foot on  $\rightarrow$ , and that you've memorized the entire sequence of arrows. For example, if the sequence is  $\uparrow\uparrow\downarrow\downarrow\leftarrow\rightarrow\leftarrow\rightarrow$ , you can earn 5 style points by moving your feet as shown below:



- (a) *Prove* that for *any* sequence of  $n$  arrows, it is possible to earn at least  $n/4 - 1$  style points.
- (b) Describe an efficient algorithm to find the maximum number of style points you can earn during a given VVR routine.<sup>3</sup> The input to your algorithm is an array  $Arrow[1..n]$  containing the sequence of arrows. [Hint: Build a graph!]
4. It's almost time to show off your flippin' sweet dancing skills! Tomorrow is the big dance contest you've been training for your entire life, except for that summer you spent with your uncle in Alaska hunting wolverines. You've obtained an advance copy of the list of  $n$  songs that the judges will play during the contest, in chronological order.

You know all the songs, all the judges, and your own dancing ability extremely well. For each integer  $k$ , you know that if you dance to the  $k$ th song on the schedule, you will be awarded exactly  $Score[k]$  points, but then you will be physically unable to dance for the next  $Wait[k]$  songs (that is, you cannot dance to songs  $k + 1$  through  $k + Wait[k]$ ). The dancer with the highest total score at the end of the night wins the contest, so you want your total score to be as high as possible.

Describe and analyze an efficient algorithm to compute the maximum total score you can achieve. The input to your sweet algorithm is the pair of arrays  $Score[1..n]$  and  $Wait[1..n]$ .<sup>4</sup>

---

<sup>3</sup>This is really a question about paging.

<sup>4</sup>This is really a question about processor scheduling.

5. You're organizing the First Annual UIUC Computer Science 72-Hour Dance Exchange, to be held all day Friday, Saturday, and Sunday. Several 30-minute sets of music will be played during the event, and a large number of DJs have applied to perform. You need to hire DJs according to the following constraints.

- Exactly  $k$  sets of music must be played each day, and thus  $3k$  sets altogether.
- Each set must be played by a single DJ in a consistent music genre (ambient, bubblegum, dubstep, horrorcore, hyphy, trip-hop, Nitzhonot, Kwaito, J-pop, Nashville country, ...).
- Each genre must be played at most once per day.
- Each candidate DJ has given you a list of genres they are willing to play.
- Each DJ can play at most three sets during the entire event.

Suppose there are  $n$  candidate DJs and  $g$  different musical genres available. Describe and analyze an efficient algorithm that either assigns a DJ and a genre to each of the  $3k$  sets, or correctly reports that no such assignment is possible.

6. You've been put in charge of putting together a team for the “Dancing with the Computer Scientists” international competition. Good teams in this competition must be capable of performing a wide variety of dance styles. You are auditioning a set of  $n$  dancing computer scientists, each of whom specializes in a particular style of dance.

Describe an algorithm to determine in  $O(n)$  time if *more than half* of the  $n$  dancers specialize exactly in the same dance style. The input to your algorithm is an array of  $n$  positive integers, where each integer identifies a style: 1 = ballroom, 2 = latin, 3 = swing, 4 = b-boy, 42 = contact improv, 101 = peanut butter jelly time, and so on. [Hint: Remember the *SELECT* algorithm!]

7. The party you are attending is going great, but now it's time to line up for *The Algorithm March* (アルゴリズムこうしん)! This dance was originally developed by the Japanese comedy duo Itsumo Kokokara (いつもここから) for the children's television show PythagoraSwitch (ピタゴラスイッチ). The Algorithm March is performed by a line of people; each person in line starts a specific sequence of movements one measure later than the person directly in front of them. Thus, the march is the dance equivalent of a musical round or canon, like “Row Row Row Your Boat”.

Proper etiquette dictates that each marcher must know the person directly in front of them in line, lest a minor mistake during lead to horrible embarrassment between strangers. Suppose you are given a complete list of which people at your party know each other. **Prove** that it is NP-hard to determine the largest number of party-goers that can participate in the Algorithm March.<sup>5</sup>

---

<sup>5</sup>This is really a question about ninjas.

**You may assume the following problems are NP-hard:**

**CIRCUITSAT:** Given a boolean circuit, are there any input values that make the circuit output True?

**PLANAR CIRCUITSAT:** Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output True?

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

**MAXINDEPENDENTSET:** Given an undirected graph  $G$ , what is the size of the largest subset of vertices in  $G$  that have no edges among them?

**MAXCLIQUE:** Given an undirected graph  $G$ , what is the size of the largest complete subgraph of  $G$ ?

**MINVERTEXCOVER:** Given an undirected graph  $G$ , what is the size of the smallest subset of vertices that touch every edge in  $G$ ?

**MINDOMINATINGSET:** Given an undirected graph  $G$ , what is the size of the smallest subset  $S$  of vertices such that every vertex in  $G$  is either in  $S$  or adjacent to a vertex in  $S$ ?

**MINSETCOVER:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subcollection whose union is  $S$ ?

**MINHITTINGSET:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subset of  $S$  that intersects every subset  $S_i$ ?

**3COLOR:** Given an undirected graph  $G$ , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

**CHROMATICNUMBER:** Given an undirected graph  $G$ , what is the minimum number of colors needed to color its vertices, so that every edge touches vertices with two different colors?

**MAXCUT:** Given a graph  $G$ , what is the size (number of edges) of the largest bipartite subgraph of  $G$ ?

**HAMILTONIANCYCLE:** Given a graph  $G$ , is there a cycle in  $G$  that visits every vertex exactly once?

**HAMILTONIANPATH:** Given a graph  $G$ , is there a path in  $G$  that visits every vertex exactly once?

**TRAVELINGSALESMAN:** Given a graph  $G$  with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in  $G$ ?

**SUBSETSUM:** Given a set  $X$  of positive integers and an integer  $k$ , does  $X$  have a subset whose elements sum to  $k$ ?

**PARTITION:** Given a set  $X$  of positive integers, can  $X$  be partitioned into two subsets with the same sum?

**3PARTITION:** Given a set  $X$  of  $n$  positive integers, can  $X$  be partitioned into  $n/3$  three-element subsets, all with the same sum?

**MINESWEEPER:** Given a Minesweeper configuration and a particular square  $x$ , is it safe to click on  $x$ ?

**TETRIS:** Given a sequence of  $N$  Tetris pieces and a partially filled  $n \times k$  board, is it possible to play every piece in the sequence without overflowing the board?

**SUDOKU:** Given an  $n \times n$  Sudoku puzzle, does it have a solution?

**KENKEN:** Given an  $n \times n$  Ken-Ken puzzle, does it have a solution?

# CS 573: Graduate Algorithms, Fall 2010

## Homework 0

Due Wednesday, September 1, 2010 in class

---

- This homework tests your familiarity with prerequisite material (<http://www.cs.uiuc.edu/class/fa10/cs573/stuff-you-already-know.html>) to help you identify gaps in your background knowledge. **You are responsible for filling those gaps.** For most topics, the early chapters of any algorithms textbook should be sufficient review, but you may also want consult your favorite discrete mathematics and data structures textbooks. If you need help, please ask in office hours and/or on the course newsgroup.
- Each student must submit individual solutions for these homework problems. For all future homeworks, groups of up to three students may submit (or present) a single group solution for each problem.
- Please carefully read the course policies linked from the course web site. If you have *any* questions, please ask during lecture or office hours, or post your question to the course newsgroup. In particular:
  - Submit five separately stapled solutions, one for each numbered problem, with your name and NetID clearly printed on each page. Please do *not* staple everything together.
  - You may use any source at your disposal—paper, electronic, or human—but you **must** write your solutions in your own words, and you **must** cite every source that you use. In particular, each solution should include a list of *everyone* you worked with to solve that problem.
  - Unless explicitly stated otherwise, **every** homework problem requires a proof.
  - Answering “I don’t know” to any homework or exam problem (except for extra credit problems) is worth 25% partial credit.
  - Algorithms or proofs containing phrases like “and so on” or “repeat this process for all  $n$ ” instead of an explicit loop, recursion, or induction, will receive 0 points.

1. (•) ***Write the sentence “I understand the course policies.”***

Solutions that omit this sentence will not be graded.

- (a) [5 pts] Solve the following recurrences. State tight asymptotic bounds for each function in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ . Assume reasonable but nontrivial base cases if none are given. ***Do not submit proofs***—just a list of five functions—but you should do them anyway, just for practice.

- $A(n) = 4A(n-1) + 1$
- $B(n) = B(n-3) + n^2$
- $C(n) = 2C(n/2) + 3C(n/3) + n^2$
- $D(n) = 2D(n/3) + \sqrt{n}$

$$\bullet E(n) = \begin{cases} n & \text{if } n \leq 3, \\ \frac{E(n-1)E(n-2)}{E(n-3)} & \text{otherwise} \end{cases} \quad [\text{Hint: This is easier than it looks!}]$$

- (b) [5 pts] Sort the following functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. ***Do not submit proofs***—just a sorted list of 16 functions—but you should do them anyway, just for practice.

Write  $f(n) \ll g(n)$  to indicate that  $f(n) = o(g(n))$ , and write  $f(n) \equiv g(n)$  to mean  $f(n) = \Theta(g(n))$ . We use the notation  $\lg n = \log_2 n$ .

$n$	$\lg n$	$\sqrt{n}$	$7^n$
$\sqrt{\lg n}$	$\lg \sqrt{n}$	$7^{\sqrt{n}}$	$\sqrt{7^n}$
$7^{\lg n}$	$\lg(7^n)$	$7^{\lg \sqrt{n}}$	$7^{\sqrt{\lg n}}$
$\sqrt{7^{\lg n}}$	$\lg(7^{\sqrt{n}})$	$\lg \sqrt{7^n}$	$\sqrt{\lg(7^n)}$

2. Professore Giorgio della Giungla has a 23-node binary tree, in which every node is labeled with a unique letter of the Roman alphabet, which is just like the modern English alphabet, but without the letters **J**, **U**, and **W**. Inorder and postorder traversals of the tree visit the nodes in the following order:

- Inorder: **S V Z A T P R D B X O L F E H I Q M N G Y K C**
- Postorder: **A Z P T X B D L E F O H R I V N M K C Y G Q S**

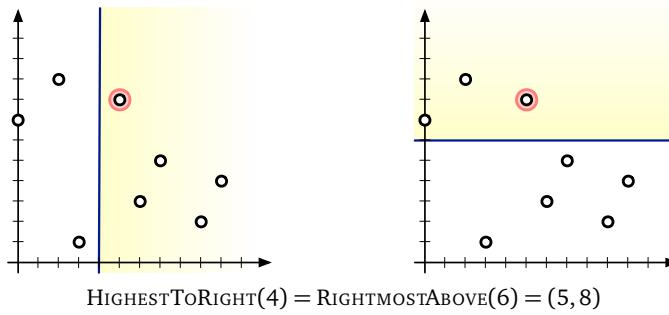
- (a) List the nodes in Prof. della Giungla’s tree in the order visited by a *preorder* traversal.  
 (b) Draw Prof. della Giungla’s tree.

3. The original version of this problem asked to support the mirror-image operations `LOWESTToRIGHT` and `LEFTMOSTABOVE`, which are *much* harder to support with a single data structure that stores each point at most once. We will accept  $O(n)$ -space data structures for either version of the problem for full credit.

Describe a data structure that stores a set  $S$  of  $n$  points in the plane, each represented by a pair  $(x, y)$  of coordinates, and supports the following queries.

- **HIGHESTToRIGHT( $\ell$ )**: Return the highest point in  $S$  whose  $x$ -coordinate is greater than or equal to  $\ell$ . If every point in  $S$  has  $x$ -coordinate less than  $\ell$ , return `NONE`.
- **RIGHTMOSTABOVE( $\ell$ )**: Return the rightmost point in  $S$  whose  $y$ -coordinate is greater than or equal to  $\ell$ . If every point in  $S$  has  $y$ -coordinate less than  $\ell$ , return `NONE`.

For example, if  $S = \{(3, 1), (1, 9), (9, 2), (6, 3), (5, 8), (7, 5), (10, 4), (0, 7)\}$ , then both `HIGHESTToRIGHT(4)` and `RIGHTMOSTABOVE(6)` should return the point  $(5, 8)$ , and `HIGHESTToRIGHT(15)` should return `NONE`.



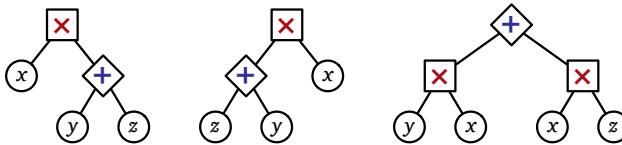
Analyze both the size of your data structure and the running times of your query algorithms. For full credit, your data structure should use  $O(n)$  space, and each query algorithm should run in  $O(\log n)$  time. **For 5 extra credit points, describe a data structure that stores each point at most once.** You may assume that no two points in  $S$  have equal  $x$ -coordinates or equal  $y$ -coordinates.

[Hint: Modify one of the standard data structures listed at <http://www.cs.uiuc.edu/class/fa10/cs573/stuff-you-already-know.html>, but just describe your changes; don't regurgitate the details of the standard data structure.]

4. An **arithmetic expression tree** is a binary tree where every leaf is labeled with a variable, every internal node is labeled with an arithmetic operation, and every internal node has exactly two children. For this problem, assume that the only allowed operations are  $+$  and  $\times$ . Different leaves may or may not represent different variables.

Every arithmetic expression tree represents a function, transforming input values for the leaf variables into an output value for the root, by following two simple rules: (1) The value of any  $+$ -node is the sum of the values of its children. (2) The value of any  $\times$ -node is the product of the values of its children.

Two arithmetic expression trees are **equivalent** if they represent the same function; that is, the same input values for the leaf variables always leads to the same output value at both roots.



Three equivalent expression trees. Only the third tree is in normal form.

An arithmetic expression tree is in *normal form* if the parent of every  $+$ -node (if any) is another  $+$ -node.

Prove that for any arithmetic expression tree, there is an equivalent arithmetic expression tree in normal form. [Hint: Be careful. This is trickier than it looks.]

5. Recall that a standard (Anglo-American) deck of 52 playing cards contains 13 cards in each of four suits: spades ( $\spadesuit$ ), hearts ( $\heartsuit$ ), diamonds ( $\diamondsuit$ ), and clubs ( $\clubsuit$ ). Within each suit, the 13 cards have distinct *ranks*: 2, 3, 4, 5, 6, 7, 8, 9, 10, jack ( $J$ ), queen ( $Q$ ), king ( $K$ ), and ace ( $A$ ). The ranks are ordered  $2 < 3 < \dots < 9 < 10 < J < Q < K < A$ ; thus, for example, the jack of spades has higher rank than the eight of diamonds.

Professor Jay is about to perform a public demonstration with two decks of cards, one with red backs ('the red deck') and one with blue backs ('the blue deck'). Both decks lie face-down on a table in front of Professor Jay, shuffled uniformly and independently. Thus, in each deck, every permutation of the 52 cards is equally likely.

To begin the demonstration, Professor Jay turns over the top card from each deck. Then, while he has not yet turned over a three of clubs (3 $\clubsuit$ ), the good Professor hurls the two cards he just turned over into the *thick, pachydermatous outer melon layer* of a nearby watermelon (that most prodigious of household fruits) and then turns over the next card from the top of each deck. Thus, if 3 $\clubsuit$  is the last card in both decks, the demonstration ends with 102 cards embedded in the watermelon.

- (a) What is the *exact* expected number of cards that Professor Jay hurls into the watermelon?
- (b) For each of the statements below, give the *exact* probability that the statement is true of the *first* pair of cards Professor Jay turns over.
  - i. Both cards are threes.
  - ii. One card is a three, and the other card is a club.
  - iii. If (at least) one card is a heart, then (at least) one card is a diamond.
  - iv. The card from the red deck has higher rank than the card from the blue deck.
- (c) For each of the statements below, give the *exact* probability that the statement is true of the *last* pair of cards Professor Jay turns over.
  - i. Both cards are threes.
  - ii. One card is a three, and the other card is a club.
  - iii. If (at least) one card is a heart, then (at least) one card is a diamond.
  - iv. The card from the red deck has higher rank than the card from the blue deck.

Express each of your answers as rational numbers in simplest form, like 123/4567. ***Do not submit proofs***—just a list of rational numbers—but you should do them anyway, just for practice.

# CS 573: Graduate Algorithms, Fall 2010

## Homework 1

Due Friday, September 10, 2010 at 1pm

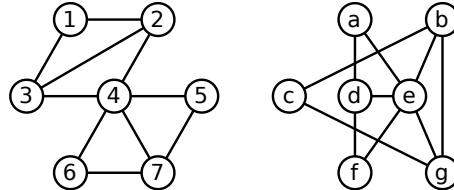
Due Monday, September 13, 2010 at 5pm  
(in the homework drop boxes in the basement of Siebel)

---

For this and all future homeworks, groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name and NetID on each page of your submission.

---

1. Two graphs are said to be **isomorphic** if one can be transformed into the other just by relabeling the vertices. For example, the graphs shown below are isomorphic; the left graph can be transformed into the right graph by the relabeling  $(1, 2, 3, 4, 5, 6, 7) \mapsto (c, g, b, e, a, f, d)$ .



Two isomorphic graphs.

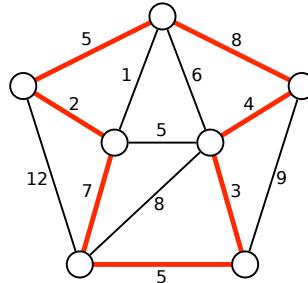
Consider the following related decision problems:

- **GRAPHISOMORPHISM:** Given two graphs  $G$  and  $H$ , determine whether  $G$  and  $H$  are isomorphic.
  - **EVENGRAPHISOMORPHISM:** Given two graphs  $G$  and  $H$ , such that every vertex in  $G$  and  $H$  has even degree, determine whether  $G$  and  $H$  are isomorphic.
  - **SUBGRAPHISOMORPHISM:** Given two graphs  $G$  and  $H$ , determine whether  $G$  is isomorphic to a subgraph of  $H$ .
- (a) Describe a polynomial-time reduction from **EVENGRAPHISOMORPHISM** to **GRAPHISOMORPHISM**.
  - (b) Describe a polynomial-time reduction from **GRAPHISOMORPHISM** to **EVENGRAPHISOMORPHISM**.
  - (c) Describe a polynomial-time reduction from **GRAPHISOMORPHISM** to **SUBGRAPHISOMORPHISM**.
  - (d) Prove that **SUBGRAPHISOMORPHISM** is NP-complete.
  - (e) What can you conclude about the NP-hardness of **GRAPHISOMORPHISM**? Justify your answer.

[Hint: These are all easy!]

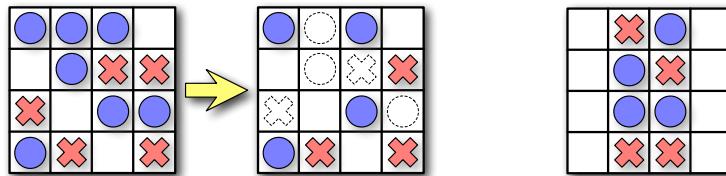
2. Suppose you are given a magic black box that can solve the **3COLORABLE** problem **in polynomial time**. That is, given an arbitrary graph  $G$  as input, the magic black box returns TRUE if  $G$  has a proper 3-coloring, and returns FALSE otherwise. Describe and analyze a **polynomial-time** algorithm that computes an actual proper 3-coloring of a given graph  $G$ , or correctly reports that no such coloring exists, using this magic black box as a subroutine. [Hint: The input to the black box is a graph. Just a graph. Nothing else.]

3. Let  $G$  be an undirected graph with weighted edges. A *heavy Hamiltonian cycle* is a cycle  $C$  that passes through each vertex of  $G$  exactly once, such that the total weight of the edges in  $C$  is at least half of the total weight of all edges in  $G$ . Prove that deciding whether a graph has a heavy Hamiltonian cycle is NP-complete.



A heavy Hamiltonian cycle. The cycle has total weight 34; the graph has total weight 67.

4. Consider the following solitaire game. The puzzle consists of an  $n \times m$  grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions: (1) every row contains at least one stone, and (2) no column contains stones of both colors. For some initial configurations of stones, reaching this goal is impossible.



A solvable puzzle and one of its many solutions.

An unsolvable puzzle.

Prove that it is NP-hard to determine, given an initial configuration of red and blue stones, whether the puzzle can be solved.

5. A boolean formula in *exclusive-or conjunctive normal form* (XCNF) is a conjunction (AND) of several clauses, each of which is the exclusive-or of one or more literals. For example:

$$(u \oplus v \oplus \bar{w} \oplus x) \wedge (\bar{u} \oplus \bar{w} \oplus y) \wedge (\bar{v} \oplus y) \wedge (\bar{u} \oplus \bar{v} \oplus x \oplus y) \wedge (w \oplus x) \wedge y$$

The XCNF-SAT problem asks whether a given XCNF boolean formula is satisfiable. Either describe a polynomial-time algorithm for XCNF-SAT or prove that it is NP-complete.

# CS 573: Graduate Algorithms, Fall 2010

## Homework 2

Due Monday, September 27, 2010 at 5pm  
(in the homework drop boxes in the basement of Siebel)

---

- For this and all future homeworks, groups of up to three students may submit a single, common solution. Please neatly print (or typeset) the full name and NetID of every group member on the first page of your submission.
- We will use the following rubric to grade all dynamic programming algorithms:
  - 60% for a correct recurrence (including base cases and a plain-English specification); no credit for anything else if this is wrong.
  - 10% for describing a suitable memoization data structure.
  - 20% for describing a correct evaluation order. (A clear picture is sufficient.)
  - 10% point for analyzing the running time of the resulting algorithm.

Official solutions will always include pseudocode for the final dynamic programming algorithm, but this is **not** required for full credit. However, if you do provide correct pseudocode for the dynamic programming algorithm, it is not necessary to separately describe the recurrence, the memoization data structure, or the evaluation order.

It is **not** necessary to state a space bound. There is no penalty for using more space than the official solution, but +1 extra credit for using less space with the same (or better) running time.

- The official solution for every problem will provide a target time bound. Algorithms faster than the official solution are worth more points (as extra credit); algorithms slower than the official solution are worth fewer points. For slower algorithms, partial credit is scaled to the lower maximum score. For example, if a full dynamic programming algorithm would be worth 5 points, just the recurrence is worth 3 points. However, incorrect algorithms are worth zero points, no matter how fast they are.
  - Greedy algorithms *must* be accompanied by proofs of correctness in order to receive *any* credit. Otherwise, **any correct algorithm, no matter how slow, is worth at least 2½ points**, assuming it is properly analyzed.
- 

1. Suppose you are given an array  $A[1..n]$  of positive integers. Describe and analyze an algorithm to find the *smallest* positive integer that is *not* an element of  $A$  in  $O(n)$  time.
2. Suppose you are given an  $m \times n$  bitmap, represented by an array  $M[1..m, 1..n]$  whose entries are all 0 or 1. A *solid block* is a subarray of the form  $M[i..i', j..j']$  in which every bit is equal to 1. Describe and analyze an efficient algorithm to find a solid block in  $M$  with maximum area.

3. Let  $T$  be a tree in which each edge  $e$  has a weight  $w(e)$ . A *matching*  $M$  in  $T$  is a subset of the edges such that each vertex of  $T$  is incident to at most one edge in  $M$ . The weight of a matching  $M$  is the sum of the weights of its edges. Describe and analyze an algorithm to compute a maximum weight matching, given the tree  $T$  as input.
4. For any string  $x$  and any non-negative integer  $k$ , let  $x^k$  denote the string obtained by concatenating  $k$  copies of  $x$ . For example,  $\text{STRING}^3 = \text{STRINGSTRINGSTRING}$  and  $\text{STRING}^0$  is the empty string.

A *repetition* of  $x$  is a prefix of  $x^k$  for some integer  $k$ . For example,  $\text{STRINGSTRINGSTRINGST}$  and  $\text{STR}$  are both repetitions of  $\text{STRING}$ , as is the empty string.

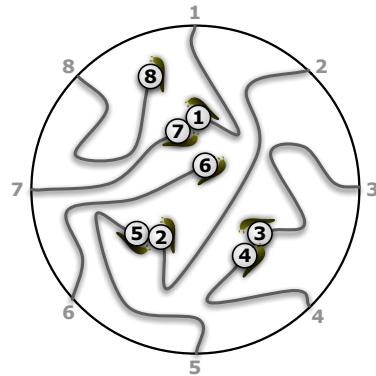
An *interleaving* of two strings  $x$  and  $y$  is any string obtained by shuffling a repetition of  $x$  with a repetition of  $y$ . For example,  $\text{STRWORDINDGSTWORDIRNGDWSTORR}$  is an interleaving of  $\text{STRING}$  and  $\text{WORD}$ , as is the empty string.

Describe and analyze an algorithm that accepts three strings  $x$ ,  $y$ , and  $z$  as input, and decides whether  $z$  is an interleaving of  $x$  and  $y$ .

5. Every year, as part of its annual meeting, the Antarctic Snail Lovers of Upper Glacierville hold a Round Table Mating Race. Several high-quality breeding snails are placed at the edge of a round table. The snails are numbered in order around the table from 1 to  $n$ . During the race, each snail wanders around the table, leaving a trail of slime behind it. The snails have been specially trained never to fall off the edge of the table or to cross a slime trail, even their own. If two snails meet, they are declared a breeding pair, removed from the table, and whisked away to a romantic hole in the ground to make little baby snails. Note that some snails may never find a mate, even if the race goes on forever.

For every pair of snails, the Antarctic SLUG race organizers have posted a monetary reward, to be paid to the owners if that pair of snails meets during the Mating Race. Specifically, there is a two-dimensional array  $M[1..n, 1..n]$  posted on the wall behind the Round Table, where  $M[i, j] = M[j, i]$  is the reward to be paid if snails  $i$  and  $j$  meet.

Describe and analyze an algorithm to compute the maximum total reward that the organizers could be forced to pay, given the array  $M$  as input.



The end of a typical Antarctic SLUG race. Snails 6 and 8 never find mates.

The organizers must pay  $M[3, 4] + M[2, 5] + M[1, 7]$ .

# CS 573: Graduate Algorithms, Fall 2010

## Homework 3

Due Monday, October 18, 2010 at 5pm  
 (in the homework drop boxes in the basement of Siebel)

---

1. Suppose we are given two arrays  $C[1..n]$  and  $R[1..n]$  of positive integers. An  $n \times n$  matrix of 0s and 1s *agrees with R and C* if, for every index  $i$ , the  $i$ th row contains  $R[i]$  1s, and the  $i$ th column contains  $C[i]$  1s. Describe and analyze an algorithm that either constructs a matrix that agrees with  $R$  and  $C$ , or correctly reports that no such matrix exists.
2. Suppose we have  $n$  skiers with heights given in an array  $P[1..n]$ , and  $n$  skis with heights given in an array  $S[1..n]$ . Describe an efficient algorithm to assign a ski to each skier, so that the average difference between the height of a skier and her assigned ski is as small as possible. The algorithm should compute a permutation  $\sigma$  such that the expression

$$\frac{1}{n} \sum_{i=1}^n |P[i] - S[\sigma(i)]|$$

is as small as possible.

3. Alice wants to throw a party and she is trying to decide who to invite. She has  $n$  people to choose from, and she knows which pairs of these people know each other. She wants to pick as many people as possible, subject to two constraints:

- For each guest, there should be at least five other guests that they already know.
- For each guest, there should be at least five other guests that they *don't* already know.

Describe and analyze an algorithm that computes the largest possible number of guests Alice can invite, given a list of  $n$  people and the list of pairs who know each other.

4. Consider the following heuristic for constructing a vertex cover of a connected graph  $G$ : return the set of non-leaf nodes in any depth-first spanning tree of  $G$ .
  - (a) Prove that this heuristic returns a vertex cover of  $G$ .
  - (b) Prove that this heuristic returns a 2-approximation to the minimum vertex cover of  $G$ .
  - (c) Describe an infinite family of graphs for which this heuristic returns a vertex cover of size  $2 \cdot OPT$ .
5. Suppose we want to route a set of  $N$  calls on a telecommunications network that consist of a cycle on  $n$  nodes, indexed in order from 0 to  $n - 1$ . Each call has a source node and a destination node, and can be routed either clockwise or counterclockwise around the cycle. Our goal is to route the calls so as to minimize the overall load on the network. The load  $L_i$  on any edge  $(i, (i + 1) \bmod n)$  is the number of calls routed through that edge, and the overall load is  $\max_i L_i$ . Describe and analyze an efficient 2-approximation algorithm for this problem.

# CS 573: Graduate Algorithms, Fall 2010

## Homework 4

Due Monday, November 1, 2010 at 5pm  
(in the homework drop boxes in the basement of Siebel)

---

1. Consider an  $n$ -node treap  $T$ . As in the lecture notes, we identify nodes in  $T$  by the ranks of their search keys. Thus, ‘node 5’ means the node with the 5th smallest search key. Let  $i, j, k$  be integers such that  $1 \leq i \leq j \leq k \leq n$ .
  - (a) What is the *exact* probability that node  $j$  is a common ancestor of node  $i$  and node  $k$ ?
  - (b) What is the *exact* expected length of the unique path from node  $i$  to node  $k$  in  $T$ ?
2. Let  $M[1..n, 1..n]$  be an  $n \times n$  matrix in which every row and every column is sorted. Such an array is called *totally monotone*. No two elements of  $M$  are equal.
  - (a) Describe and analyze an algorithm to solve the following problem in  $O(n)$  time: Given indices  $i, j, i', j'$  as input, compute the number of elements of  $M$  smaller than  $M[i, j]$  and larger than  $M[i', j']$ .
  - (b) Describe and analyze an algorithm to solve the following problem in  $O(n)$  time: Given indices  $i, j, i', j'$  as input, return an element of  $M$  chosen uniformly at random from the elements smaller than  $M[i, j]$  and larger than  $M[i', j']$ . Assume the requested range is always non-empty.
  - (c) Describe and analyze a randomized algorithm to compute the median element of  $M$  in  $O(n \log n)$  expected time.
3. Suppose we are given a complete undirected graph  $G$ , in which each edge is assigned a weight chosen independently and uniformly at random from the real interval  $[0, 1]$ . Consider the following greedy algorithm to construct a Hamiltonian cycle in  $G$ . We start at an arbitrary vertex. While there is at least one unvisited vertex, we traverse the minimum-weight edge from the current vertex to an unvisited neighbor. After  $n - 1$  iterations, we have traversed a Hamiltonian path; to complete the Hamiltonian cycle, we traverse the edge from the last vertex back to the first vertex. What is the expected weight of the resulting Hamiltonian cycle? [Hint: What is the expected weight of the first edge? Consider the case  $n = 3$ .]

4. (a) Consider the following deterministic algorithm to construct a vertex cover  $C$  of a graph  $G$ .

```
VERTEXCOVER( $G$ ):
 $C \leftarrow \emptyset$ 
while  $C$  is not a vertex cover
    pick an arbitrary edge  $uv$  that is not covered by  $C$ 
    add either  $u$  or  $v$  to  $C$ 
return  $C$ 
```

Prove that VERTEXCOVER can return a vertex cover that is  $\Omega(n)$  times larger than the smallest vertex cover. You need to describe both an input graph with  $n$  vertices, for any integer  $n$ , and the sequence of edges and endpoints chosen by the algorithm.

- (b) Now consider the following randomized variant of the previous algorithm.

```
RANDOMVERTEXCOVER( $G$ ):
 $C \leftarrow \emptyset$ 
while  $C$  is not a vertex cover
    pick an arbitrary edge  $uv$  that is not covered by  $C$ 
    with probability  $1/2$ 
        add  $u$  to  $C$ 
    else
        add  $v$  to  $C$ 
return  $C$ 
```

Prove that the expected size of the vertex cover returned by RANDOMVERTEXCOVER is at most  $2 \cdot \text{OPT}$ , where OPT is the size of the smallest vertex cover.

- (c) Let  $G$  be a graph in which each vertex  $v$  has a weight  $w(v)$ . Now consider the following randomized algorithm that constructs a vertex cover.

```
RANDOMWEIGHTEDVERTEXCOVER( $G$ ):
 $C \leftarrow \emptyset$ 
while  $C$  is not a vertex cover
    pick an arbitrary edge  $uv$  that is not covered by  $C$ 
    with probability  $w(v)/(w(u)+w(v))$ 
        add  $u$  to  $C$ 
    else
        add  $v$  to  $C$ 
return  $C$ 
```

Prove that the expected weight of the vertex cover returned by RANDOMWEIGHTEDVERTEXCOVER is at most  $2 \cdot \text{OPT}$ , where OPT is the weight of the minimum-weight vertex cover. A correct answer to this part automatically earns full credit for part (b).

5. (a) Suppose  $n$  balls are thrown uniformly and independently at random into  $m$  bins. For any integer  $k$ , what is the *exact* expected number of bins that contain exactly  $k$  balls?
- (b) Consider the following balls and bins experiment, where we repeatedly throw a fixed number of balls randomly into a shrinking set of bins. The experiment starts with  $n$  balls and  $n$  bins. In each round  $i$ , we throw  $n$  balls into the remaining bins, and then discard any non-empty bins; thus, only bins that are empty at the end of round  $i$  survive to round  $i + 1$ .

```
BALLSDESTROYBINS( $n$ ):
  start with  $n$  empty bins
  while any bins remain
    throw  $n$  balls randomly into the remaining bins
    discard all bins that contain at least one ball
```

Suppose that in every round, *precisely* the expected number of bins are empty. Prove that under these conditions, the experiment ends after  $O(\log^* n)$  rounds.<sup>1</sup>

- \*(c) [Extra credit] Now assume that the balls are really thrown randomly into the bins in each round. Prove that with high probability, BALLSDESTROYBINS( $n$ ) ends after  $O(\log^* n)$  rounds.
- (d) Now consider a variant of the previous experiment in which we discard balls instead of bins. Again, the experiment  $n$  balls and  $n$  bins. In each round  $i$ , we throw the remaining balls into  $n$  bins, and then discard any ball that lies in a bin by itself; thus, only balls that collide in round  $i$  survive to round  $i + 1$ .

```
BINSDESTROYSINGLEBALLS( $n$ ):
  start with  $n$  balls
  while any balls remain
    throw the remaining balls randomly into  $n$  bins
    discard every ball that lies in a bin by itself
    retrieve the remaining balls from the bins
```

Suppose that in every round, *precisely* the expected number of bins contain exactly one ball. Prove that under these conditions, the experiment ends after  $O(\log \log n)$  rounds.

- \*(e) [Extra credit] Now assume that the balls are really thrown randomly into the bins in each round. Prove that with high probability, BINSDESTROYSINGLEBALLS( $n$ ) ends after  $O(\log \log n)$  rounds.

---

<sup>1</sup>Recall that the iterated logarithm is defined as follows:  $\log^* n = 0$  if  $n \leq 1$ , and  $\log^* n = 1 + \log^*(\lg n)$  otherwise.

# CS 573: Graduate Algorithms, Fall 2010

## Homework 5

Due Friday, November 19, 2010 at 5pm  
 (in the homework drop boxes in the basement of Siebel)

---

- Suppose we are given a set of boxes, each specified by their height, width, and depth in centimeters. All three side lengths of every box lie strictly between 10cm and 20cm. As you should expect, one box can be placed inside another if the smaller box can be rotated so that its height, width, and depth are respectively smaller than the height, width, and depth of the larger box. Boxes can be nested recursively. Call a box is *visible* if it is not inside another box.

Describe and analyze an algorithm to nest the boxes so that the number of visible boxes is as small as possible.

- Suppose we are given an array  $A[1..m][1..n]$  of non-negative real numbers. We want to *round*  $A$  to an integer matrix, by replacing each entry  $x$  in  $A$  with either  $\lfloor x \rfloor$  or  $\lceil x \rceil$ , without changing the sum of entries in any row or column of  $A$ . For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

Describe an efficient algorithm that either rounds  $A$  in this fashion, or reports correctly that no such rounding is possible.

- The Autocratic Party is gearing up their fund-raising campaign for the 2012 election. Party leaders have already chosen their slate of candidates for president and vice-president, as well as various governors, senators, representatives, city council members, school board members, and dog-catchers. For each candidate, the party leaders have determined how much money they must spend on that candidate's campaign to guarantee their election.

The party is soliciting donations from each of its members. Each voter has declared the total amount of money they are willing to give each candidate between now and the election. (Each voter pledges different amounts to different candidates. For example, everyone is happy to donate to the presidential candidate,<sup>1</sup> but most voters in New York will not donate anything to the candidate for Trash Commissioner of Los Angeles.) Federal election law limits each person's total political contributions to \$100 per day.

Describe and analyze an algorithm to compute a donation schedule, describing how much money each voter should send to each candidate on each day, that guarantees that every candidate gets enough money to win their election. (Party members will of course follow their given schedule perfectly.<sup>2</sup>) The schedule must obey both Federal laws and individual voters' budget constraints. If no such schedule exists, your algorithm should report that fact.

---

<sup>1</sup>or some nice men in suits will be visiting their home.

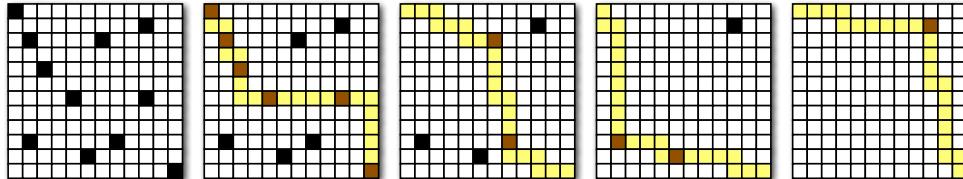
<sup>2</sup>It's a nice house you've got here. Shame if anything happened to it.

4. Consider an  $n \times n$  grid, some of whose cells are marked. A *monotone* path through the grid starts at the top-left cell, moves only right or down at each step, and ends at the bottom-right cell. We want to compute the minimum number of monotone paths that cover all the marked cells.

- (a) One of your friends suggests the following greedy strategy:

- Find (somehow) one “good” path  $\pi$  that covers the maximum number of marked cells.
- Unmark the cells covered by  $\pi$ .
- If any cells are still marked, recursively cover them.

Prove that this greedy strategy does *not* always compute an optimal solution.



Greedily covering the marked cells in a grid with four monotone paths.

- (b) Describe and analyze an efficient algorithm to compute the smallest set of monotone paths that covers every marked cell. The input to your algorithm is an array  $M[1..n, 1..n]$  of booleans, where  $M[i, j] = \text{TRUE}$  if and only if cell  $(i, j)$  is marked.
5. Let  $G$  be a directed graph with two distinguished vertices  $s$  and  $t$ , and let  $r$  be a positive integer. Two players named Paul and Sally play the following game. Paul chooses a path  $P$  from  $s$  to  $t$ , and Sally chooses a subset  $S$  of **at most**  $r$  edges in  $G$ . The players reveal their chosen subgraphs simultaneously. If  $P \cap S = \emptyset$ , Paul wins; if  $P \cap S \neq \emptyset$ , then Sally wins. Both players want to maximize their chances of winning the game.
- (a) Prove that if Paul uses a deterministic strategy, and Sally knows his strategy, then Sally can guarantee that she wins.<sup>3</sup>
  - (b) Let  $M$  be the number of edges in a minimum  $(s, t)$ -cut. Describe a deterministic strategy for Sally that guarantees that she wins when  $r \geq M$ , no matter what strategy Paul uses.
  - (c) Prove that if Sally uses a deterministic strategy, and Paul knows her strategy then Paul can guarantee that he wins when  $r < M$ .
  - (d) Describe a randomized strategy for Sally that guarantees that she wins with probability at least  $\min\{r/M, 1\}$ , no matter what strategy Paul uses.
  - (e) Describe a randomized strategy for Paul that guarantees that he loses with probability at most  $\min\{r/M, 1\}$ , no matter what strategy Sally uses.

Paul and Sally’s strategies are, of course, algorithms. (For example, Paul’s strategy is an algorithm that takes the graph  $G$  and the integer  $r$  as input and produces a path  $P$  as output.) You do *not* need to analyze the running times of these algorithms, but you must prove all claims about their winning probabilities. Most of these questions are easy.

---

<sup>3</sup>“Good old rock. Nothing beats rock. . . D’oh!”

# CS 573: Graduate Algorithms, Fall 2010

## Homework 5

Practice only — Do not submit solutions

---

1. (a) Describe how to transform any linear program written in general form into an equivalent linear program written in *slack* form.

$$\begin{aligned} & \text{maximize} \quad \sum_{j=1}^d c_j x_j \\ & \text{subject to} \quad \sum_{j=1}^d a_{ij} x_j \leq b_i \quad \text{for each } i = 1 \dots p \\ & \quad \sum_{j=1}^d a_{ij} x_j = b_i \quad \text{for each } i = p+1 \dots p+q \\ & \quad \sum_{j=1}^d a_{ij} x_j \geq b_i \quad \text{for each } i = p+q+1 \dots n \end{aligned}$$

$\implies$ 

$$\begin{aligned} & \max \quad c \cdot x \\ & \text{s.t. } Ax = b \\ & \quad x \geq 0 \end{aligned}$$

- (b) Describe precisely how to dualize a linear program written in slack form.  
(c) Describe precisely how to dualize a linear program written in general form.

In all cases, keep the number of variables in the resulting linear program as small as possible.

2. Suppose you have a subroutine that can solve linear programs in polynomial time, but only if they are both feasible and bounded. Describe an algorithm that solves *arbitrary* linear programs in polynomial time. Your algorithm should return an optimal solution if one exists; if no optimum exists, your algorithm should report that the input instance is UNBOUNDED or INFEASIBLE, whichever is appropriate. [Hint: Add one variable and one constraint.]
3. An *integer program* is a linear program with the additional constraint that the variables must take only integer values.
- (a) Prove that deciding whether an integer program has a feasible solution is NP-complete.  
(b) Prove that finding the optimal feasible solution to an integer program is NP-hard.
- [Hint: Almost any NP-hard decision problem can be formulated as an integer program. Pick your favorite.]
4. Give a linear-programming formulation of the *minimum-cost feasible circulation problem*. You are given a flow network whose edges have both capacities and costs, and your goal is to find a feasible circulation (flow with value 0) whose cost is as small as possible.

5. Given points  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$  in the plane, the *linear regression problem* asks for real numbers  $a$  and  $b$  such that the line  $y = ax + b$  fits the points as closely as possible, according to some criterion. The most common fit criterion is minimizing the  $L_2$  error, defined as follows:<sup>1</sup>

$$\varepsilon_2(a, b) = \sum_{i=1}^n (y_i - ax_i - b)^2.$$

But there are several other fit criteria, some of which can be optimized via linear programming.

- (a) The  $L_1$  error (or *total absolute deviation*) of the line  $y = ax + b$  is defined as follows:

$$\varepsilon_1(a, b) = \sum_{i=1}^n |y_i - ax_i - b|.$$

Describe a linear program whose solution  $(a, b)$  describes the line with minimum  $L_1$  error.

- (b) The  $L_\infty$  error (or *maximum absolute deviation*) of the line  $y = ax + b$  is defined as follows:

$$\varepsilon_\infty(a, b) = \max_{i=1}^n |y_i - ax_i - b|.$$

Describe a linear program whose solution  $(a, b)$  describes the line with minimum  $L_\infty$  error.

---

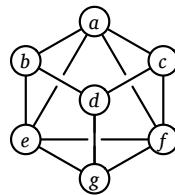
<sup>1</sup>This measure is also known as *sum of squared residuals*, and the algorithm to compute the best fit is normally called *(ordinary/linear) least squares fitting*.

This exam lasts 90 minutes.

**Write your answers in the separate answer booklet.**

Please return this question sheet with your answers.

1. (a) Suppose  $A[1..n]$  is an array of  $n$  distinct integers, sorted so that  $A[1] < A[2] < \dots < A[n]$ . Each integer  $A[i]$  could be positive, negative, or zero. Describe and analyze an efficient algorithm that either computes an index  $i$  such that  $A[i] = i$  or correctly reports that no such index exists.
- (b) Now suppose  $A[1..n]$  is a sorted array of  $n$  distinct **positive** integers. Describe and analyze an even faster algorithm that either computes an index  $i$  such that  $A[i] = i$  or correctly reports that no such index exists.
2. A *double-Hamiltonian circuit* a closed walk in a graph that visits every vertex exactly *twice*. *Prove* that it is NP-hard to determine whether a given graph contains a double-Hamiltonian circuit.



This graph contains the double-Hamiltonian circuit  $a \rightarrow b \rightarrow d \rightarrow g \rightarrow e \rightarrow b \rightarrow d \rightarrow c \rightarrow f \rightarrow a \rightarrow c \rightarrow f \rightarrow g \rightarrow e \rightarrow a$ .

3. A palindrome is any string that is exactly the same as its reversal, like I, or DEED, or HANNAH, or AMANAPLANACATACANALPANAMA. Describe and analyze an algorithm to find the length of the longest subsequence of a given string that is also a palindrome.

For example, the longest palindrome subsequence of MHDYNAMICPROGRAMZLETMSHOWYUTHEM is MHYMRORMYHM, so given that string as input, your algorithm should return the integer 11.

4. Suppose you are given a magic black box that can determine **in polynomial time**, given an arbitrary graph  $G$ , the number of vertices in the largest complete subgraph of  $G$ . Describe and analyze a **polynomial-time** algorithm that computes, given an arbitrary graph  $G$ , a complete subgraph of  $G$  of maximum size, using this magic black box as a subroutine.
5. Suppose we are given a  $4 \times n$  grid, where each grid cell has an integer value. Suppose we want to mark a subset of the grid cells, so that the total value of the marked cells is as large as possible. However, we are forbidden to mark any pair of grid cells that are immediate horizontal or vertical neighbors. (Marking diagonal neighbors is fine.) Describe and analyze an algorithm that computes the largest possible sum of marked cells, subject to this non-adjacency condition.

For example, given the grid on the left below, your algorithm should return the integer 36, which is the sum of the circled numbers on the right.

4	-5	1	6			
2	6	-1	8			
5	4	3	3			
1	-1	7	4			
-3	4	5	-2			

 $\Rightarrow$ 

(4)	-5	1	(6)			
2	(6)	-1	8			
(5)	4	3	(3)			
1	-1	(7)	4			
-3	(5)	4	-2			

**You may assume the following problems are NP-hard:**

**CIRCUITSAT:** Given a boolean circuit, are there any input values that make the circuit output True?

**PLANAR CIRCUITSAT:** Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output True?

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

**MAX2SAT:** Given a boolean formula in conjunctive normal form, with exactly two literals per clause, what is the largest number of clauses that can be satisfied by an assignment?

**MAXINDEPENDENTSET:** Given an undirected graph  $G$ , what is the size of the largest subset of vertices in  $G$  that have no edges among them?

**MAXCLIQUE:** Given an undirected graph  $G$ , what is the size of the largest complete subgraph of  $G$ ?

**MINVERTEXCOVER:** Given an undirected graph  $G$ , what is the size of the smallest subset of vertices that touch every edge in  $G$ ?

**MINDOMINATINGSET:** Given an undirected graph  $G$ , what is the size of the smallest subset  $S$  of vertices such that every vertex in  $G$  is either in  $S$  or adjacent to a vertex in  $S$ ?

**MINSETCOVER:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subcollection whose union is  $S$ ?

**MINHITTINGSET:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subset of  $S$  that intersects every subset  $S_i$ ?

**3COLOR:** Given an undirected graph  $G$ , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

**CHROMATICNUMBER:** Given an undirected graph  $G$ , what is the minimum number of colors needed to color its vertices, so that every edge touches vertices with two different colors?

**MAXCUT:** Given a graph  $G$ , what is the size (number of edges) of the largest bipartite subgraph of  $G$ ?

**HAMILTONIANCYCLE:** Given a graph  $G$ , is there a cycle in  $G$  that visits every vertex exactly once?

**HAMILTONIANPATH:** Given a graph  $G$ , is there a path in  $G$  that visits every vertex exactly once?

**TRAVELINGSALESMAN:** Given a graph  $G$  with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in  $G$ ?

**SUBSETSUM:** Given a set  $X$  of positive integers and an integer  $k$ , does  $X$  have a subset whose elements sum to  $k$ ?

**PARTITION:** Given a set  $X$  of positive integers, can  $X$  be partitioned into two subsets with the same sum?

**3PARTITION:** Given a set  $X$  of  $n$  positive integers, can  $X$  be partitioned into  $n/3$  three-element subsets, all with the same sum?

**MINESWEEPER:** Given a Minesweeper configuration and a particular square  $x$ , is it safe to click on  $x$ ?

**TETRIS:** Given a sequence of  $N$  Tetris pieces and a partially filled  $n \times k$  board, is it possible to play every piece in the sequence without overflowing the board?

**SUDOKU:** Given an  $n \times n$  Sudoku puzzle, does it have a solution?

**KENKEN:** Given an  $n \times n$  Ken-Ken puzzle, does it have a solution?

This exam lasts 90 minutes.  
**Write your answers in the separate answer booklet.**  
Please return this question sheet with your answers.

1. Assume we have access to a function  $\text{RANDOM}(k)$  that returns, given any positive integer  $k$ , an integer chosen independently and uniformly at random from the set  $\{1, 2, \dots, k\}$ , in  $O(1)$  time. For example, to perform a fair coin flip, we could call  $\text{RANDOM}(2)$ .

Now suppose we want to write an efficient function  $\text{RANDOMPERMUTATION}(n)$  that returns a permutation of the set  $\{1, 2, \dots, n\}$  chosen uniformly at random; that is, each permutation must be chosen with probability  $1/n!$ .

- (a) **Prove** that the following algorithm is *not* correct. [Hint: Consider the case  $n = 3$ .]

```
RANDOMPERMUTATION( $n$ ):
for  $i \leftarrow 1$  to  $n$ 
     $\pi[i] \leftarrow i$ 
for  $i \leftarrow 1$  to  $n$ 
    swap  $\pi[i] \leftrightarrow \pi[\text{RANDOM}(n)]$ 
return  $\pi$ 
```

- (b) Describe and analyze a correct  $\text{RANDOMPERMUTATION}$  algorithm that runs in  $O(n)$  expected time. (In fact,  $O(n)$  worst-case time is possible.)
2. Suppose we have  $n$  pieces of candy with weights  $W[1..n]$  (in ounces) that we want to load into boxes. Our goal is to load the candy into as many boxes as possible, so that each box contains at least  $L$  ounces of candy. Describe an efficient 2-approximation algorithm for this problem. **Prove** that the approximation ratio of your algorithm is 2.

(For 7 points partial credit, assume that every piece of candy weighs less than  $L$  ounces.)

3. The **MAXIMUM- $k$ -CUT** problem is defined as follows. We are given a graph  $G$  with weighted edges and an integer  $k$ . Our goal is to partition the vertices of  $G$  into  $k$  subsets  $S_1, S_2, \dots, S_k$ , so that the sum of the weights of the edges that cross the partition (that is, with endpoints in different subsets) is as large as possible.
  - (a) Describe an efficient randomized approximation algorithm for **MAXIMUM- $k$ -CUT**, and **prove** that its expected approximation ratio is **at most**  $(k - 1)/k$ .
  - (b) Now suppose we want to minimize the sum of the weights of edges that do *not* cross the partition. What expected approximation ratio does your algorithm from part (a) achieve for this new problem? **Prove** your answer is correct.

4. The citizens of Binaria use coins whose values are powers of two. That is, for any non-negative integer  $k$ , there are Binarian coins with value is  $2^k$  bits. Consider the natural greedy algorithm to make  $x$  bits in change: If  $x > 0$ , use one coin with the largest denomination  $d \leq x$  and then recursively make  $x - d$  bits in change. (Assume you have an unlimited supply of each denomination.)

- (a) **Prove** that this algorithm uses at most one coin of each denomination.
- (b) **Prove** that this algorithm finds the minimum number of coins whose total value is  $x$ .

5. Any permutation  $\pi$  can be represented as a set of disjoint cycles, by considering the directed graph whose vertices are the integers between 1 and  $n$  and whose edges are  $i \rightarrow \pi(i)$  for each  $i$ . For example, the permutation  $\langle 5, 4, 2, 6, 7, 8, 1, 3, 9 \rangle$  has three cycles:  $(175)(24683)(9)$ .

In the following questions, let  $\pi$  be a permutation of  $\{1, 2, \dots, n\}$  chosen uniformly at random, and let  $k$  be an arbitrary integer such that  $1 \leq k \leq n$ .

- (a) **Prove** that the probability that the number 1 lies in a cycle of length  $k$  in  $\pi$  is precisely  $1/n$ .  
*[Hint: Consider the cases  $k = 1$  and  $k = 2$ .]*
- (b) What is the *exact* expected length of the cycle in  $\pi$  that contains the number 1?
- (c) What is the *exact* expected number of cycles of length  $k$  in  $\pi$ ?
- (d) What is the *exact* expected number of cycles in  $\pi$ ?

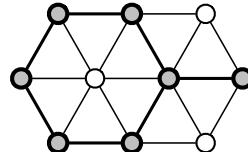
You may assume part (a) in your solutions to parts (b), (c), and (d).

This exam lasts 180 minutes.

**Write your answers in the separate answer booklet.**

Please return this question sheet with your answers.

1. A subset  $S$  of vertices in an undirected graph  $G$  is called *triangle-free* if, for every triple of vertices  $u, v, w \in S$ , at least one of the three edges  $uv, uw, vw$  is *absent* from  $G$ . **Prove** that finding the size of the largest triangle-free subset of vertices in a given undirected graph is NP-hard.

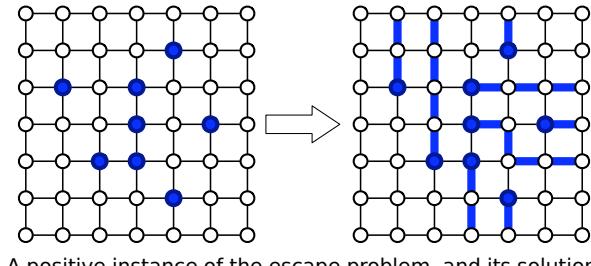


A triangle-free subset of 7 vertices.

This is **not** the largest triangle-free subset in this graph.

2. An  $n \times n$  grid is an undirected graph with  $n^2$  vertices organized into  $n$  rows and  $n$  columns. We denote the vertex in the  $i$ th row and the  $j$ th column by  $(i, j)$ . Every vertex in the grid have exactly four neighbors, except for the *boundary* vertices, which are the vertices  $(i, j)$  such that  $i = 1, i = n, j = 1$ , or  $j = n$ .

Let  $(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$  be distinct vertices, called *terminals*, in the  $n \times n$  grid. The **escape problem** is to determine whether there are  $m$  vertex-disjoint paths in the grid that connect the terminals to any  $m$  distinct boundary vertices. Describe and analyze an efficient algorithm to solve the escape problem.



A positive instance of the escape problem, and its solution.

3. Consider the following problem, called **UNIQUESETCOVER**. The input is an  $n$ -element set  $S$ , together with a collection of  $m$  subsets  $S_1, S_2, \dots, S_m \subseteq S$ , such that each element of  $S$  lies in exactly  $k$  subsets  $S_i$ . Our goal is to select some of the subsets so as to maximize the number of elements of  $S$  that lie in *exactly one* selected subset.

- (a) Fix a real number  $p$  between 0 and 1, and consider the following algorithm:

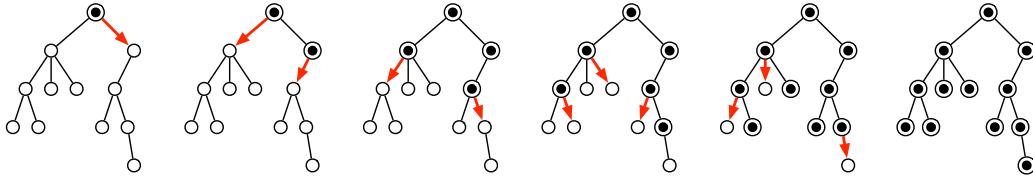
For each index  $i$ , select subset  $S_i$  independently with probability  $p$ .

What is the *exact* expected number of elements that are uniquely covered by the chosen subsets? (Express your answer as a function of the parameters  $p$  and  $k$ .)

- (b) What value of  $p$  maximizes this expectation?

- (c) Describe a polynomial-time randomized algorithm for **UNIQUESETCOVER** whose expected approximation ratio is  $O(1)$ .

4. Suppose we need to distribute a message to all the nodes in a rooted tree. Initially, only the root node knows the message. In a single round, any node that knows the message can forward it to at most one of its children. Describe and analyze an efficient algorithm to compute the minimum number of rounds required for the message to be delivered to every node.



A message being distributed through a tree in five rounds.

5. Every year, Professor Dumbledore assigns the instructors at Hogwarts to various faculty committees. There are  $n$  faculty members and  $c$  committees. Each committee member has submitted a list of their *prices* for serving on each committee; each price could be positive, negative, zero, or even infinite. For example, Professor Snape might declare that he would serve on the Student Recruiting Committee for 1000 Galleons, that he would *pay* 10000 Galleons to serve on the Defense Against the Dark Arts Course Revision Committee, and that he would not serve on the Muggle Relations committee for any price.

Conversely, Dumbledore knows how many instructors are needed for each committee, as well as a list of instructors who would be suitable members for each committee. (For example: “Dark Arts Revision: 5 members, anyone but Snape.”) If Dumbledore assigns an instructor to a committee, he must pay that instructor’s price from the Hogwarts treasury.

Dumbledore needs to assign instructors to committees so that (1) each committee is full, (3) no instructor is assigned to more than three committees, (2) only suitable and willing instructors are assigned to each committee, and (4) the total cost of the assignment is as small as possible. Describe and analyze an efficient algorithm that either solves Dumbledore’s problem, or correctly reports that there is no valid assignment whose total cost is finite.

6. Suppose we are given a rooted tree  $T$ , where every edge  $e$  has a non-negative length  $\ell(e)$ . Describe and analyze an efficient algorithm to assign a *stretched* length  $s\ell(e) \geq \ell(e)$  to every edge  $e$ , satisfying the following conditions:

- Every root-to-leaf path in  $T$  has the same total stretched length.
- The total stretch  $\sum_e (s\ell(e) - \ell(e))$  is as small as possible.

7. Let  $G = (V, E)$  be a directed graph with edge capacities  $c: E \rightarrow \mathbb{R}^+$ , a source vertex  $s$ , and a target vertex  $t$ . Suppose someone hands you an *arbitrary* function  $f: E \rightarrow \mathbb{R}$ . Describe and analyze fast and *simple* algorithms to answer the following questions:

- Is  $f$  a feasible  $(s, t)$ -flow in  $G$ ?
- Is  $f$  a *maximum*  $(s, t)$ -flow in  $G$ ?
- Is  $f$  the *unique* maximum  $(s, t)$ -flow in  $G$ ?

**Chernoff bounds:**

If  $X$  is the sum of independent indicator variables and  $\mu = E[X]$ , then

$$\Pr[X > (1 + \delta)\mu] \leq \left( \frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^\mu \quad \text{for any } \delta > 0$$

$$\Pr[X > (1 - \delta)\mu] \leq \left( \frac{e^{-\delta}}{(1 - \delta)^{1-\delta}} \right)^\mu \quad \text{for any } 0 < \delta < 1$$

**You may assume the following running times:**

- Maximum flow or minimum cut:  $O(E|f^*|)$  or  $O(VE \log V)$
- Minimum-cost maximum flow:  $O(E^2 \log^2 V)$

(These are *not* the best time bounds known, but they're close enough for the final exam.)

**You may assume the following problems are NP-hard:**

**CIRCUITSAT:** Given a boolean circuit, are there any input values that make the circuit output True?

**PLANAR CIRCUITSAT:** Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output True?

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

**MAX2SAT:** Given a boolean formula in conjunctive normal form, with exactly two literals per clause, what is the largest number of clauses that can be satisfied by an assignment?

**MAXINDEPENDENTSET:** Given an undirected graph  $G$ , what is the size of the largest subset of vertices in  $G$  that have no edges among them?

**MAXCLIQUE:** Given an undirected graph  $G$ , what is the size of the largest complete subgraph of  $G$ ?

**MINVERTEXCOVER:** Given an undirected graph  $G$ , what is the size of the smallest subset of vertices that touch every edge in  $G$ ?

**MINSETCOVER:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subcollection whose union is  $S$ ?

**MINHITTINGSET:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subset of  $S$  that intersects every subset  $S_i$ ?

**3COLOR:** Given an undirected graph  $G$ , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

**MAXCUT:** Given a graph  $G$ , what is the size (number of edges) of the largest bipartite subgraph of  $G$ ?

**HAMILTONIANCYCLE:** Given a graph  $G$ , is there a cycle in  $G$  that visits every vertex exactly once?

**HAMILTONIANPATH:** Given a graph  $G$ , is there a path in  $G$  that visits every vertex exactly once?

**TRAVELINGSALESMAN:** Given a graph  $G$  with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in  $G$ ?

**SUBSETSUM:** Given a set  $X$  of positive integers and an integer  $k$ , does  $X$  have a subset whose elements sum to  $k$ ?

**PARTITION:** Given a set  $X$  of positive integers, can  $X$  be partitioned into two subsets with the same sum?

**3PARTITION:** Given a set  $X$  of  $n$  positive integers, can  $X$  be partitioned into  $n/3$  three-element subsets, all with the same sum?

# CS 473: Undergraduate Algorithms, Fall 2013

## Homework 0

Due Tuesday, September 3, 2013 at 12:30pm

---

Quiz 0 (on the course Moodle page)  
is also due Tuesday, September 3, 2013 at noon.

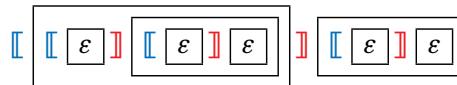
---

- Please carefully read the course policies on the course web site. These policies may be different than other classes you have taken. (For example: No late anything ever; “I don’t know” is worth 25%, but “Repeat this for all  $n$ ” is an automatic zero; *every* homework question requires a proof; collaboration is allowed, but you must cite your collaborators.) If you have *any* questions, please ask in lecture, in headbanging, on Piazza, in office hours, or by email.
  - Homework 0 and Quiz 0 test your familiarity with prerequisite material—big-Oh notation, elementary algorithms and data structures, recurrences, graphs, and most importantly, induction—to help you identify gaps in your background knowledge. You are responsible for filling those gaps. The course web page has pointers to several excellent online resources for prerequisite material. If you need help, please ask in headbanging, on Piazza, in office hours, or by email.
  - Each student must submit individual solutions for these homework problems. You may use any source at your disposal—paper, electronic, or human—but you **must** cite *every* source that you use. For all future homeworks, groups of up to three students may submit joint solutions.
  - Submit your solutions on standard printer/copier paper, not notebook paper. If you write your solutions by hand, please use the last three pages of this homework as a template. At the top of each page, please clearly print your name and NetID, and indicate your registered discussion section. Use both sides of the page. If you plan to typeset your homework, you can find a  $\text{\LaTeX}$  template on the course web site; well-typeset homework will get a small amount of extra credit.
  - Submit your solution to each numbered problem (stapled if necessary) in the corresponding drop box outside 1404 Siebel, **or** in the corresponding box in Siebel 1404 immediately before/after class. (This is the last homework we’ll collect in 1404.) ***Do not staple your entire homework together.***
-

1. Consider the following recursively-defined sets of strings of left brackets  $\llbracket$  and right brackets  $\rrbracket$ :

- A string  $x$  is **balanced** if it satisfies one of the following conditions:
  - $x$  is the empty string, or
  - $x = \llbracket y \rrbracket z$ , where  $y$  and  $z$  are balanced strings.

For example, the following diagram shows that the string  $\llbracket \llbracket \rrbracket \llbracket \rrbracket \llbracket \rrbracket \llbracket \rrbracket$  is balanced. Each boxed substring is balanced, and  $\varepsilon$  is the empty string.



- A string  $x$  is **erasable** if it satisfies one of two conditions:
  - $x$  is the empty string, or
  - $x = y \llbracket \rrbracket z$ , where  $yz$  is an erasable string.

For example, we can prove that the string  $\llbracket \llbracket \rrbracket \llbracket \rrbracket \llbracket \rrbracket \llbracket \rrbracket$  is erasable as follows:

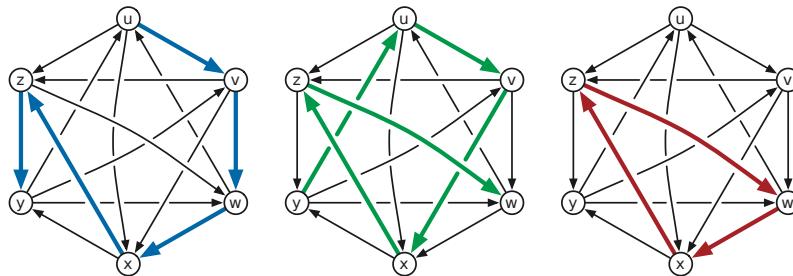
$$\llbracket \llbracket \rrbracket \llbracket \rrbracket \llbracket \rrbracket \llbracket \rrbracket \rightarrow \llbracket \llbracket \rrbracket \llbracket \rrbracket \llbracket \rrbracket \rightarrow \llbracket \rrbracket \llbracket \rrbracket \rightarrow \llbracket \rrbracket \rightarrow \varepsilon$$

Your task is to prove that these two definitions are equivalent.

- (a) Prove that every balanced string is erasable.
- (b) Prove that every erasable string is balanced.

2. A **tournament** is a directed graph with exactly one directed edge between each pair of vertices. That is, for any vertices  $v$  and  $w$ , a tournament contains either an edge  $v \rightarrow w$  or an edge  $w \rightarrow v$ , but not both. A **Hamiltonian path** in a directed graph  $G$  is a directed path that visits every vertex of  $G$  exactly once.

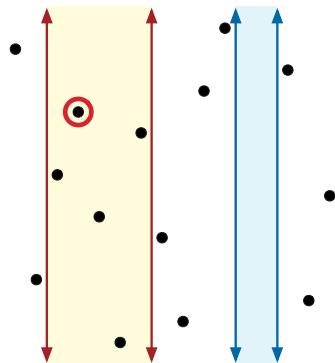
- (a) Prove that every tournament contains a Hamiltonian path.
- (b) Prove that every tournament contains either *exactly one* Hamiltonian path or a directed cycle of length three.



A tournament with two Hamiltonian paths  $u \rightarrow v \rightarrow w \rightarrow x \rightarrow z \rightarrow y$  and  $y \rightarrow u \rightarrow v \rightarrow x \rightarrow z \rightarrow w$  and a directed triangle  $w \rightarrow x \rightarrow z \rightarrow w$ .

3. Suppose you are given a set  $P = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$  of  $n$  points in the plane with distinct  $x$ - and  $y$ -coordinates. Describe a data structure that can answer the following query as quickly as possible:

Given two numbers  $l$  and  $r$ , find the highest point in  $P$  inside the vertical slab  $l < x < r$ . More formally, find the point  $(x_i, y_i) \in P$  such that  $l < x_i < r$  and  $y_i$  is as large as possible. Return `NONE` if the slab does not contain any points in  $P$ .



A query with the left slab returns the indicated point.

A query with the right slab returns `NONE`.

To receive full credit, your solution must include (a) a concise description of your data structure, (b) a concise description of your query algorithm, (c) a proof that your query algorithm is correct, (d) a bound on the size of your data structure, and (e) a bound on the running time of your query algorithm. You do **not** need to describe or analyze an algorithm to construct your data structure.

Smaller data structures and faster query times are worth more points.

Starting with this homework, groups of up to three students may submit a single solution for each numbered problem. Every student in the group receives the same grade.  
 Groups can be different for different problems.

1. Consider the following cruel and unusual sorting algorithm.

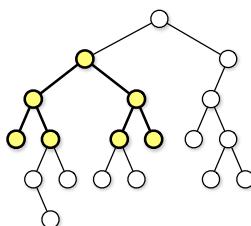
```

CRUEL( $A[1..n]$ ):
  if  $n > 1$ 
    CRUEL( $A[1..n/2]$ )
    CRUEL( $A[n/2+1..n]$ )
    UNUSUAL( $A[1..n]$ )

UNUSUAL( $A[1..n]$ ):
  if  $n = 2$ 
    if  $A[1] > A[2]$                                 «the only comparison!»
      swap  $A[1] \leftrightarrow A[2]$ 
    else
      for  $i \leftarrow 1$  to  $n/4$                   «swap 2nd and 3rd quarters»
        swap  $A[i+n/4] \leftrightarrow A[i+n/2]$ 
      UNUSUAL( $A[1..n/2]$ )                      «recurse on left half»
      UNUSUAL( $A[n/2+1..n]$ )                    «recurse on right half»
      UNUSUAL( $A[n/4+1..3n/4]$ )                 «recurse on middle half»
```

Notice that the comparisons performed by the algorithm do not depend at all on the values in the input array; such a sorting algorithm is called *oblivious*. Assume for this problem that the input size  $n$  is always a power of 2.

- (a) Prove that CRUEL correctly sorts any input array. [Hint: Consider an array that contains  $n/4$  1s,  $n/4$  2s,  $n/4$  3s, and  $n/4$  4s. Why is considering this special case enough? What does UNUSUAL actually do?]
  - (b) Prove that CRUEL would *not* always sort correctly if we removed the for-loop from UNUSUAL.
  - (c) Prove that CRUEL would *not* always sort correctly if we swapped the last two lines of UNUSUAL.
  - (d) What is the running time of UNUSUAL? Justify your answer.
  - (e) What is the running time of CRUEL? Justify your answer.
2. For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth. Describe and analyze a recursive algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return the root and the depth of this subtree.



The largest complete subtree of this binary tree has depth 2.

3. (a) Suppose we are given two sorted arrays  $A[1..n]$  and  $B[1..n]$ . Describe an algorithm to find the median of the union of  $A$  and  $B$  in  $O(\log n)$  time. Assume the arrays contain no duplicate elements.
- (b) Now suppose we are given *three* sorted arrays  $A[1..n]$ ,  $B[1..n]$ , and  $C[1..n]$ . Describe an algorithm to find the median element of  $A \cup B \cup C$  in  $O(\log n)$  time.

- \*4. **Extra credit; due September 17.** (The “I don’t know” rule does not apply to extra credit problems.)

Bob Ratenbur, a new student in CS 225, is trying to write code to perform preorder, inorder, and postorder traversal of binary trees. Bob understands the basic idea behind the traversal algorithms, but whenever he tries to implement them, he keeps mixing up the recursive calls. Five minutes before the deadline, Bob submitted code with the following structure:

<u>PREORDER(<math>v</math>):</u>	<u>INORDER(<math>v</math>):</u>	<u>POSTORDER(<math>v</math>):</u>
<pre> if <math>v = \text{NULL}</math>     return else     print <i>label</i>(<math>v</math>)     ■■■ ORDER(left(<math>v</math>))     ■■■ ORDER(right(<math>v</math>)) </pre>	<pre> if <math>v = \text{NULL}</math>     return else     ■■■ ORDER(left(<math>v</math>))     print <i>label</i>(<math>v</math>)     ■■■ ORDER(right(<math>v</math>)) </pre>	<pre> if <math>v = \text{NULL}</math>     return else     ■■■ ORDER(left(<math>v</math>))     ■■■ ORDER(right(<math>v</math>))     print <i>label</i>(<math>v</math>) </pre>

Each ■■■ represents either PRE, IN, or POST. Moreover, each of the following function calls appears exactly once in Bob’s submitted code:

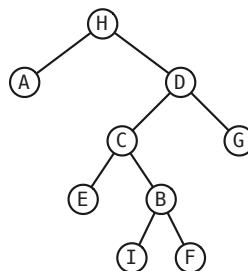
$$\begin{array}{lll}
\text{PREORDER}(\text{left}(v)) & \text{INORDER}(\text{left}(v)) & \text{POSTORDER}(\text{left}(v)) \\
\text{PREORDER}(\text{right}(v)) & \text{INORDER}(\text{right}(v)) & \text{POSTORDER}(\text{right}(v))
\end{array}$$

Thus, there are exactly 36 possibilities for Bob’s code. Unfortunately, Bob accidentally deleted his source code after submitting the executable, so neither you nor he knows which functions were called where.

Your task is to reconstruct a binary tree  $T$  from the output of Bob’s traversal algorithms, which has been helpfully parsed into three arrays  $\text{Pre}[1..n]$ ,  $\text{In}[1..n]$ , and  $\text{Post}[1..n]$ . Your algorithm should return the unknown tree  $T$ . You may assume that the vertex labels of the unknown tree are distinct, and that every internal node has exactly two children. For example, given the input

$$\begin{aligned}
\text{Pre}[1..n] &= [\text{H A E C B I F G D}] \\
\text{In}[1..n] &= [\text{A H D C E I F B G}] \\
\text{Post}[1..n] &= [\text{A E I B F C D G H}]
\end{aligned}$$

your algorithm should return the following tree:



In general, the traversal sequences may not give you enough information to reconstruct Bob's code; however, to produce the example sequences above, Bob's code must look like this:

PREORDER( $v$ ):

```
if  $v$  = NULL  
    return  
else  
    print label( $v$ )  
    PREORDER(left( $v$ ))  
    POSTORDER(right( $v$ ))
```

INORDER( $v$ ):

```
if  $v$  = NULL  
    return  
else  
    POSTORDER(left( $v$ ))  
    print label( $v$ )  
    PREORDER(right( $v$ ))
```

POSTORDER( $v$ ):

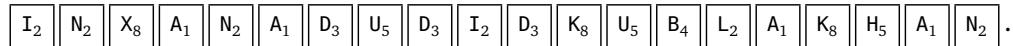
```
if  $v$  = NULL  
    return  
else  
    INORDER(left( $v$ ))  
    INORDER(right( $v$ ))  
    print label( $v$ )
```

1. Suppose we are given an array  $A[1..n]$  of integers, some positive and negative, which we are asked to partition into contiguous subarrays, which we call ***chunks***. The *value* of any chunk is the *square* of the sum of elements in that chunk; the value of a partition of  $A$  is the sum of the values of its chunks.

For example, suppose  $A = [3, -1, 4, -1, 5, -9]$ . The partition  $[3, -1, 4], [-1, 5], [-9]$  has three chunks with total value  $(3 - 1 + 4)^2 + (-1 + 5)^2 + (-9)^2 = 6^2 + 4^2 + 9^2 = 133$ , while the partition  $[3, -1], [4, -1, 5, -9]$  has two chunks with total value  $(3 - 1)^2 + (4 - 1 + 5 - 9)^2 = 5$ .

- (a) Describe and analyze an algorithm that computes the minimum-value partition of a given array of  $n$  numbers.
  - (b) Now suppose we are given an integer  $k > 0$ . Describe and analyze an algorithm that computes the minimum-value partition of a given array of  $n$  numbers *into at most  $k$  chunks*.
2. Consider the following solitaire form of Scrabble. We begin with a fixed, finite sequence of tiles; each tile contains a letter and a numerical value. At the start of the game, we draw the seven tiles from the sequence and put them into our hand. In each turn, we form an English word from some or all of the tiles in our hand, place those tiles on the table, and receive the total value of those tiles as points. If no English word can be formed from the tiles in our hand, the game immediately ends. Then we repeatedly draw the next tile from the start of the sequence until either (a) we have seven tiles in our hand, or (b) the sequence is empty. (Sorry, no double/triple word/letter scores, bingos, blanks, or passing.) Our goal is to obtain as many points as possible.

For example, suppose we are given the tile sequence



Then we can earn 68 points as follows:

- We initially draw  $I_2, N_2, X_8, A_1, N_2, A_1, D_3$ .
- Play the word  $N_2, A_1, I_2, A_1, D_3$  for 9 points, leaving  $N_2, X_8$  in our hand.
- Draw the next five tiles  $U_5, D_3, I_2, D_3, K_8$ .
- Play the word  $U_5, N_2, D_3, I_2, D_3$  for 15 points, leaving  $K_8, X_8$  in our hand.
- Draw the next five tiles  $U_5, B_4, L_2, A_1, K_8$ .
- Play the word  $B_4, U_5, L_2, K_8$  for 19 points, leaving  $K_8, X_8, A_1$  in our hand.
- Draw the next three tiles  $H_5, A_1, N_2$ , emptying the list.
- Play the word  $A_1, N_2, K_8, H_5$  for 16 points, leaving  $X_8, A_1$  in our hand.
- Play the word  $A_1, X_8$  for 9 points, emptying our hand and ending the game.

Design and analyze an algorithm to compute the maximum number of points that can be earned from a given sequence of tiles. *The input consists of two arrays Letter[1..n], containing a sequence of letters between A and Z, and Value[A..Z], where Value[i] is the value of letter i.* The output is a single number. Assume that you can find all English words that can be made from any seven tiles, along with the point values of those words, in  $O(1)$  time.

3. **Extra credit.** Submit your answer to Homework 1 problem 4.

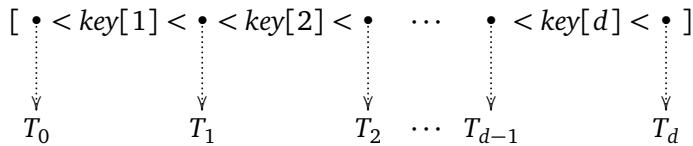
1. A standard method to improve the cache performance of search trees is to pack more search keys and subtrees into each node. A ***B-tree*** is a rooted tree in which each internal node stores up to  $B$  keys and pointers to up to  $B + 1$  children, each the root of a smaller  $B$ -tree. Specifically each node  $v$  stores three fields:

- a positive integer  $v.d \leq B$ ,
- a *sorted* array  $v.key[1..v.d]$ , and
- an array  $v.child[0..v.d]$  of child pointers.

In particular, the number of child pointers is always exactly one more than the number of keys.

Each pointer  $v.child[i]$  is either NULL or a pointer to the root of a  $B$ -tree whose keys are all larger than  $v.key[i]$  and smaller than  $v.key[i + 1]$ . In particular, all keys in the leftmost subtree  $v.child[0]$  are smaller than  $v.key[1]$ , and all keys in the rightmost subtree  $v.child[v.d]$  are larger than  $v.key[v.d]$ .

Intuitively, you should have the following picture in mind:



Here  $T_i$  is the subtree pointed to by  $child[i]$ .

The **cost** of searching for a key  $x$  in a  $B$ -tree is the number of nodes in the path from the root to the node containing  $x$  as one of its keys. A 1-tree is just a standard binary search tree.

Fix an arbitrary positive integer  $B > 0$ . (I suggest  $B = 8$ .) Suppose we are given a sorted array  $A[1, \dots, n]$  of search keys and a corresponding array  $F[1, \dots, n]$  of frequency counts, where  $F[i]$  is the number of times that we will search for  $A[i]$ .

Describe and analyze an efficient algorithm to find a  $B$ -tree that minimizes the total cost of searching for  $n$  keys with a given array of frequencies.

- For 5 points, describe a polynomial-time algorithm for the special case  $B = 2$ .
- For 10 points, describe an algorithm for arbitrary  $B$  that runs in  $O(n^{B+c})$  time for some fixed integer  $c$ .
- For 15 points, describe an algorithm for arbitrary  $B$  that runs in  $O(n^c)$  time for some fixed integer  $c$  that does *not* depend on  $B$ .

Like all other homework problems, 10 points is full credit; any points above 10 will be awarded as extra credit.

**A few comments about  $B$ -trees.** Normally,  $B$ -trees are required to satisfy two additional constraints, which guarantee a worst-case search cost of  $O(\log_B n)$ : Every leaf must have exactly the same depth, and every node except possibly the root must contain at least  $B/2$  keys. However, in this problem, we are not interested in optimizing the *worst-case* search cost, but rather the *total* cost of a sequence of searches, so we will not impose these additional constraints.

In most large database systems, the parameter  $B$  is chosen so that each node exactly fits in a cache line. Since the entire cache line is loaded into cache anyway, and the cost of loading a cache

line exceeds the cost of searching within the cache, the running time is dominated by the number of cache faults. This effect is even more noticeable if the data is too big to lie in RAM at all; then the cost is dominated by the number of page faults, and  $B$  should be roughly the size of a page.

Finally, don't worry about the cache/disk performance in your homework solutions; just analyze the CPU time as usual. Designing algorithms with few cache misses or page faults is a interesting pastime; simultaneously optimizing CPU time *and* cache misses *and* page faults is even more interesting. But this kind of design and analysis requires tools we won't see in this class.

2. ***Extra credit***, because we screwed up the first version.

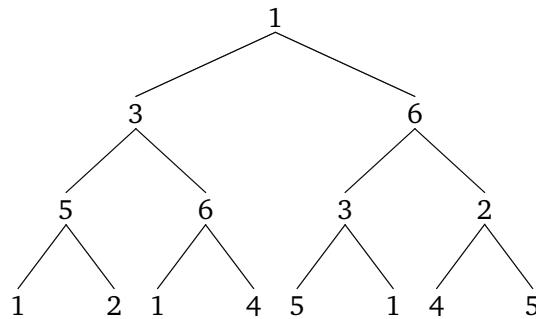
A ***k-2 coloring*** of a tree assigns each vertex a value from the set  $\{1, 2, \dots, k\}$ , called its *color*, such that the following constraints are satisfied:

- A node and its parent cannot have the same color or adjacent colors.
- A node and its grandparent cannot have the same color.
- Two nodes with the same parent cannot have the same color.

The last two rules can be written more simply as “Two nodes that are two edges apart cannot have the same color.” Diagrammatically, if we write the names of the colors inside the vertices,

$$\begin{array}{c} \textcircled{i} - \textcircled{j} \implies |i - j| \geq 2 \\ \textcircled{i} - \textcircled{\phantom{j}} - \textcircled{j} \implies i \neq j \end{array}$$

For example, here is a valid 6-2 coloring of the complete binary tree with depth 3:



- Describe and analyze an algorithm that computes a 6-2 coloring of a given binary tree. The existence of such an algorithm proves that every binary tree has a 6-2 coloring.
- Prove that not every binary tree has a 5-2 coloring.
- A *ternary tree* is a rooted tree where every node has at most *three* children. What is the smallest integer  $k$  such that every ternary tree has a  $k$ -2 coloring? Prove your answer is correct.

1. A *meldable priority queue* stores a set of values, called *priorities*, from some totally-ordered universe (such as the integers) and supports the following operations:

- **MAKEQUEUE**: Return a new priority queue containing the empty set.
- **FINDMIN( $Q$ )**: Return the smallest element of  $Q$  (if any).
- **DELETEMIN( $Q$ )**: Remove the smallest element in  $Q$  (if any).
- **INSERT( $Q, x$ )**: Insert priority  $x$  into  $Q$ , if it is not already there.
- **DECREASE( $Q, x, y$ )**: Replace some element  $x \in Q$  with a smaller priority  $y$ . (If  $y > x$ , the operation fails.) The input is a pointer directly to the node in  $Q$  containing  $x$ .
- **DELETE( $Q, x$ )**: Delete the priority  $x \in Q$ . The input is a pointer directly to the node in  $Q$  containing  $x$ .
- **MELD( $Q_1, Q_2$ )**: Return a new priority queue containing all the elements of  $Q_1$  and  $Q_2$ ; this operation destroys  $Q_1$  and  $Q_2$ .

A simple way to implement such a data structure is to use a *heap-ordered binary tree* — each node stores a priority, which is smaller than the priorities of its children, along with pointers to its parent and at most two children. MELD can be implemented using the following randomized algorithm:

```
MELD( $Q_1, Q_2$ ):
    if  $Q_1$  is empty return  $Q_2$ 
    if  $Q_2$  is empty return  $Q_1$ 
    if  $\text{priority}(Q_1) > \text{priority}(Q_2)$ 
        swap  $Q_1 \leftrightarrow Q_2$ 
    with probability 1/2
         $\text{left}(Q_1) \leftarrow \text{MELD}(\text{left}(Q_1), Q_2)$ 
    else
         $\text{right}(Q_1) \leftarrow \text{MELD}(\text{right}(Q_1), Q_2)$ 
    return  $Q_1$ 
```

- (a) Prove that for *any* heap-ordered binary trees  $Q_1$  and  $Q_2$  (**not** just those constructed by the operations listed above), the expected running time of  $\text{MELD}(Q_1, Q_2)$  is  $O(\log n)$ , where  $n = |Q_1| + |Q_2|$ . [Hint: How long is a random root-to-leaf path in an  $n$ -node binary tree if each left/right choice is made uniformly and independently at random?]
- (b) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and  $O(1)$  additional time. (This implies that every operation takes  $O(\log n)$  expected time.)
2. Recall that a *priority search tree* is a binary tree in which every node has both a *search key* and a *priority*, arranged so that the tree is simultaneously a binary search tree for the keys and a min-heap for the priorities. A *treap* is a priority search tree whose search keys are given by the user and whose priorities are independent random numbers.

A *heater* is a priority search tree whose *priorities* are given by the user and whose *search keys* are distributed uniformly and independently at random in the real interval  $[0, 1]$ . Intuitively, a heater is a sort of anti-treap.<sup>1</sup>

---

<sup>1</sup>There are those who think that life has nothing left to chance, a host of holy horrors to direct our aimless dance.

The following problems consider an  $n$ -node heater  $T$ . We identify nodes in  $T$  by their *priority rank*; for example, “node 5” means the node in  $T$  with the 5th smallest priority. The min-heap property implies that node 1 is the root of  $T$ . You may assume all search keys and priorities are distinct. Finally, let  $i$  and  $j$  be arbitrary integers with  $1 \leq i < j \leq n$ .

- (a) Prove that if we permute the set  $\{1, 2, \dots, n\}$  uniformly at random, integers  $i$  and  $j$  are adjacent with probability  $2/n$ .
- (b) Prove that node  $i$  is an ancestor of node  $j$  with probability  $2/(i+1)$ . [Hint: Use part (a)!]
- (c) What is the probability that node  $i$  is a descendant of node  $j$ ? [Hint: Don’t use part (a)!]
- (d) What is the exact expected depth of node  $j$ ?
- (e) Describe and analyze an algorithm to insert a new item into an  $n$ -node heater.
- (f) Describe and analyze an algorithm to delete the smallest priority (the root) from an  $n$ -node heater.

- \*3. **Extra credit; due October 15.** In the usual theoretical presentation of treaps, the priorities are random real numbers chosen uniformly from the interval  $[0, 1]$ . In practice, however, computers have access only to random bits. This problem asks you to analyze an implementation of treaps that takes this limitation into account.

Suppose the priority of a node  $v$  is abstractly represented as an infinite sequence  $\pi_v[1..∞]$  of random bits, which is interpreted as the rational number

$$\text{priority}(v) = \sum_{i=1}^{\infty} \pi_v[i] \cdot 2^{-i}.$$

However, only a finite number  $ℓ_v$  of these bits are actually known at any given time. When a node  $v$  is first created, *none* of the priority bits are known:  $ℓ_v = 0$ . We generate (or “reveal”) new random bits only when they are necessary to compare priorities. The following algorithm compares the priorities of any two nodes in  $O(1)$  expected time:

```

LARGERPRIORITY( $v, w$ ):
  for  $i \leftarrow 1$  to  $\infty$ 
    if  $i > ℓ_v$ 
       $ℓ_v \leftarrow i$ ;  $\pi_v[i] \leftarrow \text{RANDOMBIT}$ 
    if  $i > ℓ_w$ 
       $ℓ_w \leftarrow i$ ;  $\pi_w[i] \leftarrow \text{RANDOMBIT}$ 
    if  $\pi_v[i] > \pi_w[i]$ 
      return  $v$ 
    else if  $\pi_v[i] < \pi_w[i]$ 
      return  $w$ 
  
```

Suppose we insert  $n$  items one at a time into an initially empty treap. Let  $L = \sum_v ℓ_v$  denote the total number of random bits generated by calls to LARGERPRIORITY during these insertions.

- (a) Prove that  $E[L] = \Theta(n)$ .
- (b) Prove that  $E[\ell_v] = \Theta(1)$  for any node  $v$ . [Hint: This is equivalent to part (a). Why?]
- (c) Prove that  $E[\ell_{\text{root}}] = \Theta(\log n)$ . [Hint: Why doesn’t this contradict part (b)?]

1. Recall that a standard (FIFO) queue maintains a sequence of items subject to the following operations.

- $\text{PUSH}(x)$ : Add item  $x$  to the end of the sequence.
- $\text{PULL}()$ : Remove and return the item at the beginning of the sequence.
- $\text{SIZE}()$ : Return the current number of items in the sequence.

It is easy to implement a queue using a doubly-linked list, so that it uses  $O(n)$  space (where  $n$  is the number of items in the queue) and the worst-case time for each of these operations is  $O(1)$ .

Consider the following new operation, which removes every tenth element from the queue, starting at the beginning, in  $\Theta(n)$  worst-case time.

```
DECIMATE():
  n ← SIZE()
  for i ← 0 to n – 1
    if i mod 10 = 0
      PULL()  «result discarded»
    else
      PUSH(PULL())
```

Prove that in any intermixed sequence of  $\text{PUSH}$ ,  $\text{PULL}$ , and  $\text{DECIMATE}$  operations, the amortized cost of each operation is  $O(1)$ .

2. *This problem is extra credit*, because the original problem statement had several confusing small errors. I believe these errors are corrected in the current revision.

Deleting an item from an open-addressed hash table is not as straightforward as deleting from a chained hash table. The obvious method for deleting an item  $x$  simply empties the entry in the hash table that contains  $x$ . Unfortunately, the obvious method doesn't always work. (Part (a) of this question asks you to prove this.)

Knuth proposed the following *lazy* deletion strategy. Every cell in the table stores both an *item* and a *label*; the possible labels are **EMPTY**, **FULL**, and **JUNK**. The **DELETE** operation marks cells as **JUNK** instead of actually erasing their contents. Then **FIND** pretends that **JUNK** cells are occupied, and **INSERT** pretends that **JUNK** cells are actually empty. In more detail:

<u><b>FIND</b></u> ( $H, x$ ): <pre>           for i ← 0 to m – 1             j ← <math>h_i(x)</math>             if <math>H.\text{label}[j] = \text{FULL}</math> and <math>H.\text{item}[j] = x</math>               return j             else if <math>H.\text{label}[j] = \text{EMPTY}</math>               return None</pre>	<u><b>INSERT</b></u> ( $H, x$ ): <pre>           for i ← 0 to m – 1             j ← <math>h_i(x)</math>             if <math>H.\text{label}[j] = \text{FULL}</math> and <math>H.\text{item}[j] = x</math>               return «already there»             if <math>H.\text{label}[j] \neq \text{FULL}</math>               <math>H.\text{item}[j] \leftarrow x</math>               <math>H.\text{label}[j] \leftarrow \text{FULL}</math>             return</pre>
<u><b>DELETE</b></u> ( $H, x$ ): <pre>           j ← <b>FIND</b>(<math>H, x</math>)           if <math>j \neq \text{None}</math>             <math>H.\text{label}[j] \leftarrow \text{JUNK}</math></pre>	

Lazy deletion is always *correct*, but it is only *efficient* if we don't perform too many deletions. The search time depends on the fraction of non-**EMPTY** cells, not on the number of actual items stored in the table; thus, even if the number of items stays small, the table may fill up with **JUNK** cells, causing unsuccessful searches to scan the entire table. Less significantly, the data structure may use significantly more space than necessary for the number of items it actually stores. To avoid both of these issues, we use the following rebuilding rules:

- After each **INSERT** operation, if less than 1/4 of the cells are **EMPTY**, rebuild the hash table.
- After each **DELETE** operation, if less than 1/4 of the cells are **FULL**, rebuild the hash table.

To rebuild the hash table, we allocate a new hash table whose size is twice the number of **FULL** cells (unless that number is smaller than some fixed constant), **INSERT** each item in a **FULL** cell in the old hash table into the new hash table, and then discard the old hash table, as follows:

```
REBUILD( $H$ ):
   $count \leftarrow 0$ 
  for  $j \leftarrow 0$  to  $H.size - 1$ 
    if  $H.label[j] = FULL$ 
       $count \leftarrow count + 1$ 
   $H' \leftarrow$  new hash table of size  $\max\{2 \cdot count, 32\}$ 
  for  $j \leftarrow 0$  to  $H.size - 1$ 
    if  $H.label[j] = FULL$ 
      INSERT( $H'$ ,  $H.item[j]$ )
  discard  $H$ 
  return  $H'$ 
```

Finally, here are your actual homework questions!

- Describe a *small* example where the “obvious” deletion algorithm is incorrect; that is, show that the hash table can reach a state where a search can return the wrong result. Assume collisions are resolved by linear probing.
- Suppose we use Knuth's lazy deletion strategy instead. Prove that after several **INSERT** and **DELETE** operations into a table of arbitrary size  $m$ , it is possible for a single item  $x$  to be stored in *almost half* of the table cells. (However, at most one of those cells can be labeled **FULL**.)
- For purposes of analysis,<sup>1</sup> suppose **FIND** and **INSERT** run in  $O(1)$  time when at least 1/4 of the table cells are **EMPTY**. Prove that in any intermixed sequence of **INSERT** and **DELETE** operations, using Knuth's lazy deletion strategy, the amortized time per operation is  $O(1)$ .

\*3. **Extra credit.** Submit your answer to Homework 4 problem 3.

---

<sup>1</sup>In fact, **FIND** and **INSERT** run in  $O(1)$  *expected* time when at least 1/4 of the table cells are **EMPTY**, and therefore each **INSERT** and **DELETE** takes  $O(1)$  *expected* amortized time. But probability doesn't play any role whatsoever in the amortized analysis, so we can safely ignore the word “expected”.

1. Suppose we want to maintain an array  $X[1..n]$  of bits, which are all initially subject to the following operations.

- $\text{LOOKUP}(i)$ : Given an index  $i$ , return  $X[i]$ .
- $\text{BLACKEN}(i)$ : Given an index  $i < n$ , set  $X[i] \leftarrow 1$ .
- $\text{NEXTWHITE}(i)$ : Given an index  $i$ , return the smallest index  $j \geq i$  such that  $X[j] = 0$ . (Because we never change  $X[n]$ , such an index always exists.)

If we use the array  $X[1..n]$ , it is trivial to implement  $\text{LOOKUP}$  and  $\text{BLACKEN}$  in  $O(1)$  time and  $\text{NEXTWHITE}$  in  $O(n)$  time. But you can do better! Describe data structures that support  $\text{LOOKUP}$  in  $O(1)$  worst-case time and the other two operations in the following time bounds. (We want a different data structure for each set of time bounds, not one data structure that satisfies all bounds simultaneously!)

- (a) The worst-case time for both  $\text{BLACKEN}$  and  $\text{NEXTWHITE}$  is  $O(\log n)$ .
  - (b) The amortized time for both  $\text{BLACKEN}$  and  $\text{NEXTWHITE}$  is  $O(\log n)$ . In addition, the *worst-case* time for  $\text{BLACKEN}$  is  $O(1)$ .
  - (c) The amortized time for  $\text{BLACKEN}$  is  $O(\log n)$ , and the worst-case time for  $\text{NEXTWHITE}$  is  $O(1)$ .
  - (d) The worst-case time for  $\text{BLACKEN}$  is  $O(1)$ , and the amortized time for  $\text{NEXTWHITE}$  is  $O(\alpha(n))$ .
- [Hint: There is no WHITEN.]*

2. Recall that a standard (FIFO) queue maintains a sequence of items subject to the following operations:

- $\text{PUSH}(x)$ : Add item  $x$  to the back of the queue (the end of the sequence).
- $\text{PULL}()$ : Remove and return the item at the front of the queue (the beginning of the sequence).

It is easy to implement a queue using a doubly-linked list and a counter, using  $O(n)$  space altogether, so that each  $\text{PUSH}$  or  $\text{PULL}$  requires  $O(1)$  time.

- (a) Now suppose we want to support the following operation instead of  $\text{PULL}$ :
  - $\text{MULTIPULL}(k)$ : Remove the first  $k$  items from the front of the queue, and return the  $k$ th item removed.

Suppose further that we implement  $\text{MULTIPULL}$  using the obvious algorithm:

```

MULTIPULL( $k$ ):
  for  $i \leftarrow 1$  to  $k$ 
     $x \leftarrow \text{PULL}()$ 
  return  $x$ 
  
```

Prove that in any intermixed sequence of  $\text{PUSH}$  and  $\text{MULTIPULL}$  operations, starting with an empty queue, the amortized cost of each operation is  $O(1)$ . You may assume that  $k$  is never larger than the number of items in the queue.

- (b) Now suppose we **also** want to support the following operation instead of  $\text{PUSH}$ :

- $\text{MULTIPUSH}(x, k)$ : Insert  $k$  copies of  $x$  into the back of the queue.

Suppose further that we implement  $\text{MULTIPUSH}$  using the obvious algorithm:

<b>MULTIPUSH(<math>k, x</math>):</b> for $i \leftarrow 1$ to $k$ $\text{PUSH}(x)$
---

Prove that for any integers  $\ell$  and  $n$ , there is a sequence of  $\ell$  MULTIPUSH and MULTIPULL operations that require  $\Omega(n\ell)$  time, where  $n$  is the maximum number of items in the queue at any time. Such a sequence implies that the amortized cost of each operation is  $\Omega(n)$ .

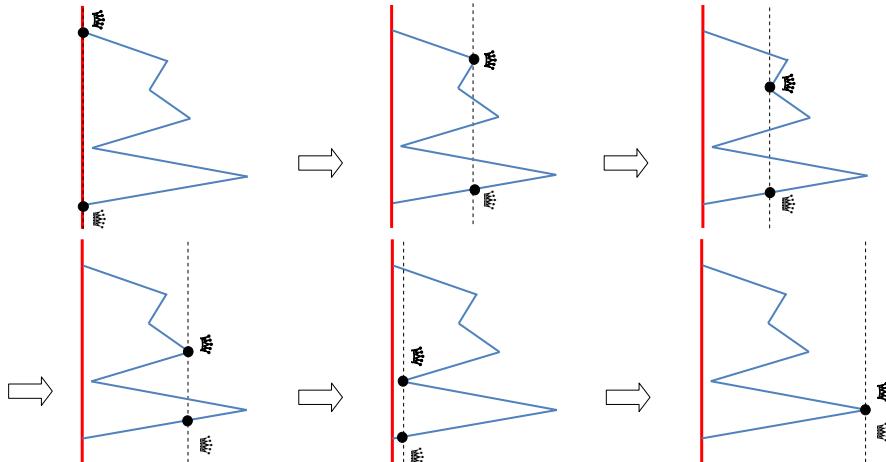
- (c) Finally, describe a data structure that supports arbitrary intermixed sequences of MULTIPUSH and MULTIPULL operations in  $O(1)$  amortized cost per operation. Like a standard queue, your data structure must use only  $O(1)$  space per item. [Hint: **Don't** use the obvious algorithms!]
3. In every cheesy romance movie there's always that scene where the romantic couple, physically separated and looking for one another, suddenly matches eyes and then slowly approach one another with unwavering eye contact as the music rolls and in and the rain lifts and the sun shines through the clouds and kittens and puppies....

Suppose a romantic couple—in grand computer science tradition, named Alice and Bob—enters a park from the northwest and southwest corners of the park, locked in dramatic eye contact. However, they can't just walk to one another in a straight line, because the paths of the park zig-zag between the northwest and southwest entrances. Instead, Alice and Bob must traverse the zig-zagging path so that their eyes are always locked perfectly in vertical eye-contact; thus, their  $x$ -coordinates must always be identical.

We can describe the zigzag path as an array  $P[0..n]$  of points, which are the corners of the path in order from the southwest endpoint to the northwest endpoint, satisfying the following conditions:

- $P[i].y > P[i - 1].y$  for every index  $i$ . That is, the path always moves upward.
- $P[0].x = P[n].x = 0$ , and  $P[i].x > 0$  for every index  $1 \leq i \leq n - 1$ . Thus, the ends of the path are further to the left than any other point on the path.

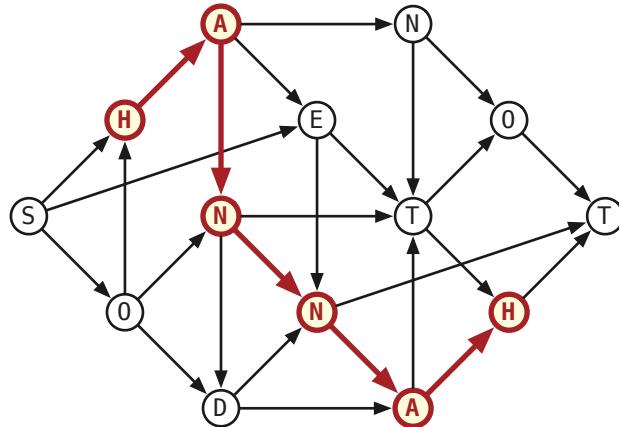
Prove that Alice and Bob can *always* meet.<sup>1</sup> [Hint: Describe a graph that models all possible locations of the couple along the path. What are the vertices of this graph? What are the edges? What can we say about the degrees of the vertices?]



<sup>1</sup>It follows that every cheesy romance movie (that reaches this scene) must have a happy, sappy ending.

1. Suppose we are given a directed acyclic graph  $G$  with labeled vertices. Every path in  $G$  has a label, which is a string obtained by concatenating the labels of its vertices in order. Recall that a *palindrome* is a string that is equal to its reversal.

Describe and analyze an algorithm to find the length of the longest palindrome that is the label of a path in  $G$ . For example, given the graph below, your algorithm should return the integer 6, which is the length of the palindrome **HANNAH**.



2. Let  $G$  be a connected directed graph that contains both directions of every edge; that is, if  $u \rightarrow v$  is an edge in  $G$ , its reversal  $v \rightarrow u$  is also an edge in  $G$ . Consider the following non-standard traversal algorithm.

```
SPAGHETTITRAVERSAL( $G$ ):
  for all vertices  $v$  in  $G$ 
    unmark  $v$ 
  for all edges  $u \rightarrow v$  in  $G$ 
    color  $u \rightarrow v$  white
   $s \leftarrow$  any vertex in  $G$ 
  SPAGHETTI( $s$ )
```

<u>SPAGHETTI(<math>v</math>):</u> mark $v$ <span style="float: right;">{{"visit <math>v</math>"}}</span> if there is a white arc $v \rightarrow w$ if $w$ is unmarked           color $w \rightarrow v$ green           color $v \rightarrow w$ red <span style="float: right;">{{"traverse <math>v \rightarrow w</math>"}}</span> SPAGHETTI( $w$ )         else if there is a green arc $v \rightarrow w$ color $v \rightarrow w$ red <span style="float: right;">{{"traverse <math>v \rightarrow w</math>"}}</span> SPAGHETTI( $w$ )       {{else every arc $v \rightarrow w$ is red, so halt}}
---

We informally say that this algorithm “visits” vertex  $v$  every time it marks  $v$ , and it “traverses” edge  $v \rightarrow w$  when it colors that edge red. Unlike our standard graph-traversal algorithms, SPAGHETTI may (in fact, will) mark/visit each vertex more than once.

The following series of exercises leads to a proof that SPAGHETTI traverses each directed edge of  $G$  exactly once. Most of the solutions are very short.

- (a) Prove that no directed edge in  $G$  is traversed more than once.
- (b) When the algorithm visits a vertex  $v$  for the  $k$ th time, exactly how many edges into  $v$  are red, and exactly how many edges out of  $v$  are red? [Hint: Consider the starting vertex  $s$  separately from the other vertices.]

- (c) Prove each vertex  $v$  is visited at most  $\deg(v)$  times, except the starting vertex  $s$ , which is visited at most  $\deg(s) + 1$  times. This claim immediately implies that  $\text{SPAGHETTI TRAVERSAL}(G)$  terminates.
- (d) Prove that when  $\text{SPAGHETTI TRAVERSAL}(G)$  ends, the last visited vertex is the starting vertex  $s$ .
- (e) For every vertex  $v$  that  $\text{SPAGHETTI TRAVERSAL}(G)$  visits, prove that all edges incident to  $v$  (either in or out) are red when  $\text{SPAGHETTI TRAVERSAL}(G)$  halts. [Hint: Consider the vertices in the order that they are marked for the first time, starting with  $s$ , and prove the claim by induction.]
- (f) Prove that  $\text{SPAGHETTI TRAVERSAL}(G)$  visits every vertex of  $G$ .
- (g) Finally, prove that  $\text{SPAGHETTI TRAVERSAL}(G)$  traverses every edge of  $G$  exactly once.

1. Let  $G$  be a directed graph with (possibly negative!) edge weights, and let  $s$  be an arbitrary vertex of  $G$ . Suppose every vertex  $v \neq s$  stores a pointer  $\text{pred}(v)$  to another vertex in  $G$ .

Describe and analyze an algorithm to determine whether these predecessor pointers define a single-source shortest path tree rooted at  $s$ . Do **not** assume that the graph  $G$  has no negative cycles.

*[Hint: There is a similar problem in head-banging, where you're given distances instead of predecessor pointers.]*

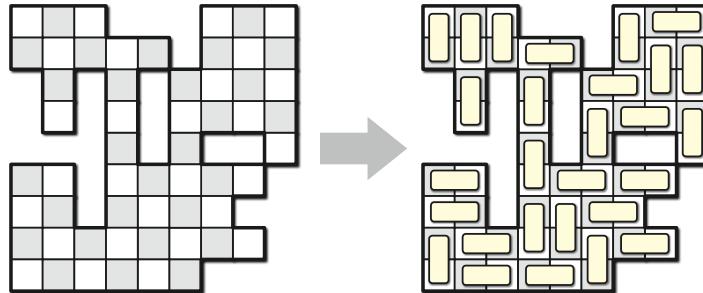
2. Let  $G$  be a directed graph with positive edge weights, and let  $s$  and  $t$  be arbitrary vertices of  $G$ . Describe an algorithm to determine the *number* of different shortest paths in  $G$  from  $s$  to  $t$ . Assume that you can perform arbitrary arithmetic operations in  $O(1)$  time. *[Hint: Which edges of  $G$  belong to shortest paths from  $s$  to  $t$ ?]*
3. Describe and analyze an algorithm to find the second smallest spanning tree of a given undirected graph  $G$  with weighted edges, that is, the spanning tree of  $G$  with smallest total weight except for the minimum spanning tree.

1. You're organizing the First Annual UIUC Computer Science 72-Hour Dance Exchange, to be held all day Friday, Saturday, and Sunday. Several 30-minute sets of music will be played during the event, and a large number of DJs have applied to perform. You need to hire DJs according to the following constraints.

- Exactly  $k$  sets of music must be played each day, and thus  $3k$  sets altogether.
- Each set must be played by a single DJ in a consistent music genre (ambient, bubblegum, dubstep, horrorcore, hyphy, trip-hop, Nitphonot, Kwaito, J-pop, Nashville country, ...).
- Each genre must be played at most once per day.
- Each candidate DJ has given you a list of genres they are willing to play.
- Each DJ can play at most three sets during the entire event.

Suppose there are  $n$  candidate DJs and  $g$  different musical genres available. Describe and analyze an efficient algorithm that either assigns a DJ and a genre to each of the  $3k$  sets, or correctly reports that no such assignment is possible.

2. Suppose you are given an  $n \times n$  checkerboard with some of the squares deleted. You have a large set of dominos, just the right size to cover two squares of the checkerboard. Describe and analyze an algorithm to determine whether one can tile the board with dominos—each domino must cover exactly two undeleted squares, and each undeleted square must be covered by exactly one domino.



Your input is a two-dimensional array  $Deleted[1..n, 1..n]$  of bits, where  $Deleted[i, j] = \text{TRUE}$  if and only if the square in row  $i$  and column  $j$  has been deleted. Your output is a single bit; you do **not** have to compute the actual placement of dominos. For example, for the board shown above, your algorithm should return TRUE.

3. Suppose we are given an array  $A[1..m][1..n]$  of non-negative real numbers. We want to *round*  $A$  to an integer matrix, by replacing each entry  $x$  in  $A$  with either  $\lfloor x \rfloor$  or  $\lceil x \rceil$ , without changing the sum of entries in any row or column of  $A$ . For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

Describe and analyze an efficient algorithm that either rounds  $A$  in this fashion, or reports correctly that no such rounding is possible.

1. For any integer  $k$ , the problem  $k$ -COLOR asks whether the vertices of a given graph  $G$  can be colored using at most  $k$  colors so that neighboring vertices does not have the same color.

- (a) Prove that  $k$ -COLOR is NP-hard, for every integer  $k \geq 3$ .
- (b) Now fix an integer  $k \geq 3$ . Suppose you are given a magic black box that can determine **in polynomial time** whether an arbitrary graph is  $k$ -colorable; the box returns TRUE if the given graph is  $k$ -colorable and FALSE otherwise. The input to the magic black box is a graph. Just a graph. Vertices and edges. Nothing else.

Describe and analyze a **polynomial-time** algorithm that either computes a proper  $k$ -coloring of a given graph  $G$  or correctly reports that no such coloring exists, using this magic black box as a subroutine.

2. A boolean formula is in *conjunctive normal form* (or *CNF*) if it consists of a *conjunction* (AND) of several *terms*, each of which is the disjunction (OR) of one or more literals. For example, the formula

$$(\bar{x} \vee y \vee \bar{z}) \wedge (y \vee z) \wedge (x \vee \bar{y} \vee \bar{z})$$

is in conjunctive normal form. The problem **CNF-SAT** asks whether a boolean formula in conjunctive normal form is satisfiable. 3SAT is the special case of CNF-SAT where every clause in the input formula must have exactly three literals; it follows immediately that CNF-SAT is NP-hard.

Symmetrically, a boolean formula is in *disjunctive normal form* (or *DNF*) if it consists of a *disjunction* (OR) of several *terms*, each of which is the conjunction (AND) of one or more literals. For example, the formula

$$(\bar{x} \wedge y \wedge \bar{z}) \vee (y \wedge z) \vee (x \wedge \bar{y} \wedge \bar{z})$$

is in disjunctive normal form. The problem DNF-SAT asks whether a boolean formula in disjunctive normal form is satisfiable.

- (a) Describe a polynomial-time algorithm to solve DNF-SAT.
  - (b) Describe a reduction from CNF-SAT to DNF-SAT.
  - (c) Why do parts (a) and (b) not imply that P=NP?
3. The 42-PARTITION problem asks whether a given set  $S$  of  $n$  positive integers can be partitioned into subsets  $A$  and  $B$  (meaning  $A \cup B = S$  and  $A \cap B = \emptyset$ ) such that

$$\sum_{a \in A} a = 42 \sum_{b \in B} b$$

For example, we can 42-partition the set  $\{1, 2, 34, 40, 52\}$  into  $A = \{34, 40, 52\}$  and  $B = \{1, 2\}$ , since  $\sum A = 126 = 42 \cdot 3$  and  $\sum B = 3$ . But the set  $\{4, 8, 15, 16, 23, 42\}$  cannot be 42-partitioned.

- (a) Prove that 42-PARTITION is NP-hard.
- (b) Let  $M$  denote the largest integer in the input set  $S$ . Describe an algorithm to solve 42-PARTITION in time polynomial in  $n$  and  $M$ . For example, your algorithm should return TRUE when  $S = \{1, 2, 34, 40, 52\}$  and FALSE when  $S = \{4, 8, 15, 16, 23, 42\}$ .
- (c) Why do parts (a) and (b) not imply that P=NP?

# CS 473: Undergraduate Algorithms, Fall 2013

## Headbanging 0: Induction!

August 28 and 29

---

1. Prove that any non-negative integer can be represented as the sum of distinct powers of 2. (“Write it in binary” is not a proof; it’s just a restatement of what you have to prove.)
2. Prove that every integer (positive, negative, or zero) can be written in the form  $\sum_i \pm 3^i$ , where the exponents  $i$  are distinct non-negative integers. For example:

$$42 = 3^4 - 3^3 - 3^2 - 3^1 \quad 25 = 3^3 - 3^1 + 3^0 \quad 17 = 3^3 - 3^2 - 3^0$$

3. Recall that a **full binary tree** is either an isolated *leaf*, or an *internal node* with a left subtree and a right subtree, each of which is a full binary tree. Equivalently, a binary tree is **full** if every internal node has exactly two children. Give at least *three different* proofs of the following fact: *In every full binary tree, the number of leaves is exactly one more than the number of internal nodes.*
- 

### Take-home points:

- Induction is recursion. Recursion is induction.
- All induction is strong/structural induction. There is absolutely no point in using a weak induction hypothesis. None. Ever.
- To prove that all snarks are boojums, start with an *arbitrary* snark and remove some tentacles. Do not start with a smaller snark and try to add tentacles. Snarks don’t like that.
- Every induction proof requires an exhaustive case analysis. Write down the cases. Make sure they’re exhaustive.
- Do the most general cases first. Whatever is left over are the base cases.
- The empty set is the best base case.

*Khelm is Warsaw. Warsaw is Khelm. Khelm is Warsaw. Zay gezunt!  
Warsaw is Khelm. Khelm is Warsaw. Warsaw is Khelm. For gezunt!*

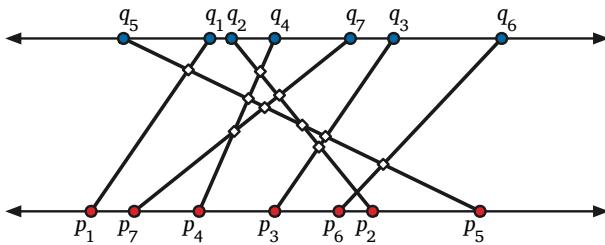
— Golem (feat. Amanda Palmer), “Warsaw is Khelm”, *Fresh Off Boat* (2006)

1. An *inversion* in an array  $A[1..n]$  is a pair of indices  $(i, j)$  such that  $i < j$  and  $A[i] > A[j]$ . The number of inversions in an  $n$ -element array is between 0 (if the array is sorted) and  $\binom{n}{2}$  (if the array is sorted backward).

Describe and analyze a divide-and-conquer algorithm to count the number of inversions in an  $n$ -element array in  $O(n \log n)$  time. Assume all the elements of the input array are distinct.

2. Suppose you are given two sets of  $n$  points, one set  $\{p_1, p_2, \dots, p_n\}$  on the line  $y = 0$  and the other set  $\{q_1, q_2, \dots, q_n\}$  on the line  $y = 1$ . Create a set of  $n$  line segments by connect each point  $p_i$  to the corresponding point  $q_i$ . Describe and analyze a divide-and-conquer algorithm to determine how many pairs of these line segments intersect, in  $O(n \log n)$  time. [Hint: Use your solution to problem 1.]

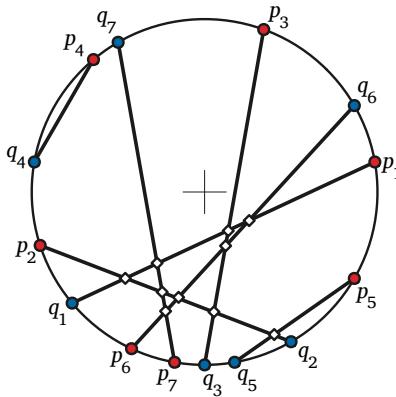
Assume a reasonable representation for the input points, and assume the  $x$ -coordinates of the input points are distinct. For example, for the input shown below, your algorithm should return the number 10.



Ten intersecting pairs of segments with endpoints on parallel lines.

3. Now suppose you are given two sets  $\{p_1, p_2, \dots, p_n\}$  and  $\{q_1, q_2, \dots, q_n\}$  of  $n$  points *on the unit circle*. Connect each point  $p_i$  to the corresponding point  $q_i$ . Describe and analyze a divide-and-conquer algorithm to determine how many pairs of these line segments intersect in  $O(n \log^2 n)$  time. [Hint: Use your solution to problem 2.]

Assume a reasonable representation for the input points, and assume all input points are distinct. For example, for the input shown below, your algorithm should return the number 10.



Ten intersecting pairs of segments with endpoints on a circle.

4. **To think about later:** Solve problem 3 in  $O(n \log n)$  time.

1. A *longest common subsequence* of a set of strings  $\{A_i\}$  is a longest string that is a subsequence of  $A_i$  for each  $i$ . For example, alrit is a longest common subsequence of strings

algorithm and altruistic.

Given two strings  $A[1..n]$  and  $B[1..n]$ , describe and analyze a dynamic programming algorithm that computes the length of a longest common subsequence of the two strings in  $O(n^2)$  time.

2. Describe and analyze a dynamic programming algorithm that computes the length of a longest common subsequence of three strings  $A[1..n]$ ,  $B[1..n]$ , and  $C[1..n]$  in  $O(n^3)$  time. [Hint: Try **not** to use your solution to problem 1 directly.]
3. A *lucky-10 number* is a string  $D[1..n]$  of digits from 1 to 9 (no zeros), such that the  $i$ -th digit and the last  $i$ -th digit sum up to 10; in another words,  $D[i] + D[n - i + 1] = 10$  for all  $i$ . For example,

3141592648159697 and 11599

are both lucky-10 numbers. Given a string of digits  $D[1..n]$ , describe and analyze a dynamic programming algorithm that computes the length of a longest lucky-10 subsequence of the string. [Hint: Try to use your solution to problem 1 **directly**.]

4. **To think about later:** Can you solve problem 1 in  $O(n)$  space?

1. A *vertex cover* of a graph is a subset  $S$  of the vertices such that every vertex  $v$  either belongs to  $S$  or has a neighbor in  $S$ . In other words, the vertices in  $S$  cover all the edges. Finding the minimum size of a vertex cover is  $NP$ -hard, but in trees it can be found using dynamic programming.

Given a tree  $T$  and non-negative weight  $w(v)$  for each vertex  $v$ , describe an algorithm computing the minimum weight of a vertex cover of  $T$ .

2. Suppose you are given an unparenthesized mathematical expression containing  $n$  numbers, where the only operators are  $+$  and  $-$ ; for example:

$$1 + 3 - 2 - 5 + 1 - 6 + 7$$

You can change the value of the expression by adding parentheses in different positions. For example:

$$\begin{aligned} 1 + 3 - 2 - 5 + 1 - 6 + 7 &= -1 \\ (1 + 3 - (2 - 5)) + (1 - 6) + 7 &= 9 \\ (1 + (3 - 2)) - (5 + 1) - (6 + 7) &= -17 \end{aligned}$$

Design an algorithm that, given a list of integers separated by  $+$  and  $-$  signs, determines the maximum possible value the expression can take by adding parentheses.

You can only insert parentheses immediately before and immediately after numbers; in particular, you are not allowed to insert implicit multiplication as in  $1 + 3(-2)(-5) + 1 - 6 + 7 = 33$ .

3. Fix an arbitrary sequence  $c_1 < c_2 < \dots < c_k$  of coin values, all in cents. We have an infinite number of coins of each denomination. Describe a dynamic programming algorithm to determine, given an arbitrary non-negative integer  $x$ , the least number of coins whose total value is  $x$ . For simplicity, you may assume that  $c_1 = 1$ .

**To think about later after learning “greedy algorithms”:**

- (a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.
- (b) Suppose that the available coins have the values  $c^0, c^1, \dots, c^k$  for some integers  $c > 1$  and  $k \geq 1$ . Show that the greedy algorithm always yields an optimal solution.
- (c) Describe a set of 4 coin values for which the greedy algorithm does *not* yield an optimal solution.

**Note:** All the questions in this session are taken from past CS473 midterms.

1. (Fall 2006) **Multiple Choice:** Each of the questions on this page has one of the following five answers: For each question, write the letter that corresponds to your answer.

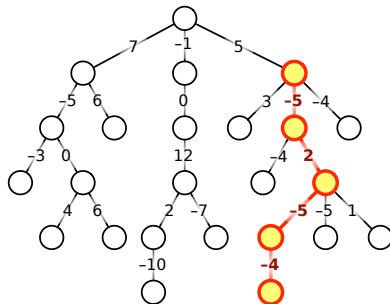
A:  $\Theta(1)$  B:  $\Theta(\log n)$  C:  $\Theta(n)$  D:  $\Theta(n \log n)$  E:  $\Theta(n^2)$

- (a) What is  $\frac{5}{n} + \frac{n}{5}$ ?
- (b) What is  $\sum_{i=1}^n \frac{n}{i}$ ?
- (c) What is  $\sum_{i=1}^n \frac{i}{n}$ ?
- (d) How many bits are required to represent the nth Fibonacci number in binary?
- (e) What is the solution to the recurrence  $T(n) = 2T(n/4) + \Theta(n)$ ?
- (f) What is the solution to the recurrence  $T(n) = 16T(n/4) + \Theta(n)$ ?
- (g) What is the solution to the recurrence  $T(n) = T(n - 1) + \frac{1}{n^2}$ ?
- (h) What is the worst-case time to search for an item in a binary search tree?
- (i) What is the worst-case running time of quicksort?
- (j) What is the running time of the fastest possible algorithm to solve Sudoku puzzles? A Sudoku puzzle consists of a  $9 \times 9$  grid of squares, partitioned into nine  $3 \times 3$  sub-grids; some of the squares contain digits between 1 and 9. The goal of the puzzle is to enter digits into the blank squares, so that each digit between 1 and 9 appears exactly once in each row, each column, and each  $3 \times 3$  sub-grid. The initial conditions guarantee that the solution is unique.

2								4
	7		5					
				1		9		
6	4			2				
	8						5	
		9			3		7	
		1	4					
				3		8		
	5							6

A Sudoku puzzle. **Don't try to solve this during the exam!**

2. (Spring 2010) Let  $T$  be a rooted tree with integer weights on its edges, which could be positive, negative, or zero. The weight of a path in  $T$  is the sum of the weights of its edges. Describe and analyze an algorithm to compute the minimum weight of any path from a node in  $T$  down to one of its descendants. It is not necessary to compute the actual minimum-weight path; just its weight. For example, given the tree shown below, your algorithm should return the number -12.
3. (Fall 2006) Suppose you are given an array  $A[1..n]$  of  $n$  distinct integers, sorted in increasing order. Describe and analyze an algorithm to determine whether there is an index  $i$  such that  $A[i] = i$ , in  $\Theta(n)$  time. [Hint: Yes, that's little-oh of  $n$ . What can you say about the sequence  $A[i] - i$ ?]



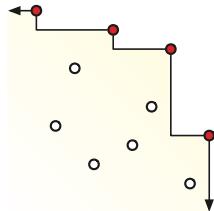
The minimum-weight downward path in this tree has weight  $-12$ .

4. (Spring 2010 and Spring 2004) Describe and analyze efficient algorithms to solve the following problems:

- (a) Given a set of  $n$  integers, does it contain two elements  $a, b$  such that  $a + b = 0$ ?
- (b) Given a set of  $n$  integers, does it contain three elements  $a, b, c$  such that  $a + b = c$ ?

1. What is the *exact* expected number of leaves in a treap with  $n$  nodes?
2. Recall question 5 from Midterm 1:

Suppose you are given a set  $P$  of  $n$  points in the plane. A point  $p \in P$  is *maximal* in  $P$  if no other point in  $P$  is both above and to the right of  $p$ . Intuitively, the maximal points define a “staircase” with all the other points of  $P$  below it.



A set of ten points, four of which are maximal.

Describe and analyze an algorithm to compute the number of maximal points in  $P$  in  $O(n \log n)$  time. For example, given the ten points shown above, your algorithm should return the integer 4.

Suppose the points in  $P$  are generated independently and uniformly at random in the unit square  $[0, 1]^2$ . What is the *exact* expected number of maximal points in  $P$ ?

3. Suppose you want to write an app for your new Pebble smart watch that monitors the global Twitter stream and selects a small sample of *random* tweets. You will not know when the stream ends until your app attempts to read the next tweet and receives the error message FAILWHALE. The Pebble has only a small amount of memory, far too little to store the entire stream.
  - (a) Describe an algorithm that, as soon as the stream ends, returns a single tweet chosen uniformly at random from the stream. Prove your algorithm is correct. (You may assume that the stream contains at least one tweet.)
  - (b) Now fix an arbitrary positive integer  $k$ . Describe an algorithm that picks  $k$  tweets uniformly at random from the stream. Prove your algorithm is correct. (You may assume that the stream contains at least  $k$  tweets.)

Recall the following elementary data structures from CS 225.

- A *stack* supports the following operations.
  - PUSH pushes an element on top of the stack.
  - POP removes the top element from a stack.
  - IsEMPTY checks if a stack is empty.
- A *queue* supports the following operations.
  - PUSH adds an element to the back of the queue.
  - PULL removes an element from the front of the queue.
  - IsEMPTY checks if a queue is empty.
- A *deque*, or double-ended queue, supports the following operations.
  - PUSH adds an element to the back of the queue.
  - PULL removes an element from the back of the queue.
  - CUT adds an element from the front of the queue.
  - POP removes an element from the front of the queue.
  - IsEMPTY checks if a queue is empty.

Suppose you have a stack implementation that supports all stack operations in constant time.

1. Describe how to implement a queue using two stacks and  $O(1)$  additional memory, so that each queue operation runs in  $O(1)$  amortized time.
2. Describe how to implement a deque using three stacks and  $O(1)$  additional memory, so that each deque operation runs in  $O(1)$  amortized time.

1. Let  $P$  be a set of  $n$  points in the plane. Recall from the midterm that the *staircase* of  $P$  is the set of all points in the plane that have at least one point in  $P$  both above and to the right.
  - (a) Describe and analyze a data structure that stores the staircase of a set of points, and an algorithm  $\text{ABOVE?}(x, y)$  that returns TRUE if the point  $(x, y)$  is above the staircase, or FALSE otherwise. Your data structure should use  $O(n)$  space, and your  $\text{ABOVE?}$  algorithm should run in  $O(\log n)$  time.
  - (b) Describe and analyze a data structure that maintains the staircase of a set of points as new points are inserted. Specifically, your data structure should support a function  $\text{INSERT}(x, y)$  that adds the point  $(x, y)$  to the underlying point set and returns TRUE or FALSE to indicate whether the staircase of the set has changed. Your data structure should use  $O(n)$  space, and your  $\text{INSERT}$  algorithm should run in  $O(\log n)$  amortized time.
2. An *ordered stack* is a data structure that stores a sequence of items and supports the following operations.
  - $\text{ORDEREDPUSH}(x)$  removes all items smaller than  $x$  from the beginning of the sequence and then adds  $x$  to the beginning of the sequence.
  - $\text{POP}$  deletes and returns the first item in the sequence (or  $\text{NULL}$  if the sequence is empty).

Suppose we implement an ordered stack with a simple linked list, using the obvious  $\text{ORDEREDPUSH}$  and  $\text{POP}$  algorithms. Prove that if we start with an empty data structure, the amortized cost of each  $\text{ORDEREDPUSH}$  or  $\text{POP}$  operation is  $O(1)$ .

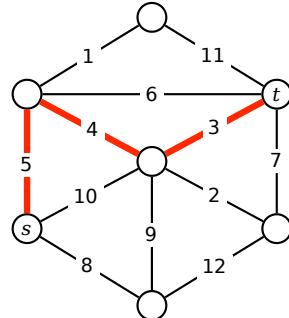
3. Consider the following solution for the union-find problem, called *union-by-weight*. Each set leader  $\bar{x}$  stores the number of elements of its set in the field  $\text{weight}(\bar{x})$ . Whenever we  $\text{UNION}$  two sets, the leader of the *smaller* set becomes a new child of the leader of the *larger* set (breaking ties arbitrarily).

$\text{MAKESET}(x)$ $\text{parent}(x) \leftarrow x$ $\text{weight}(x) \leftarrow 1$	$\text{UNION}(x, y)$ $\bar{x} \leftarrow \text{FIND}(x)$ $\bar{y} \leftarrow \text{FIND}(y)$ if $\text{weight}(\bar{x}) > \text{weight}(\bar{y})$ $\text{parent}(\bar{y}) \leftarrow \bar{x}$ $\text{weight}(\bar{x}) \leftarrow \text{weight}(\bar{x}) + \text{weight}(\bar{y})$ else $\text{parent}(\bar{x}) \leftarrow \bar{y}$ $\text{weight}(\bar{x}) \leftarrow \text{weight}(\bar{x}) + \text{weight}(\bar{y})$
$\text{FIND}(x)$ while $x \neq \text{parent}(x)$ $x \leftarrow \text{parent}(x)$ return $x$	

Prove that if we use union-by-weight, the *worst-case* running time of  $\text{FIND}(x)$  is  $O(\log n)$ , where  $n$  is the cardinality of the set containing  $x$ .

1. Let  $G$  be an undirected graph.
  - (a) Suppose we start with two coins on two arbitrarily chosen nodes. At every step, each coin must move to an adjacent node. Describe an algorithm to compute the minimum number of steps to reach a configuration that two coins are on the same node.
  - (b) Now suppose there are three coins, numbered 0, 1, and 2. Again we start with an arbitrary coin placement with all three coins facing up. At each step, we move each coin to an adjacent node at each step. Moreover, for every integer  $i$ , we flip coin  $i \bmod 3$  at the  $i$ th step. Describe an algorithm to compute the minimum number of steps to reach a configuration that all three coins are on the same node and all facing up. What is the running time of your algorithm?
2. Let  $G$  be a directed acyclic graph with a unique source  $s$  and a unique sink  $t$ .
  - (a) A *Hamiltonian path* in  $G$  is a directed path in  $G$  that contains every vertex in  $G$ . Describe an algorithm to determine whether  $G$  has a Hamiltonian path.
  - (b) Suppose several nodes in  $G$  are marked to be *important*; also an integer  $k$  is given. Design an algorithm which computes all the nodes that can reach  $t$  through at least  $k$  important nodes.
  - (c) Suppose the edges in  $G$  have real weights. Describe an algorithm to find a path from  $s$  to  $t$  with maximum total weight.
  - (d) Suppose the vertices of  $G$  have labels from a fixed finite alphabet, and let  $A[1..\ell]$  be a string over the same alphabet. Any directed path in  $G$  has a label, which is obtained by concatenating the labels of its vertices. Describe an algorithm to find the longest path in  $G$  whose labels are a subsequence of  $A$ .
3. Let  $G$  be a directed graph with a special source that has an edge to each other node in graph, and denote  $scc(G)$  as the strong component graph of  $G$ . Let  $S$  and  $S'$  be two strongly connected components in  $G$  with  $S \rightarrow S'$  an arc in  $scc(G)$ . (That is, if there is an arc between node  $u \in S$  and  $v \in S'$ , then it must be  $u \rightarrow v$ .) Consider a fixed depth-first search performed on  $G$  starting at  $s$ ; we define  $\text{post}(\cdot)$  as the post-order numbering of the search.
  - (a) Prove or disprove that we have  $\text{post}(u) > \text{post}(u')$  for any  $u \in S$  and  $u' \in S'$ .
  - (b) Prove or disprove that we have  $\max_{u \in S} \text{post}(u) > \max_{u' \in S'} \text{post}(u')$ .

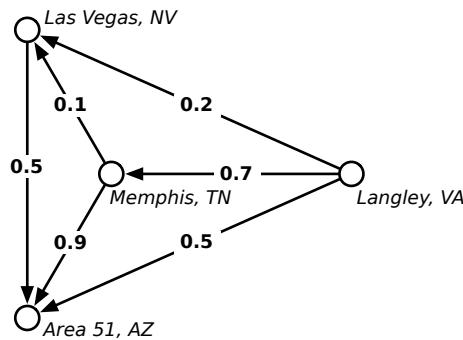
1. Consider a path between two vertices  $s$  and  $t$  in an undirected weighted graph  $G$ . The *bottleneck length* of this path is the maximum weight of any edge in the path. The *bottleneck distance* between  $s$  and  $t$  is the minimum bottleneck length of any path from  $s$  to  $t$ . (If there are no paths from  $s$  to  $t$ , the bottleneck distance between  $s$  and  $t$  is  $\infty$ )



Describe an algorithm to compute the bottleneck distance between *every* pair of vertices in an arbitrary undirected weighted graph. Assume that no two edges have the same weight.

2. Let  $G$  be a directed graph with (possibly negative) edge weights, and let  $s$  be an arbitrary vertex of  $G$ . Suppose for each vertex  $v$  we are given a real number  $d(v)$ . Describe and analyze an algorithm to determine whether the numbers  $d(v)$  on vertices are the shortest path distances from  $s$  to each vertex  $v$ . Do not assume that the graph  $G$  has no negative cycles.
3. Mulder and Scully have computed, for every road in the United States, the exact probability that someone driving on that road *won't* be abducted by aliens. Agent Mulder needs to drive from Langley, Virginia to Area 51, Nevada. What route should he take so that he has the least chance of being abducted?

More formally, you are given a directed graph  $G$ , possibly with cycles, where every edge  $e$  has an independent safety probability  $p(e)$ . The *safety* of a path is the product of the safety probabilities of its edges. Design and analyze an algorithm to determine the safest path from a given start vertex  $s$  to a given target vertex  $t$ .



For example, with the probabilities shown above, if Mulder tries to drive directly from Langley to Area 51, he has a 50% chance of getting there without being abducted. If he stops in Memphis, he has a  $0.7 \times 0.9 = 63\%$  chance of arriving safely. If he stops first in Memphis and then in Las Vegas, he has a  $1 - 0.7 \times 0.1 \times 0.5 = 96.5\%$  chance of not being abducted!<sup>1</sup>

<sup>1</sup>That's how they got Elvis, you know.

Almost all these review problems come from past midterms.

1. [Fall 2002, Spring 2004] Suppose we want to maintain a set  $X$  of numbers, under the following operations:

- $\text{INSERT}(x)$ : Add  $x$  to the set (if it isn't already there).
- $\text{PRINT\&DELETEBETWEEN}(a, b)$ : Print every element  $x \in X$  such that  $a \leq x \leq b$ , in order from smallest to largest, and then delete those elements from  $X$ .

For example, if the current set is  $\{1, 5, 3, 4, 8\}$ , then  $\text{PRINT\&DELETEBETWEEN}(4, 6)$  prints the numbers 4 and 5 and changes the set to  $\{1, 3, 8\}$ .

Describe and analyze a data structure that supports these two operations, each in  $O(\log n)$  amortized time, where  $n$  is the maximum number of elements in  $X$ .

2. [Spring 2004] Consider a random walk on a path with vertices numbered  $1, 2, \dots, n$  from left to right. We start at vertex 1. At each step, we flip a coin to decide which direction to walk, moving one step left or one step right with equal probability. The random walk ends when we fall off one end of the path, either by moving left from vertex 1 or by moving right from vertex  $n$ .

**Prove** that the probability that the walk ends by falling off the *left* end of the path is exactly  $n/(n+1)$ . [Hint: Set up a recurrence and verify that  $n/(n+1)$  satisfies it.]

3. [Fall 2006] **Prove or disprove** each of the following statements.

- Let  $G$  be an arbitrary undirected graph with arbitrary distinct weights on the edges. The minimum spanning tree of  $G$  includes the lightest edge in every cycle in  $G$ .
  - Let  $G$  be an arbitrary undirected graph with arbitrary distinct weights on the edges. The minimum spanning tree of  $G$  excludes the heaviest edge in every cycle in  $G$ .
4. [Fall 2012] Let  $G = (V, E)$  be a connected undirected graph. For any vertices  $u$  and  $v$ , let  $d_G(u, v)$  denote the length of the shortest path in  $G$  from  $u$  to  $v$ . For any sets of vertices  $A$  and  $B$ , let  $d_G(A, B)$  denote the length of the shortest path in  $G$  from any vertex in  $A$  to any vertex in  $B$ :

$$d_G(A, B) = \min_{u \in A} \min_{v \in B} d_G(u, v).$$

Describe and analyze a fast algorithm to compute  $d_G(A, B)$ , given the graph  $G$  and subsets  $A$  and  $B$  as input. You do not need to prove that your algorithm is correct.

5. Let  $G$  and  $H$  be directed acyclic graphs, whose vertices have labels from some fixed alphabet, and let  $A[1..l]$  be a string over the same alphabet. Any directed path in  $G$  has a label, which is a string obtained by concatenating the labels of its vertices.
- Describe an algorithm to find the longest string that is both a label of a directed path in  $G$  and the label of a directed path in  $H$ .
  - Describe an algorithm to find the longest string that is both a *subsequence* of the label of a directed path in  $G$  and *subsequence* of the label of a directed path in  $H$ .

1. The Island of Sodor is home to a large number of towns and villages, connected by an extensive rail network. Recently, several cases of a deadly contagious disease (either swine flu or zombies; reports are unclear) have been reported in the village of Ffarquhar. The controller of the Sodor railway plans to close down certain railway stations to prevent the disease from spreading to Tidmouth, his home town. No trains can pass through a closed station. To minimize expense (and public notice), he wants to close down as few stations as possible. However, he cannot close the Ffarquhar station, because that would expose him to the disease, and he cannot close the Tidmouth station, because then he couldn't visit his favorite pub.

Describe and analyze an algorithm to find the minimum number of stations that must be closed to block all rail travel from Ffarquhar to Tidmouth. The Sodor rail network is represented by an undirected graph, with a vertex for each station and an edge for each rail connection between two stations. Two special vertices  $f$  and  $t$  represent the stations in Ffarquhar and Tidmouth.

2. Given an undirected graph  $G = (V, E)$ , with three vertices  $u$ ,  $v$ , and  $w$ , describe and analyze an algorithm to determine whether there is a path from  $u$  to  $w$  that passes through  $v$ .
3. Suppose you have already computed a maximum flow  $f^*$  in a flow network  $G$  with *integer* edge capacities.
  - (a) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is increased by 1.
  - (b) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is decreased by 1.

Both algorithms should be significantly faster than recomputing the maximum flow from scratch.

1. An *American graph* is a directed graph with each vertex colored *red*, *white*, or *blue*. An *American Hamiltonian path* is a Hamiltonian path that cycles between red, white, and blue vertices; that is, every edge goes from red to white, or white to blue, or blue to red. The AMERICANHAMILTONIANPATH problem asks whether there is an American Hamiltonian path in an American graph.
  - (a) Prove that AMERICANHAMILTONIANPATH is NP-complete by reducing from HAMILTONIANPATH.
  - (b) In the opposite direction, reduce AMERICANHAMILTONIANPATH to HAMILTONIANPATH.
2. Given a graph  $G$ , the DEG17SPANNINGTREE problem asks whether  $G$  has a spanning tree in which each vertex of the spanning tree has degree at most 17. Prove that DEG17SPANNINGTREE is NP-complete.
3. Two graphs are *isomorphic* if one can be transformed into the other by relabeling the vertices. Consider the following related decision problems:
  - GRAPHISOMORPHISM: Given two graphs  $G$  and  $H$ , determine whether  $G$  and  $H$  are isomorphic.
  - EVENGRAPHISOMORPHISM: Given two graphs  $G$  and  $H$ , such that every vertex of  $G$  and  $H$  have even degree, determine whether  $G$  and  $H$  are isomorphic.
  - SUBGRAPHISOMORPHISM: Given two graphs  $G$  and  $H$ , determine whether  $G$  is isomorphic to a subgraph of  $H$ .
  - (a) Describe a polynomial time reduction from GRAPHISOMORPHISM to EVENGRAPHISOMORPHISM.
  - (b) Describe a polynomial time reduction from GRAPHISOMORPHISM to SUBGRAPHISOMORPHISM.

1. Prove that the following problem is NP-hard.

SETCOVER: Given a collection of sets  $\{S_1, \dots, S_m\}$ , find the smallest sub-collection of  $S_i$ 's that contains all the elements of  $\bigcup_i S_i$ .

2. Given an undirected graph  $G$  and a subset of vertices  $S$ , a *Steiner tree* of  $S$  in  $G$  is a subtree of  $G$  that contains every vertex in  $S$ . If  $S$  contains every vertex of  $G$ , a Steiner tree is just a spanning tree; if  $S$  contains exactly two vertices, any path between them is a Steiner tree.

Given a graph  $G$ , a vertex subset  $S$ , and an integer  $k$ , the *Steiner tree problem* requires us to decide whether there is a Steiner tree of  $S$  in  $G$  with at most  $k$  edges. Prove that the Steiner tree problem is NP-hard. [Hint: Reduce from VERTEXCOVER, or SETCOVER, or 3SAT.]

3. Let  $G$  be a directed graph whose edges are colored red and white. A *Canadian Hamiltonian path* is a Hamiltonian path whose edges are alternately red and white. The CANADIANHAMILTONIANPATH problem ask us to find a Canadian Hamiltonian path in a graph  $G$ . (Two weeks ago we looked for Hamiltonian paths that cycled through colors on the *vertices* instead of edges.)

- (a) Prove that CANADIANHAMILTONIANPATH is NP-Complete.
  - (b) Reduce CANADIANHAMILTONIANPATH to HAMILTONIANPATH.

This exam lasts 120 minutes.

**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

1. Each of these ten questions has one of the following five answers:

$$\text{A: } \Theta(1) \quad \text{B: } \Theta(\log n) \quad \text{C: } \Theta(n) \quad \text{D: } \Theta(n \log n) \quad \text{E: } \Theta(n^2)$$

(a) What is  $\frac{n^5 - 3n^3 - 5n + 4}{4n^3 - 2n^2 + n - 7}$ ?

(b) What is  $\sum_{i=1}^n i$ ?

(c) What is  $\sum_{i=1}^n \sqrt{\frac{n}{i}}$ ?

(d) How many bits are required to write the integer  $n^{10}$  in binary?

(e) What is the solution to the recurrence  $E(n) = E(n/2) + E(n/4) + E(n/8) + 16n$ ?

(f) What is the solution to the recurrence  $F(n) = 6F(n/6) + 6n$ ?

(g) What is the solution to the recurrence  $G(n) = 9G(\lceil n/3 \rceil + 1) + n$ ?

(h) The *total path length* of a binary tree is the sum of the depths of all nodes. What is the total path length of an  $n$ -node binary tree in the worst case?

(i) Consider the following recursive function, defined in terms of a fixed array  $X[1..n]$ :

$$WTF(i, j) = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{if } i = j \\ \max \left\{ \begin{array}{l} 2 \cdot [X[i] \neq X[j]] + WTF(i+1, j-1) \\ 1 + WTF(i+1, j) \\ 1 + WTF(i, j-1) \end{array} \right\} & \text{otherwise} \end{cases}$$

How long does it take to compute  $WTF(1, n)$  using dynamic programming?

- (j) Voyager 1 recently became the first made-made object to reach interstellar space. Currently the spacecraft is about 18 billion kilometers (roughly 60,000 light seconds) from Earth, traveling outward at approximately 17 kilometers per second (approximately 1/18000 of the speed of light). Voyager carries a golden record containing over 100 digital images and approximately one hour of sound recordings. In digital form, the recording would require about 1 gigabyte. Voyager can transmit data back to Earth at approximately 1400 bits per second. Suppose the engineers at JPL sent instructions to Voyager 1 to send the complete contents of the Golden Record back to Earth; how many seconds would they have to wait to receive the entire record?

2. You are a visitor at a political convention (or perhaps a faculty meeting) with  $n$  delegates; each delegate is a member of exactly one political party. It is impossible to tell which political party any delegate belongs to; in particular, you will be summarily ejected from the convention if you ask. However, you can determine whether any pair of delegates belong to the *same* party or not simply by introducing them to each other—members of the same party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.

Suppose more than half of the delegates belong to the same political party. Describe and analyze an efficient algorithm that identifies all members of this majority party.

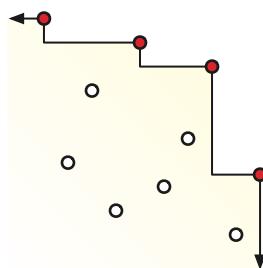
3. Recall that a *tree* is a connected undirected graph with no cycles. **Prove** that in any tree, the number of nodes is exactly one more than the number of edges.
4. Next spring break, you and your friends decide to take a road trip, but before you leave, you decide to figure out *exactly* how much money to bring for gasoline. Suppose you compile a list of all gas stations along your planned route, containing the following information:

- A sorted array  $Dist[0..n]$ , where  $Dist[0] = 0$  and  $Dist[i]$  is the number of miles from the beginning of your route to the  $i$ th gas station. Your route ends at the  $n$ th gas station.
- A second array  $Price[1..n]$ , where  $Price[i]$  is the price of one gallon of gasoline at the  $i$ th gas station. (Unlike in real life, these prices do not change over time.)

You start the trip with a full tank of gas. Whenever you buy gas, you must completely fill your tank. Your car holds exactly 10 gallons of gas and travels exactly 25 miles per gallon; thus, starting with a full tank, you can travel exactly 250 miles before your car dies. Finally,  $Dist[i+1] < Dist[i] + 250$  for every index  $i$ , so the trip is possible.

Describe and analyze an algorithm to determine the minimum amount of money you must spend on gasoline to guarantee that you can drive the entire route.

5. Suppose you are given a set  $P$  of  $n$  points in the plane. A point  $p \in P$  is *maximal* in  $P$  if no other point in  $P$  is both above and to the right of  $p$ . Intuitively, the maximal points define a “staircase” with all the other points of  $P$  below it.



A set of ten points, four of which are maximal.

Describe and analyze an algorithm to compute the number of maximal points in  $P$  in  $O(n \log n)$  time. For example, given the ten points shown above, your algorithm should return the integer 4.

This exam lasts 120 minutes.

**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

1. Each of these ten questions has one of the following five answers:

$$\text{A: } \Theta(1) \quad \text{B: } \Theta(\log n) \quad \text{C: } \Theta(n) \quad \text{D: } \Theta(n \log n) \quad \text{E: } \Theta(n^2)$$

- (a) What is  $\frac{n^5 - 3n^3 - 5n + 4}{3n^4 - 2n^2 + n - 7}$ ?
- (b) What is  $\sum_{i=1}^n \frac{n}{i}$ ?
- (c) What is  $\sum_{i=1}^n \frac{i}{n}$ ?
- (d) How many bits are required to write the integer  $10^n$  in binary?
- (e) What is the solution to the recurrence  $E(n) = E(n/3) + E(n/4) + E(n/5) + n/6$ ?
- (f) What is the solution to the recurrence  $F(n) = 16F(n/4 + 2) + n$ ?
- (g) What is the solution to the recurrence  $G(n) = G(n/2) + 2G(n/4) + n$ ?
- (h) The *total path length* of a binary tree is the sum of the depths of all nodes. What is the total path length of a perfectly balanced  $n$ -node binary tree?
- (i) Consider the following recursive function, defined in terms of two fixed arrays  $A[1..n]$  and  $B[1..n]$ :

$$WTF(i, j) = \begin{cases} 0 & \text{if } i > j \\ \max \left\{ \begin{array}{l} (A[i] - B[j])^2 + WTF(i+1, j-1) \\ A[i]^2 + WTF(i+1, j) \\ B[i]^2 + WTF(i, j-1) \end{array} \right\} & \text{otherwise} \end{cases}$$

How long does it take to compute  $WTF(1, n)$  using dynamic programming?

- (j) Voyager 1 recently became the first made-made object to reach interstellar space. Currently the spacecraft is about 18 billion kilometers (roughly 60,000 light seconds) from Earth, traveling outward at approximately 17 kilometers per second (approximately  $1/18000$  of the speed of light). Voyager carries a golden record containing over 100 digital images and approximately one hour of sound recordings. In digital form, the recording would require about 1 gigabyte. Voyager can transmit data back to Earth at approximately 1400 bits per second. Suppose the engineers at JPL sent instructions to Voyager 1 to send the complete contents of the Golden Record back to Earth; how many seconds would they have to wait to receive the entire record?

2. Suppose we are given an array  $A[0..n+1]$  with fencepost values  $A[0] = A[n+1] = -\infty$ . We say that an element  $A[x]$  is a *local maximum* if it is less than or equal to its neighbors, or more formally, if  $A[x-1] \leq A[x]$  and  $A[x] \geq A[x+1]$ . For example, there are five local maxima in the following array:

$-\infty$	6	(7)	2	1	3	(7)	5	4	(9)	(9)	3	4	(8)	6	$-\infty$
-----------	---	-----	---	---	---	-----	---	---	-----	-----	---	---	-----	---	-----------

We can obviously find a local maximum in  $O(n)$  time by scanning through the array. Describe and analyze an algorithm that returns the index of one local maximum in  $O(\log n)$  time. [Hint: With the given boundary conditions, the array **must** have at least one local maximum. Why?]

3. **Prove** that in any binary tree, the number of nodes with no children (leaves) is exactly one more than the number of nodes with two children. (Remember that a binary tree can have nodes with only one child.)

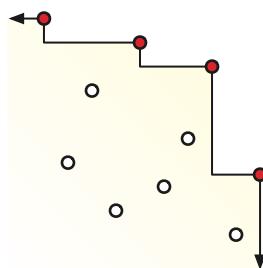
4. A string  $x$  is a *supersequence* of a string  $y$  if we can obtain  $x$  by inserting zero or more letters into  $y$ , or equivalently, if  $y$  is a subsequence of  $x$ . For example, the string DYNAMICPROGRAMMING is a supersequence of the string DAMPRAG.

A *palindrome* is any string that is exactly the same as its reversal, like I, DAD, HANNAH, AIBOOPHOBIA (fear of palindromes), or the empty string.

Describe and analyze an algorithm to find the length of the shortest supersequence of a given string that is also a palindrome.

For example, the 11-letter string EHECADACEHE is the shortest palindrome supersequence of HEADACHE, so given the string HEADACHE as input, your algorithm should output the number 11.

5. Suppose you are given a set  $P$  of  $n$  points in the plane. A point  $p \in P$  is *maximal* in  $P$  if no other point in  $P$  is both above and to the right of  $p$ . Intuitively, the maximal points define a “staircase” with all the other points of  $P$  below it.



A set of ten points, four of which are maximal.

Describe and analyze an algorithm to compute the number of maximal points in  $P$  in  $O(n \log n)$  time. For example, given the ten points shown above, your algorithm should return the integer 4.

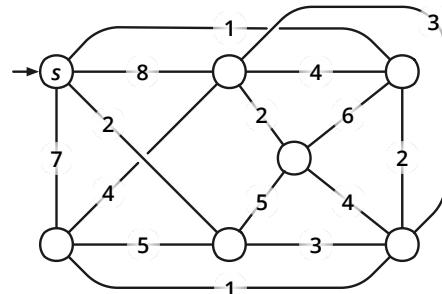
This exam lasts 120 minutes.

**Write your answers in the separate answer booklet.**

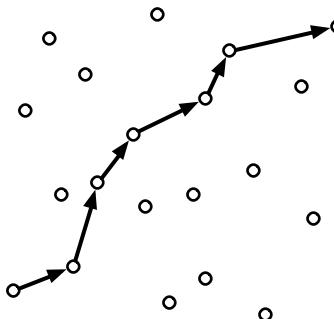
Please return this question sheet and your cheat sheet with your answers.

1. **Clearly** indicate the following spanning trees in the weighted graph pictured below. Some of these subproblems have more than one correct answer.

- (a) A depth-first spanning tree rooted at  $s$
- (b) A breadth-first spanning tree rooted at  $s$
- (c) A shortest-path tree rooted at  $s$
- (d) A minimum spanning tree
- (e) A *maximum* spanning tree



2. A **polygonal path** is a sequence of line segments joined end-to-end; the endpoints of these line segments are called the **vertices** of the path. The **length** of a polygonal path is the sum of the lengths of its segments. A polygonal path with vertices  $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)$  is **monotonically increasing** if  $x_i < x_{i+1}$  and  $y_i < y_{i+1}$  for every index  $i$ —informally, each vertex of the path is above and to the right of its predecessor.



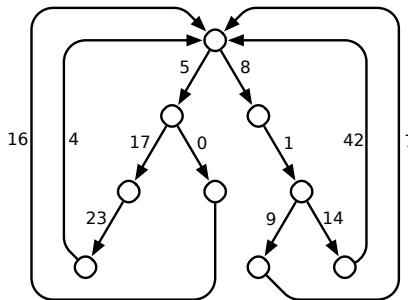
A monotonically increasing polygonal path with seven vertices through a set of points

Suppose you are given a set  $S$  of  $n$  points in the plane, represented as two arrays  $X[1..n]$  and  $Y[1..n]$ . Describe and analyze an algorithm to compute the length of the maximum-length monotonically increasing path with vertices in  $S$ . Assume you have a subroutine  $\text{LENGTH}(x, y, x', y')$  that returns the length of the segment from  $(x, y)$  to  $(x', y')$ .

3. Suppose you are maintaining a circular array  $X[0..n-1]$  of counters, each taking a value from the set  $\{0, 1, 2\}$ . The following algorithm increments one of the counters; if the counter overflows, the algorithm resets it 0 and recursively increments its two neighbors.

```
INCREMENT( $i$ ):
   $X[i] \leftarrow X[i] + 1$ 
  if  $X[i] = 3$ 
     $X[i] \leftarrow 0$ 
    INCREMENT( $((i - 1) \bmod n)$ )
    INCREMENT( $((i + 1) \bmod n)$ )
```

- (a) Suppose  $n = 5$  and  $X = [2, 2, 2, 2, 2]$ . What does  $X$  contain after we call  $\text{INCREMENT}(3)$ ?  
 (b) Suppose all counters are initially 0. **Prove** that  $\text{INCREMENT}$  runs in  $O(1)$  amortized time.  
 4. A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has non-negative weight.



A looped tree.

- (a) How much time would Dijkstra's algorithm require to compute the shortest path from an arbitrary vertex  $s$  to another arbitrary vertex  $t$ , in a looped tree with  $n$  vertices?  
 (b) Describe and analyze a faster algorithm. Your algorithm should compute the actual shortest path, not just its length.
5. Consider the following algorithm for finding the smallest element in an unsorted array:

```
RANDOMMIN( $A[1..n]$ ):
   $min \leftarrow \infty$ 
  for  $i \leftarrow 1$  to  $n$  in random order
    if  $A[i] < min$ 
       $min \leftarrow A[i]$  (*)
```

Assume the elements of  $A$  are all distinct.

- (a) In the worst case, how many times does  $\text{RANDOMMIN}$  execute line (\*)?  
 (b) What is the probability that line (\*) is executed during the *last* iteration of the for loop?  
 (c) What is the *exact* expected number of executions of line (\*)?

This exam lasts 180 minutes.

**Write your answers in the separate answer booklet.**

Please return this question handout and your cheat sheets with your answers.

- Suppose you are given a sorted array of  $n$  distinct numbers that has been *rotated  $k$  steps*, for some **unknown** integer  $k$  between 1 and  $n - 1$ . That is, you are given an array  $A[1 \dots n]$  such that the prefix  $A[1 \dots k]$  is sorted in increasing order, the suffix  $A[k + 1 \dots n]$  is sorted in increasing order, and  $A[n] < A[1]$ . For example, you might be given the following 16-element array (where  $k = 10$ ):

9	13	16	18	19	23	28	31	37	42	-4	0	2	5	7	8
---	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---

Describe and analyze an algorithm to determine if the given array contains a given number  $x$ . For example, given the previous array and the number 17 as input, your algorithm should return FALSE. The index  $k$  is **NOT** part of the input.

- You are hired as a cyclist for the Giggle Highway View project, which will provide street-level images along the entire US national highway system. As a pilot project, you are asked to ride the Giggle Highway-View Fixed-Gear Carbon-Fiber Bicycle from “the Giggleplex” in Portland, Oregon to “Giggleburg” in Williamsburg, Brooklyn, New York.

You are a hopeless caffeine addict, but like most Giggle employees you are also a coffee snob; you only drink independently roasted organic shade-grown single-origin espresso. After each espresso shot, you can bike up to  $L$  miles before suffering a caffeine-withdrawal migraine.

Giggle helpfully provides you with a map of the United States, in the form of an undirected graph  $G$ , whose vertices represent coffee shops that sell independently roasted organic shade-grown single-origin espresso, and whose edges represent highway connections between them. Each edge  $e$  is labeled with the length  $\ell(e)$  of the corresponding stretch of highway. Naturally, there are espresso stands at both Giggle offices, represented by two specific vertices  $s$  and  $t$  in the graph  $G$ .

- Describe and analyze an algorithm to determine whether it is possible to bike from the Giggleplex to Giggleburg without suffering a caffeine-withdrawal migraine.
- You discover that by wearing a more expensive fedora, you can increase the distance  $L$  that you can bike between espresso shots. Describe and analyze an algorithm to find the minimum value of  $L$  that allows you to bike from the Giggleplex to Giggleburg without suffering a caffeine-withdrawal migraine.

3. Suppose you are given a collection of up-trees representing a partition of the set  $\{1, 2, \dots, n\}$  into subsets. **You have no idea how these trees were constructed.** You are also given an array  $node[1..n]$ , where  $node[i]$  is a pointer to the up-tree node containing element  $i$ . Your task is to create a new array  $label[1..n]$  using the following algorithm:

```
LABELEVERYTHING:
for i ← 1 to n
    label[i] ← FIND(node[i])
```

Recall that there are two natural ways to implement FIND: simple pointer-chasing and pointer-chasing with path compression. Pseudocode for both methods is shown below.

```
FIND(x):
    while x ≠ parent(x)
        x ← parent(x)
    return x
```

Without path compression

```
FIND(x):
    if x ≠ parent(x)
        parent(x) ← FIND(parent(x))
    return parent(x)
```

With path compression

- (a) What is the worst-case running time of LABELEVERYTHING if we implement FIND *without* path compression?
- (b) **Prove** that if we implement FIND using path compression, LABELEVERYTHING runs in  $O(n)$  time in the worst case.

4. Congratulations! You have successfully conquered Camelot, transforming the former battle-scarred kingdom with an anarcho-syndicalist commune, where citizens take turns to act as a sort of executive-officer-for-the-week, but with all the decisions of that officer ratified at a special bi-weekly meeting, by a simple majority in the case of purely internal affairs, but by a two-thirds majority, in the case of more major....

As a final symbolic act, you order the Round Table (surprisingly, an actual circular table) to be split into pizza-like wedges and distributed to the citizens of Camelot as trophies. Each citizen has submitted a request for an angular wedge of the table, specified by two angles—for example, Sir Robin the Brave might request the wedge from  $23.17^\circ$  to  $42^\circ$ . Each citizen will be happy if and only if they receive *precisely* the wedge that they requested. Unfortunately, some of these ranges overlap, so satisfying *all* the citizens' requests is simply impossible. Welcome to politics.

Describe and analyze an algorithm to find the maximum number of requests that can be satisfied.

5. The NSA has established several monitoring stations around the country, each one conveniently hidden in the back of a Starbucks. Each station can monitor up to 42 cell-phone towers, but can only monitor cell-phone towers within a 20-mile radius. To ensure that every cell-phone call is recorded even if some stations malfunction, the NSA requires each cell-phone tower to be monitored by at least 3 different stations.

Suppose you know that there are  $n$  cell-phone towers and  $m$  monitoring stations, and you are given a function  $\text{DISTANCE}(i, j)$  that returns the distance between the  $i$ th tower and the  $j$ th station in  $O(1)$  time. Describe and analyze an algorithm that either computes a valid assignment of cell-phone towers to monitoring stations, or reports correctly that there is no such assignment (in which case the NSA will build another Starbucks).

6. Consider the following closely related problems:

- HAMILTONIANPATH: Given an undirected graph  $G$ , determine whether  $G$  contains a **path** that visits every vertex of  $G$  exactly once.
- HAMILTONIANCYCLE: Given an undirected graph  $G$ , determine whether  $G$  contains a **cycle** that visits every vertex of  $G$  exactly once.

Describe a polynomial-time reduction from HAMILTONIANPATH to HAMILTONIANCYCLE. **Prove** your reduction is correct. [Hint: A polynomial-time reduction is allowed to call the black-box subroutine more than once.]

7. An array  $X[1..n]$  of distinct integers is **wobbly** if it alternates between increasing and decreasing:  $X[i] < X[i + 1]$  for every odd index  $i$ , and  $X[i] > X[i + 1]$  for every even index  $i$ . For example, the following 16-element array is wobbly:

12	13	0	16	13	31	5	7	-1	23	8	10	-4	37	17	42
----	----	---	----	----	----	---	---	----	----	---	----	----	----	----	----

Describe and analyze an algorithm that permutes the elements of a given array to make the array wobbly.

**You may use the following algorithms as black boxes:**

**RANDOM( $k$ ):** Given any positive integer  $k$ , return an integer chosen independently and uniformly at random from the set  $\{1, 2, \dots, k\}$  in  $O(1)$  time.

**ORLINMAXFLOW( $V, E, c, s, t$ ):** Given a directed graph  $G = (V, E)$ , a capacity function  $c: E \rightarrow \mathbb{R}^+$ , and vertices  $s$  and  $t$ , return a maximum  $(s, t)$ -flow in  $G$  in  $O(VE)$  time. If the capacities are integral, so is the returned maximum flow.

Any other algorithm that we described in class.

**You may assume the following problems are NP-hard:**

**CIRCUITSAT:** Given a boolean circuit, are there any input values that make the circuit output TRUE?

**PLANAR CIRCUITSAT:** Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output TRUE?

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

**MAXINDEPENDENTSET:** Given an undirected graph  $G$ , what is the size of the largest subset of vertices in  $G$  that have no edges among them?

**MAXCLIQUE:** Given an undirected graph  $G$ , what is the size of the largest complete subgraph of  $G$ ?

**MINVERTEXCOVER:** Given an undirected graph  $G$ , what is the size of the smallest subset of vertices that touch every edge in  $G$ ?

**MINSETCOVER:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subcollection whose union is  $S$ ?

**MINHITTINGSET:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subset of  $S$  that intersects every subset  $S_i$ ?

**3COLOR:** Given an undirected graph  $G$ , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

**HAMILTONIANCYCLE:** Given a graph  $G$ , is there a cycle in  $G$  that visits every vertex exactly once?

**HAMILTONIANPATH:** Given a graph  $G$ , is there a path in  $G$  that visits every vertex exactly once?

**TRAVELINGSALESMAN:** Given a graph  $G$  with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in  $G$ ?

**STEINERTREE:** Given an undirected graph  $G$  with some of the vertices marked, what is the minimum number of edges in a subtree of  $G$  that contains every marked vertex?

**SUBSETSUM:** Given a set  $X$  of positive integers and an integer  $k$ , does  $X$  have a subset whose elements sum to  $k$ ?

**PARTITION:** Given a set  $X$  of positive integers, can  $X$  be partitioned into two subsets with the same sum?

**3PARTITION:** Given a set  $X$  of  $3n$  positive integers, can  $X$  be partitioned into  $n$  three-element subsets, all with the same sum?

**DRAUGHTS:** Given an  $n \times n$  international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

**DOGE:** Such N. Many P. Wow.

This exam lasts 180 minutes.

**Write your answers in the separate answer booklet.**

Please return this question handout and your cheat sheets with your answers.

- Suppose you are given a sorted array of  $n$  distinct numbers that has been *rotated  $k$  steps*, for some **unknown** integer  $k$  between 1 and  $n - 1$ . That is, you are given an array  $A[1 \dots n]$  such that the prefix  $A[1 \dots k]$  is sorted in increasing order, the suffix  $A[k + 1 \dots n]$  is sorted in increasing order, and  $A[n] < A[1]$ . Describe and analyze an algorithm to compute the unknown integer  $k$ .

For example, given the following array as input, your algorithm should output the integer 10.

9	13	16	18	19	23	28	31	37	42	-4	0	2	5	7	8
---	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---

- You are hired as a cyclist for the Giggle Highway View project, which will provide street-level images along the entire US national highway system. As a pilot project, you are asked to ride the Giggle Highway-View Fixed-Gear Carbon-Fiber Bicycle from “the Giggleplex” in Portland, Oregon to “Giggleburg” in Williamsburg, Brooklyn, New York.

You are a hopeless caffeine addict, but like most Giggle employees you are also a coffee snob; you only drink independently roasted organic shade-grown single-origin espresso. After each espresso shot, you can bike up to  $L$  miles before suffering a caffeine-withdrawal migraine.

Giggle helpfully provides you with a map of the United States, in the form of an undirected graph  $G$ , whose vertices represent coffee shops that sell independently roasted organic shade-grown single-origin espresso, and whose edges represent highway connections between them. Each edge  $e$  is labeled with the length  $\ell(e)$  of the corresponding stretch of highway. Naturally, there are espresso stands at both Giggle offices, represented by two specific vertices  $s$  and  $t$  in the graph  $G$ .

- Describe and analyze an algorithm to determine whether it is possible to bike from the Giggleplex to Giggleburg without suffering a caffeine-withdrawal migraine.
- When you report to your supervisor (whom Giggle recently hired away from competitor Yippee!) that the ride is impossible, she demands to look at your map. “Oh, I see the problem; there are no *Starbucks* on this map!” As you look on in horror, she hands you an updated graph  $G'$  that includes a vertex for every Starbucks location in the United States, helpfully marked in Starbucks Green (Pantone® 3425 C).

Describe and analyze an algorithm to find the minimum number of Starbucks locations you must visit to bike from the Giggleplex to Giggleburg without suffering a caffeine-withdrawal migraine. More formally, your algorithm should find the minimum number of green vertices on any path in  $G'$  from  $s$  to  $t$  that uses only edges of length at most  $L$ .

3. Suppose you are given a collection of up-trees representing a partition of the set  $\{1, 2, \dots, n\}$  into subsets. **You have no idea how these trees were constructed.** You are also given an array  $node[1..n]$ , where  $node[i]$  is a pointer to the up-tree node containing element  $i$ . Your task is to create a new array  $label[1..n]$  using the following algorithm:

```
LABELEVERYTHING:
for i ← 1 to n
    label[i] ← FIND(node[i])
```

Recall that there are two natural ways to implement FIND: simple pointer-chasing and pointer-chasing with path compression. Pseudocode for both methods is shown below.

```
FIND(x):
    while x ≠ parent(x)
        x ← parent(x)
    return x
```

Without path compression

```
FIND(x):
    if x ≠ parent(x)
        parent(x) ← FIND(parent(x))
    return parent(x)
```

With path compression

- (a) What is the worst-case running time of LABELEVERYTHING if we implement FIND *without* path compression?
- (b) **Prove** that if we implement FIND using path compression, LABELEVERYTHING runs in  $O(n)$  time in the worst case.

4. Congratulations! You have successfully conquered Camelot, transforming the former battle-scarred kingdom with an anarcho-syndicalist commune, where citizens take turns to act as a sort of executive-officer-for-the-week, but with all the decisions of that officer ratified at a special bi-weekly meeting, by a simple majority in the case of purely internal affairs, but by a two-thirds majority, in the case of more major....

As a final symbolic act, you order the Round Table (surprisingly, an actual circular table) to be split into pizza-like wedges and distributed to the citizens of Camelot as trophies. Each citizen has submitted a request for an angular wedge of the table, specified by two angles—for example, Sir Robin the Brave might request the wedge from  $17^\circ$  to  $42^\circ$ . Each citizen will be happy if and only if they receive *precisely* the wedge that they requested. Unfortunately, some of these ranges overlap, so satisfying *all* the citizens' requests is simply impossible. Welcome to politics.

Describe and analyze an algorithm to find the maximum number of requests that can be satisfied.

5. The NSA has established several monitoring stations around the country, each one conveniently hidden in the back of a Starbucks. Each station can monitor up to 42 cell-phone towers, but can only monitor cell-phone towers within a 20-mile radius. To ensure that every cell-phone call is recorded even if some stations malfunction, the NSA requires each cell-phone tower to be monitored by at least 3 different stations.

Suppose you know that there are  $n$  cell-phone towers and  $m$  monitoring stations, and you are given a function  $\text{DISTANCE}(i, j)$  that returns the distance between the  $i$ th tower and the  $j$ th station in  $O(1)$  time. Describe and analyze an algorithm that either computes a valid assignment of cell-phone towers to monitoring stations, or reports correctly that there is no such assignment (in which case the NSA will build another Starbucks).

6. Consider the following closely related problems:

- HAMILTONIANPATH: Given an undirected graph  $G$ , determine whether  $G$  contains a **path** that visits every vertex of  $G$  exactly once.
- HAMILTONIANCYCLE: Given an undirected graph  $G$ , determine whether  $G$  contains a **cycle** that visits every vertex of  $G$  exactly once.

Describe a polynomial-time reduction from HAMILTONIANCYCLE to HAMILTONIANPATH. **Prove** your reduction is correct. [Hint: A polynomial-time reduction is allowed to call the black-box subroutine more than once.]

7. An array  $X[1..n]$  of distinct integers is **wobbly** if it alternates between increasing and decreasing:  $X[i] < X[i + 1]$  for every odd index  $i$ , and  $X[i] > X[i + 1]$  for every even index  $i$ . For example, the following 16-element array is wobbly:

12	13	0	16	13	31	5	7	-1	23	8	10	-4	37	17	42
----	----	---	----	----	----	---	---	----	----	---	----	----	----	----	----

Describe and analyze an algorithm that permutes the elements of a given array to make the array wobbly.

**You may use the following algorithms as black boxes:**

**RANDOM( $k$ ):** Given any positive integer  $k$ , return an integer chosen independently and uniformly at random from the set  $\{1, 2, \dots, k\}$  in  $O(1)$  time.

**ORLINMAXFLOW( $V, E, c, s, t$ ):** Given a directed graph  $G = (V, E)$ , a capacity function  $c: E \rightarrow \mathbb{R}^+$ , and vertices  $s$  and  $t$ , return a maximum  $(s, t)$ -flow in  $G$  in  $O(VE)$  time. If the capacities are integral, so is the returned maximum flow.

Any other algorithm that we described in class.

**You may assume the following problems are NP-hard:**

**CIRCUITSAT:** Given a boolean circuit, are there any input values that make the circuit output TRUE?

**PLANAR CIRCUITSAT:** Given a boolean circuit drawn in the plane so that no two wires cross, are there any input values that make the circuit output TRUE?

**3SAT:** Given a boolean formula in conjunctive normal form, with exactly three literals per clause, does the formula have a satisfying assignment?

**MAXINDEPENDENTSET:** Given an undirected graph  $G$ , what is the size of the largest subset of vertices in  $G$  that have no edges among them?

**MAXCLIQUE:** Given an undirected graph  $G$ , what is the size of the largest complete subgraph of  $G$ ?

**MINVERTEXCOVER:** Given an undirected graph  $G$ , what is the size of the smallest subset of vertices that touch every edge in  $G$ ?

**MINSETCOVER:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subcollection whose union is  $S$ ?

**MINHITTINGSET:** Given a collection of subsets  $S_1, S_2, \dots, S_m$  of a set  $S$ , what is the size of the smallest subset of  $S$  that intersects every subset  $S_i$ ?

**3COLOR:** Given an undirected graph  $G$ , can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

**HAMILTONIANCYCLE:** Given a graph  $G$ , is there a cycle in  $G$  that visits every vertex exactly once?

**HAMILTONIANPATH:** Given a graph  $G$ , is there a path in  $G$  that visits every vertex exactly once?

**TRAVELINGSALESMAN:** Given a graph  $G$  with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in  $G$ ?

**STEINERTREE:** Given an undirected graph  $G$  with some of the vertices marked, what is the minimum number of edges in a subtree of  $G$  that contains every marked vertex?

**SUBSETSUM:** Given a set  $X$  of positive integers and an integer  $k$ , does  $X$  have a subset whose elements sum to  $k$ ?

**PARTITION:** Given a set  $X$  of positive integers, can  $X$  be partitioned into two subsets with the same sum?

**3PARTITION:** Given a set  $X$  of  $3n$  positive integers, can  $X$  be partitioned into  $n$  three-element subsets, all with the same sum?

**DRAUGHTS:** Given an  $n \times n$  international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

**DOGE:** Such N. Many P. Wow.

# CS 373: Combinatorial Algorithms, Spring 1999

<http://www-courses.cs.uiuc.edu/cs373>

**Homework 0 (due January 26, 1999 by the beginning of class)**

Name:	
Net ID:	Alias:

Neatly print your name (first name first, with no comma), your network ID, and a short alias into the boxes above. Do not *sign* your name. Do not write your Social Security number. Staple this sheet of paper to the top of your homework. Grades will be listed on the course web site by alias, so your alias should not resemble your name (or your Net ID). If you do not give yourself an alias, you will be stuck with one we give you, no matter how much you hate it.

**Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit.** (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

This homework tests your familiarity with the prerequisite material from CS 225 and CS 273 (and *their* prerequisites)—many of these problems appeared on homeworks and/or exams in those classes—primarily to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.**

## Undergrad/.75U Grad/1U Grad Problems

### ▶1. [173/273]

- (a) Prove that any positive integer can be written as the sum of distinct powers of 2. (For example:  $42 = 2^5 + 2^3 + 2^1$ ,  $25 = 2^4 + 2^3 + 2^0$ ,  $17 = 2^4 + 2^0$ .)
- (b) Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers—if  $F_n$  appears in the sum, then neither  $F_{n+1}$  nor  $F_{n-1}$  will. (For example:  $42 = F_9 + F_6$ ,  $25 = F_8 + F_4 + F_2$ ,  $17 = F_7 + F_4 + F_2$ .)
- (c) Prove that *any* integer can be written in the form  $\sum_i \pm 3^i$ , where the exponents  $i$  are distinct non-negative integers. (For example:  $42 = 3^4 - 3^3 - 3^2 - 3^1$ ,  $25 = 3^3 - 3^1 + 3^0$ ,  $17 = 3^3 - 3^2 - 3^0$ .)

### ▶2. [225/273] Sort the following functions from asymptotically smallest to largest, indicating ties if there are any:

$n$ ,  $\lg n$ ,  $\lg \lg^* n$ ,  $\lg^* \lg n$ ,  $\lg^* n$ ,  $n \lg n$ ,  $\lg(n \lg n)$ ,  $n^{n/\lg n}$ ,  $n^{\lg n}$ ,  $(\lg n)^n$ ,  $(\lg n)^{\lg n}$ ,  $2^{\sqrt{\lg n \lg \lg n}}$ ,  $2^n$ ,  $n^{\lg \lg n}$ ,  $\sqrt[1000]{n}$ ,  $(1 + \frac{1}{1000})^n$ ,  $(1 - \frac{1}{1000})^n$ ,  $\lg^{1000} n$ ,  $\lg^{(1000)} n$ ,  $\log_{1000} n$ ,  $\lg^n 1000$ ,  $1$ .

[To simplify notation, write  $f(n) \ll g(n)$  to mean  $f(n) = o(g(n))$  and  $f(n) \equiv g(n)$  to mean  $f(n) = \Theta(g(n))$ . For example, the functions  $n^2$ ,  $n$ ,  $\binom{n}{2}$ ,  $n^3$  could be sorted as follows:  $n \ll n^2 \equiv \binom{n}{2} \ll n^3$ .]

3. [273/225] Solve the following recurrences. State tight asymptotic bounds for each function in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ . You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway just for practice. Assume reasonable (nontrivial) base cases. Extra credit will be given for more exact solutions.

- (a)  $A(n) = A(n/2) + n$
- (b)  $B(n) = 2B(n/2) + n$
- (c)  $C(n) = 3C(n/2) + n$
- (d)  $D(n) = \max_{n/3 < k < 2n/3} (D(k) + D(n - k) + n)$
- (e)  $E(n) = \min_{0 < k < n} (E(k) + E(n - k) + 1)$
- (f)  $F(n) = 4F(\lfloor n/2 \rfloor + 5) + n$
- (g)  $G(n) = G(n - 1) + 1/n$
- \*(h)  $H(n) = H(n/2) + H(n/4) + H(n/6) + H(n/12) + n$  [Hint:  $\frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \frac{1}{12} = 1$ .]
- \*(i)  $I(n) = 2I(n/2) + n/\lg n$
- \*(j)  $J(n) = \frac{J(n - 1)}{J(n - 2)}$

►4. [273] Alice and Bob each have a fair  $n$ -sided die. Alice rolls her die once. Bob then repeatedly throws his die until the number he rolls is at least as big as the number Alice rolled. Each time Bob rolls, he pays Alice \$1. (For example, if Alice rolls a 5, and Bob rolls a 4, then a 3, then a 1, then a 5, the game ends and Alice gets \$4. If Alice rolls a 1, then no matter what Bob rolls, the game will end immediately, and Alice will get \$1.)

Exactly how much money does Alice expect to win at this game? Prove that your answer is correct. (If you have to appeal to “intuition” or “common sense”, your answer is probably wrong.)

►5. [225] George has a 26-node binary tree, with each node labeled by a unique letter of the alphabet. The preorder and postorder sequences of nodes are as follows:

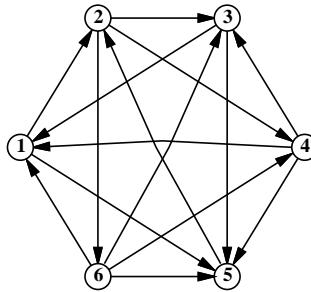
preorder: M N H C R S K W T G D X I Y A J P O E Z V B U L Q F  
 postorder: C W T K S G R H D N A O E P J Y Z I B Q L F U V X M

Draw George’s binary tree.

### Only 1U Grad Problems

►\*1. [225/273] A *tournament* is a directed graph with exactly one edge between every pair of vertices. (Think of the nodes as players in a round-robin tournament, where each edge points from the winner to the loser.) A *Hamiltonian path* is a sequence of directed edges, joined end to end, that visits every vertex exactly once.

Prove that every tournament contains at least one Hamiltonian path.



A six-vertex tournament containing the Hamiltonian path  $6 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1$ .

### Practice Problems

1. [173/273] Recall the standard recursive definition of the Fibonacci numbers:  $F_0 = 0$ ,  $F_1 = 1$ , and  $F_n = F_{n-1} + F_{n-2}$  for all  $n \geq 2$ . Prove the following identities for all positive integers  $n$  and  $m$ .

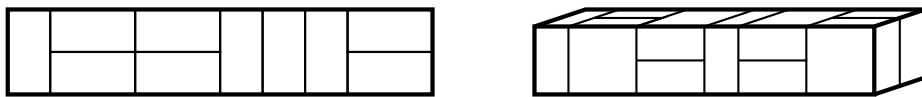
- (a)  $F_n$  is even if and only if  $n$  is divisible by 3.
- (b)  $\sum_{i=0}^n F_i = F_{n+2} - 1$
- (c)  $F_n^2 - F_{n+1}F_{n-1} = (-1)^{n+1}$
- ★(d) If  $n$  is an integer multiple of  $m$ , then  $F_n$  is an integer multiple of  $F_m$ .

2. [225/273]

- (a) Prove that  $2^{\lceil \lg n \rceil + \lfloor \lg n \rfloor} / n = \Theta(n)$ .
- (b) Is  $2^{\lfloor \lg n \rfloor} = \Theta(2^{\lceil \lg n \rceil})$ ? Justify your answer.
- (c) Is  $2^{2^{\lfloor \lg \lg n \rfloor}} = \Theta(2^{2^{\lceil \lg \lg n \rceil}})$ ? Justify your answer.

3. [273]

- (a) A *domino* is a  $2 \times 1$  or  $1 \times 2$  rectangle. How many different ways are there to completely fill a  $2 \times n$  rectangle with  $n$  dominos?
- (b) A *slab* is a three-dimensional box with dimensions  $1 \times 2 \times 2$ ,  $2 \times 1 \times 2$ , or  $2 \times 2 \times 1$ . How many different ways are there to fill a  $2 \times 2 \times n$  box with  $n$  slabs? Set up a recurrence relation and give an *exact* closed-form solution.



A  $2 \times 10$  rectangle filled with ten dominos, and a  $2 \times 2 \times 10$  box filled with ten slabs.

4. [273] Penn and Teller have a special deck of fifty-two cards, with no face cards and nothing but clubs—the ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ..., 52 of clubs. (They're big cards.) Penn shuffles the deck until each each of the  $52!$  possible orderings of the cards is equally likely. He then takes cards one at a time from the top of the deck and gives them to Teller, stopping as soon as he gives Teller the three of clubs.

- (a) On average, how many cards does Penn give Teller?
- (b) On average, what is the smallest-numbered card that Penn gives Teller?
- \*(c) On average, what is the largest-numbered card that Penn gives Teller?

Prove that your answers are correct. (If you have to appeal to “intuition” or “common sense”, your answers are probably wrong.) [Hint: Solve for an  $n$ -card deck, and then set  $n$  to 52.]

5. [273/225] Prove that for any nonnegative parameters  $a$  and  $b$ , the following algorithms terminate and produce identical output.

SLOWEUCLID( $a, b$ ) :

if  $b > a$   
    return SLOWEUCLID( $b, a$ )  
else if  $b == 0$   
    return  $a$   
else  
    return SLOWEUCLID( $a, b - a$ )

FASTEUCLID( $a, b$ ) :

if  $b == 0$   
    return  $a$   
else  
    return FASTEUCLID( $b, a \bmod b$ )

# CS 373: Combinatorial Algorithms, Spring 1999

<http://www-courses.cs.uiuc.edu/~cs373>

## Homework 1 (due February 9, 1999 by noon)

Name:	
Net ID:	Alias:

**Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit.** (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

**Note:** When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
2. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
3. Justify the correctness of your algorithm, including termination if that is not obvious.
4. Analyze the time and space complexity of your algorithm.

### Undergrad/.75U Grad/1U Grad Problems

- ▶1. Consider the following sorting algorithm:

```

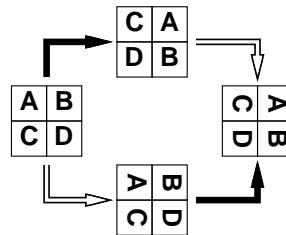
STUPIDSORT( $A[0..n - 1]$ ) :
  if  $n = 2$  and  $A[0] > A[1]$ 
    swap  $A[0] \leftrightarrow A[1]$ 
  else if  $n > 2$ 
     $m = \lceil 2n/3 \rceil$ 
    STUPIDSORT( $A[0..m - 1]$ )
    STUPIDSORT( $A[n - m..n - 1]$ )
    STUPIDSORT( $A[0..m - 1]$ )
  
```

- (a) Prove that STUPIDSORT actually sorts its input.
- (b) Would the algorithm still sort correctly if we replaced  $m = \lceil 2n/3 \rceil$  with  $m = \lfloor 2n/3 \rfloor$ ? Justify your answer.
- (c) State a recurrence (including the base case(s)) for the number of comparisons executed by STUPIDSORT.

- (d) Solve the recurrence, and prove that your solution is correct. [Hint: Ignore the ceiling.] Does the algorithm deserve its name?
  - \*(e) Show that the number of *swaps* executed by STUPID SORT is at most  $\binom{n}{2}$ .

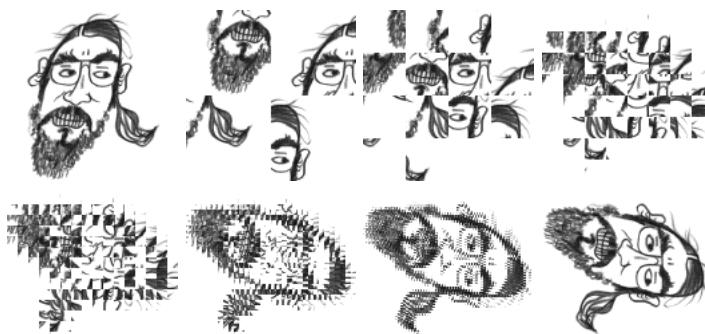
► 2. Some graphics hardware includes support for an operation called *blit*, or **block transfer**, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

Suppose we want to rotate an  $n \times n$  pixelmap 90° clockwise. One way to do this is to split the pixelmap into four  $n/2 \times n/2$  blocks, move each block to its proper position using a sequence of five blits, and then recursively rotate each block. Alternately, we can first recursively rotate the blocks and blit them into place afterwards.



Two algorithms for rotating a pixelmap.  
Black arrows indicate blitting the blocks into place.  
White arrows indicate recursively rotating the blocks.

The following sequence of pictures shows the first algorithm (blit then recurse) in action.



In the following questions, assume  $n$  is a power of two.

- (a) Prove that both versions of the algorithm are correct.
  - (b) Exactly how many blits does the algorithm perform?
  - (c) What is the algorithm's running time if a  $k \times k$  blit takes  $O(k^2)$  time?
  - (d) What if a  $k \times k$  blit takes only  $O(k)$  time?

## ►3. Dynamic Programming: The Company Party

A company is planning a party for its employees. The organizers of the party want it to be a fun party, and so have assigned a ‘fun’ rating to every employee. The employees are organized into a strict hierarchy, i.e. a tree rooted at the president. There is one restriction, though, on the guest list to the party: both an employee and their immediate supervisor (parent in the tree) cannot both attend the party (because that would be no fun at all). Give an algorithm that makes a guest list for the party that maximizes the sum of the ‘fun’ ratings of the guests.

## ►4. Dynamic Programming: Longest Increasing Subsequence (LIS)

Give an  $O(n^2)$  algorithm to find the longest increasing subsequence of a sequence of numbers. Note: the elements of the subsequence need not be adjacent in the sequence. For example, the sequence (1, 5, 3, 2, 4) has an LIS (1, 3, 4).

## ►5. Nut/Bolt Median

You are given a set of  $n$  nuts and  $n$  bolts of different sizes. Each nut matches exactly one bolt (and vice versa, of course). The sizes of the nuts and bolts are so similar that you cannot compare two nuts or two bolts to see which is larger. You can, however, check whether a nut is too small, too large, or just right for a bolt (and vice versa, of course).

In this problem, your goal is to find the median bolt (*i.e.*, the  $\lfloor n/2 \rfloor$ th largest) as quickly as possible.

- (a) Describe an efficient deterministic algorithm that finds the median bolt. How many nut/bolt comparisons does your algorithm perform in the worst case?
- (b) Describe an efficient *randomized* algorithm that finds the median bolt.
  - i. State a recurrence for the expected number of nut/bolt comparisons your algorithm performs.
  - ii. What is the probability that your algorithm compares the  $i$ th largest bolt with the  $j$ th largest nut?
  - iii. What is the expected number of nut-bolt comparisons made by your algorithm? [Hint: Use your answer to either of the previous two questions.]

## Only 1U Grad Problems

►1. You are at a political convention with  $n$  delegates. Each delegate is a member of exactly one political party. It is impossible to tell which political party a delegate belongs to. However, you can check whether any two delegates are in the *same* party or not by introducing them to each other. (Members of the same party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.)

- (a) Suppose a majority (more than half) of the delegates are from the same political party. Give an efficient algorithm that identifies a member of the majority party.
- \*(b) Suppose exactly  $k$  political parties are represented at the convention and one party has a *plurality*: more delegates belong to that party than to any other. Give an efficient algorithm that identifies a member of the plurality party.

- \*(c) Suppose you don't know how many parties there are, but you do know that one party has a plurality, and at least  $p$  people in the plurality party are present. Present a practical procedure to pick a person from the plurality party as parsimoniously as possible. (Please.)
- ★(d) Finally, suppose you don't know how many parties are represented at the convention, and you don't know how big the plurality is. Give an efficient algorithm to identify a member of the plurality party. How is the running time of your algorithm affected by the number of parties ( $k$ )? By the size of the plurality ( $p$ )?

## Practice Problems

### 1. Second Smallest

Give an algorithm that finds the *second* smallest of  $n$  elements in at most  $n + \lceil \lg n \rceil - 2$  comparisons. Hint: divide and conquer to find the smallest; where is the second smallest?

### 2. Linear in-place 0-1 sorting

Suppose that you have an array of records whose keys to be sorted consist only of 0's and 1's. Give a simple, linear-time  $O(n)$  algorithm to sort the array in place (using storage of no more than constant size in addition to that of the array).

### 3. Dynamic Programming: Coin Changing

Consider the problem of making change for  $n$  cents using the least number of coins.

- (a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.
- (b) Suppose that the available coins have the values  $c^0, c^1, \dots, c^k$  for some integers  $c > 1$  and  $k \geq 1$ . Show that the greedy algorithm always yields an optimal solution.
- (c) Give a set of 4 coin values for which the greedy algorithm does not yield an optimal solution, show why.
- (d) Give a dynamic programming algorithm that yields an optimal solution for an arbitrary set of coin values.
- (e) Prove that, with only two coins  $a, b$  whose gcd is 1, the smallest value  $n$  for which change *can* be given for all values greater than or equal to  $n$  is  $(a - 1)(b - 1)$ .
- ★(f) For only three coins  $a, b, c$  whose gcd is 1, give an algorithm to determine the smallest value  $n$  for which change *can* be given for all values greater than  $n$ . (note: this problem is currently unsolved for  $n > 4$ .)

**4. Dynamic Programming: Paragraph Justification**

Consider the problem of printing a paragraph neatly on a printer (with fixed width font). The input text is a sequence of  $n$  words of lengths  $l_1, l_2, \dots, l_n$ . The line length is  $M$  (the maximum # of characters per line). We expect that the paragraph is left justified, that all first words on a line start at the leftmost position and that there is exactly one space between any two words on the same line. We want the uneven right ends of all the lines to be together as ‘neat’ as possible. Our criterion of neatness is that we wish to minimize the sum, over all lines except the last, of the cubes of the numbers of extra space characters at the ends of the lines. Note: if a printed line contains words  $i$  through  $j$ , then the number of spaces at the end of the line is  $M - j + i - \sum_{k=i}^j l_k$ .

- (a) Give a dynamic programming algorithm to do this.
- (b) Prove that if the neatness function is linear, a linear time greedy algorithm will give an optimum ‘neatness’.

**5. Comparison of Amortized Analysis Methods**

A sequence of  $n$  operations is performed on a data structure. The  $i$ th operation costs  $i$  if  $i$  is an exact power of 2, and 1 otherwise. That is operation  $i$  costs  $f(i)$ , where:

$$f(i) = \begin{cases} i, & i = 2^k, \\ 1, & \text{otherwise} \end{cases}$$

Determine the amortized cost per operation using the following methods of analysis:

- (a) Aggregate method
- (b) Accounting method
- \*(c) Potential method

# CS 373: Combinatorial Algorithms, Spring 1999

<http://www-courses.cs.uiuc.edu/~cs373>

## Homework 2 (due Thu. Feb. 18, 1999 by noon)

Name:	
Net ID:	Alias:

**Everyone must do the problems marked ▶.** Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

**Note:** When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
2. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
3. Justify the correctness of your algorithm, including termination if that is not obvious.
4. Analyze the time and space complexity of your algorithm.

### Undergrad/.75U Grad/1U Grad Problems

▶ 1. Faster Longest Increasing Subsequence (LIS)

Give an  $O(n \log n)$  algorithm to find the longest increasing subsequence of a sequence of numbers. Hint: In the dynamic programming solution, you don't really have to look back at all previous items.

▶ 2.  $\text{SELECT}(A, k)$

Say that a binary search tree is *augmented* if every node  $v$  also stores  $|v|$ , the size of its subtree.

- (a) Show that a rotation in an augmented binary tree can be performed in constant time.
- (b) Describe an algorithm  $\text{SCAPEGOATSELECT}(k)$  that selects the  $k$ th smallest item in an augmented scapegoat tree in  $O(\log n)$  worst-case time. (The scapegoat trees presented in class were already augmented.)
- (c) Describe an algorithm  $\text{SPLAYSELECT}(k)$  that selects the  $k$ th smallest item in an augmented splay tree in  $O(\log n)$  amortized time.
- (d) Describe an algorithm  $\text{TREAPSELECT}(k)$  that selects the  $k$ th smallest item in an augmented treap in  $O(\log n)$  expected time.

## ►3. Scapegoat trees

- (a) Prove that only one tree gets rebalanced at any insertion.
- (b) Prove that  $I(v) = 0$  in every node of a perfectly balanced tree ( $I(v) = \max(0, |\hat{v}| - |\check{v}|)$ , where  $\hat{v}$  is the child of greater height and  $\check{v}$  the child of lesser height,  $|v|$  is the number of nodes in subtree  $v$ , and perfectly balanced means each subtree has as close to half the leaves as possible and is perfectly balanced itself).
- \*(c) Show that you can rebuild a fully balanced binary tree in  $O(n)$  time using only  $O(1)$  additional memory.

## ►4. Memory Management

Suppose we can insert or delete an element into a hash table in constant time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:

- After an insertion, if the table is more than 3/4 full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
- After a deletion, if the table is less than 1/4 full, we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do not use the potential method (it makes it much more difficult).

## Only 1U Grad Problems

## ▷1. Detecting overlap

- (a) You are given a list of ranges represented by min and max (e.g. [1,3], [4,5], [4,9], [6,8], [7,10]). Give an  $O(n \log n)$ -time algorithm that decides whether or not a set of ranges contains a pair that overlaps. You need not report all intersections. If a range completely covers another, they are overlapping, even if the boundaries do not intersect.
- (b) You are given a list of rectangles represented by min and max  $x$ - and  $y$ - coordinates. Give an  $O(n \log n)$ -time algorithm that decides whether or not a set of rectangles contains a pair that overlaps (with the same qualifications as above). Hint: sweep a vertical line from left to right, performing some processing whenever an end-point is encountered. Use a balanced search tree to maintain any extra info you might need.

## Practice Problems

### 1. Amortization

- (a) Modify the binary double-counter (see class notes Feb. 2) to support a new operation Sign, which determines whether the number being stored is positive, negative, or zero, in constant time. The amortized time to increment or decrement the counter should still be a constant.

[Hint: Suppose  $p$  is the number of significant bits in  $P$ , and  $n$  is the number of significant bits in  $N$ . For example, if  $P = 17 = 10001_2$  and  $N = 0$ , then  $p = 5$  and  $n = 0$ . Then  $p - n$  always has the same sign as  $P - N$ . Assume you can update  $p$  and  $n$  in  $O(1)$  time.]

- \*(b) Do the same but now you can't assume that  $p$  and  $n$  can be updated in  $O(1)$  time.

### \*2. Amortization

Suppose instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of "fits", where the  $i$ th least significant fit indicates whether the sum includes the  $i$ th Fibonacci number  $F_i$ . For example, the fit string 101110 represents the number  $F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14$ . Describe algorithms to increment and decrement a fit string in constant amortized time. [Hint: Most numbers can be represented by more than one fit string. This is *not* the same representation as on Homework 0.]

### 3. Rotations

- (a) Show that it is possible to transform any  $n$ -node binary search tree into any other  $n$ -node binary search tree using at most  $2n - 2$  rotations.
- \*(b) Use fewer than  $2n - 2$  rotations. Nobody knows how few rotations are required in the worst case. There is an algorithm that can transform any tree to any other in at most  $2n - 6$  rotations, and there are pairs of trees that are  $2n - 10$  rotations apart. These are the best bounds known.

### 4. Fibonacci Heaps: SECONDMIN

We want to find the second smallest of a set efficiently.

- (a) Implement SECONDMIN by using a Fibonacci heap as a black box. Remember to justify its correctness and running time.
- \*(b) Modify the Fibonacci Heap data structure to implement SECONDMIN in constant time.
5. Give an efficient implementation of the operation **Fib-Heap-Change-Key**( $H, x, k$ ), which changes the key of a node  $x$  in a Fibonacci heap  $H$  to the value  $k$ . The changes you make to Fibonacci heap data structure to support your implementation should not affect the amortized running time of any other Fibonacci heap operations. Analyze the amortized running time of your implementation for cases in which  $k$  is greater than, less than, or equal to  $\text{key}[x]$ .

# CS 373: Combinatorial Algorithms, Spring 1999

<http://www-courses.cs.uiuc.edu/~cs373>

## Homework 2 (due Thu. Feb. 18, 1999 by noon)

Name:	
Net ID:	Alias:

**Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit.** (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

**Note:** When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
2. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
3. Justify the correctness of your algorithm, including termination if that is not obvious.
4. Analyze the time and space complexity of your algorithm.

### Undergrad/.75U Grad/1U Grad Problems

▶ 1. Faster Longest Increasing Subsequence (LIS)

Give an  $O(n \log n)$  algorithm to find the longest increasing subsequence of a sequence of numbers. Hint: In the dynamic programming solution, you don't really have to look back at all previous items.

▶ 2.  $\text{SELECT}(A, k)$

Say that a binary search tree is *augmented* if every node  $v$  also stores  $|v|$ , the size of its subtree.

- (a) Show that a rotation in an augmented binary tree can be performed in constant time.
- (b) Describe an algorithm  $\text{SCAPEGOATSELECT}(k)$  that selects the  $k$ th smallest item in an augmented scapegoat tree in  $O(\log n)$  worst-case time. (The scapegoat trees presented in class were already augmented.)
- (c) Describe an algorithm  $\text{SPLAYSELECT}(k)$  that selects the  $k$ th smallest item in an augmented splay tree in  $O(\log n)$  amortized time.
- (d) Describe an algorithm  $\text{TREAPSELECT}(k)$  that selects the  $k$ th smallest item in an augmented treap in  $O(\log n)$  expected time.

## ►3. Scapegoat trees

- (a) Prove that only one tree gets rebalanced at any insertion.
- (b) Prove that  $I(v) = 0$  in every node of a perfectly balanced tree ( $I(v) = \max(0, |\hat{v}| - |\check{v}|)$ , where  $\hat{v}$  is the child of greater height and  $\check{v}$  the child of lesser height,  $|v|$  is the number of nodes in subtree  $v$ , and perfectly balanced means each subtree has as close to half the leaves as possible and is perfectly balanced itself).
- \*(c) Show that you can rebuild a fully balanced binary tree in  $O(n)$  time using only  $O(1)$  additional memory.

## ►4. Memory Management

Suppose we can insert or delete an element into a hash table in constant time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:

- After an insertion, if the table is more than 3/4 full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
- After a deletion, if the table is less than 1/4 full, we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do not use the potential method (it makes it much more difficult).

## Only 1U Grad Problems

## ▷1. Detecting overlap

- (a) You are given a list of ranges represented by min and max (e.g. [1,3], [4,5], [4,9], [6,8], [7,10]). Give an  $O(n \log n)$ -time algorithm that decides whether or not a set of ranges contains a pair that overlaps. You need not report all intersections. If a range completely covers another, they are overlapping, even if the boundaries do not intersect.
- (b) You are given a list of rectangles represented by min and max  $x$ - and  $y$ - coordinates. Give an  $O(n \log n)$ -time algorithm that decides whether or not a set of rectangles contains a pair that overlaps (with the same qualifications as above). Hint: sweep a vertical line from left to right, performing some processing whenever an end-point is encountered. Use a balanced search tree to maintain any extra info you might need.

## Practice Problems

### 1. Amortization

- (a) Modify the binary double-counter (see class notes Feb. 2) to support a new operation Sign, which determines whether the number being stored is positive, negative, or zero, in constant time. The amortized time to increment or decrement the counter should still be a constant.

[Hint: Suppose  $p$  is the number of significant bits in  $P$ , and  $n$  is the number of significant bits in  $N$ . For example, if  $P = 17 = 10001_2$  and  $N = 0$ , then  $p = 5$  and  $n = 0$ . Then  $p - n$  always has the same sign as  $P - N$ . Assume you can update  $p$  and  $n$  in  $O(1)$  time.]

- \*(b) Do the same but now you can't assume that  $p$  and  $n$  can be updated in  $O(1)$  time.

### \*2. Amortization

Suppose instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of "fits", where the  $i$ th least significant fit indicates whether the sum includes the  $i$ th Fibonacci number  $F_i$ . For example, the fit string 101110 represents the number  $F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14$ . Describe algorithms to increment and decrement a fit string in constant amortized time. [Hint: Most numbers can be represented by more than one fit string. This is *not* the same representation as on Homework 0.]

### 3. Rotations

- (a) Show that it is possible to transform any  $n$ -node binary search tree into any other  $n$ -node binary search tree using at most  $2n - 2$  rotations.
- \*(b) Use fewer than  $2n - 2$  rotations. Nobody knows how few rotations are required in the worst case. There is an algorithm that can transform any tree to any other in at most  $2n - 6$  rotations, and there are pairs of trees that are  $2n - 10$  rotations apart. These are the best bounds known.

### 4. Fibonacci Heaps: SECONDMIN

We want to find the second smallest of a set efficiently.

- (a) Implement SECONDMIN by using a Fibonacci heap as a black box. Remember to justify its correctness and running time.
- \*(b) Modify the Fibonacci Heap data structure to implement SECONDMIN in constant time.
5. Give an efficient implementation of the operation **Fib-Heap-Change-Key**( $H, x, k$ ), which changes the key of a node  $x$  in a Fibonacci heap  $H$  to the value  $k$ . The changes you make to Fibonacci heap data structure to support your implementation should not affect the amortized running time of any other Fibonacci heap operations. Analyze the amortized running time of your implementation for cases in which  $k$  is greater than, less than, or equal to  $\text{key}[x]$ .

# CS 373: Combinatorial Algorithms, Spring 1999

<http://www-courses.cs.uiuc.edu/~cs373>

**Homework 3 (due Thu. Mar. 11, 1999 by noon)**

Name:	
Net ID:	Alias:

**Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit.** (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

**Note:** When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. (New!) If not already done, model the problem appropriately. Often the problem is stated in real world terms; give a more rigorous description of the problem. This will help you figure out what is assumed (what you know and what is arbitrary, what operations are and are not allowed), and find the tools needed to solve the problem.
2. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
3. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
4. Justify the correctness of your algorithm, including termination if that is not obvious.
5. Analyze the time and space complexity of your algorithm.

## Undergrad/.75U Grad/1U Grad Problems

### ▶ 1. Hashing

- (a) (2 pts) Consider an open-address hash table with uniform hashing and a load factor  $\alpha = 1/2$ . What is the expected number of probes in an unsuccessful search? Successful search?
- (b) (3 pts) Let the hash function for a table of size  $m$  be

$$h(x) = \lfloor Ax \rfloor \bmod m$$

where  $A = \hat{\phi} = \frac{\sqrt{5}-1}{2}$ . Show that this gives the best possible spread, i.e. if the  $x$  are hashed in order,  $x+1$  will be hashed in the largest remaining contiguous interval.

►2. (5 pts) Euler Tour:

Given an **undirected** graph  $G = (V, E)$ , give an algorithm that finds a cycle in the graph that visits every edge *exactly* once, or says that it can't be done.

►3. (5 pts) Makefiles:

In order to facilitate recompiling programs from multiple source files when only a small number of files have been updated, there is a UNIX utility called ‘make’ that only recompiles those files that were changed, *and* any intermediate files in the compilation that depend on those changed. Design an algorithm to recompile only those necessary.

►4. (5 pts) Shortest Airplane Trip:

A person wants to fly from city  $A$  to city  $B$  in the shortest possible time. S/he turns to the traveling agent who knows all the departure and arrival times of all the flights on the planet. Give an algorithm that will allow the agent to choose an optimal route. Hint: rather than modify Dijkstra’s algorithm, modify the data. The time is from departure to arrival at the destination, so it will include layover time (time waiting for a connecting flight).

►5. (9 pts, 3 each) Minimum Spanning Tree changes Suppose you have a graph  $G$  and an MST of that graph (i.e. the MST has already been constructed).

- Give an algorithm to update the MST when an edge is added to  $G$ .
- Give an algorithm to update the MST when an edge is deleted from  $G$ .
- Give an algorithm to update the MST when a vertex (and possibly edges to it) is added to  $G$ .

## Only 1U Grad Problems

▷1. Nesting Envelopes

You are given an unlimited number of each of  $n$  different types of envelopes. The dimensions of envelope type  $i$  are  $x_i \times y_i$ . In nesting envelopes inside one another, you can place envelope  $A$  inside envelope  $B$  if and only if the dimensions  $A$  are *strictly smaller* than the dimensions of  $B$ . Design and analyze an algorithm to determine the largest number of envelopes that can be nested inside one another.

## Practice Problems

1. The incidence matrix of an undirected graph  $G = (V, E)$  is a  $|V| \times |E|$  matrix  $B = (b_{ij})$  such that

$$b_{ij} = \begin{cases} 1 & (i, j) \in E, \\ 0 & (i, j) \notin E. \end{cases}$$

- (a) Describe what all the entries of the matrix product  $BB^T$  represent ( $B^T$  is the matrix transpose). Justify.
- (b) Describe what all the entries of the matrix product  $B^TB$  represent. Justify.
- ★(c) Let  $C = BB^T - 2A$ . Let  $C'$  be  $C$  with the first row and column removed. Show that  $\det C'$  is the number of spanning trees. ( $A$  is the adjacency matrix of  $G$ , with zeroes on the diagonal).

2.  $o(V^2)$  Adjacency Matrix Algorithms

- (a) Give an  $O(V)$  algorithm to decide whether a directed graph contains a *sink* in an adjacency matrix representation. A sink is a vertex with in-degree  $V - 1$ .

- (b) An undirected graph is a scorpion if it has a vertex of degree 1 (the sting) connected to a vertex of degree two (the tail) connected to a vertex of degree  $V - 2$  (the body) connected to the other  $V - 3$  vertices (the feet). Some of the feet may be connected to other feet.

Design an algorithm that decides whether a given adjacency matrix represents a scorpion by examining only  $O(V)$  of the entries.

- (c) Show that it is impossible to decide whether  $G$  has at least one edge in  $O(V)$  time.

3. Shortest Cycle:

Given an **undirected** graph  $G = (V, E)$ , and a weight function  $f : E \rightarrow \mathbf{R}$  on the **edges**, give an algorithm that finds (in time polynomial in  $V$  and  $E$ ) a cycle of smallest weight in  $G$ .

4. Longest Simple Path:

Let graph  $G = (V, E)$ ,  $|V| = n$ . A *simple path* of  $G$ , is a path that does not contain the same vertex twice. Use dynamic programming to design an algorithm (not polynomial time) to find a simple path of maximum length in  $G$ . Hint: It can be done in  $O(n^c 2^n)$  time, for some constant  $c$ .

5. Minimum Spanning Tree:

Suppose all edge weights in a graph  $G$  are equal. Give an algorithm to compute an MST.

6. Transitive reduction:

Give an algorithm to construct a *transitive reduction* of a directed graph  $G$ , i.e. a graph  $G^{TR}$  with the fewest edges (but with the same vertices) such that there is a path from  $a$  to  $b$  in  $G$  iff there is also such a path in  $G^{TR}$ .

7. (a) What is  $5^{2^{29}5^0 + 23^4 + 17^3 + 11^2 + 5^1} \mod 6$ ?

- (b) What is the capital of Nebraska? Hint: It is not Omaha. It is named after a famous president of the United States that was not George Washington. The distance from the Earth to the Moon averages roughly 384,000 km.

# CS 373: Combinatorial Algorithms, Spring 1999

<http://www-courses.cs.uiuc.edu/~cs373>

## Homework 4 (due Thu. Apr. 1, 1999 by noon)

Name:	
Net ID:	Alias:

**Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit.** (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

**Note:** When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. (New!) If not already done, model the problem appropriately. Often the problem is stated in real world terms; give a more rigorous description of the problem. This will help you figure out what is assumed (what you know and what is arbitrary, what operations are and are not allowed), and find the tools needed to solve the problem.
2. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
3. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
4. Justify the correctness of your algorithm, including termination if that is not obvious.
5. Analyze the time and space complexity of your algorithm.

### Undergrad/.75U Grad/1U Grad Problems

▶ 1. (5 pts total) Collinearity

Give an  $O(n^2 \log n)$  algorithm to determine whether any three points of a set of  $n$  points are collinear. Assume two dimensions and exact arithmetic.

▶ 2. (4 pts, 2 each) Convex Hull Recurrence

Consider the following generic recurrence for convex hull algorithms that divide and conquer:

$$T(n, h) = T(n_1, h_1) + T(n_2, h_2) + O(n)$$

where  $n \geq n_1 + n_2$ ,  $h = h_1 + h_2$  and  $n \geq h$ . This means that the time to compute the convex hull is a function of both  $n$ , the number of input points, and  $h$ , the number of convex hull vertices. The splitting and merging parts of the divide-and-conquer algorithm take  $O(n)$  time. When  $n$  is a constant,  $T(n, h)$  is  $O(1)$ , but when  $h$  is a constant,  $T(n, h)$  is  $O(n)$ . Prove that for both of the following restrictions, the solution to the recurrence is  $O(n \log h)$ :

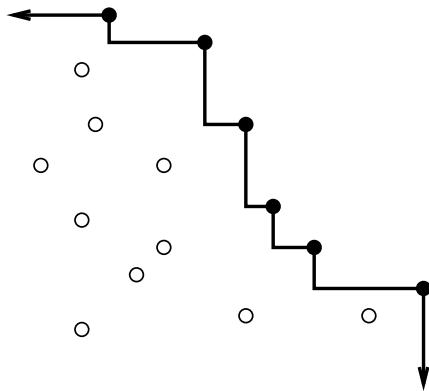
- (a)  $h_1, h_2 < \frac{3}{4}h$
- (b)  $n_1, n_2 < \frac{3}{4}n$

►3. (5 pts) Circle Intersection

Give an  $O(n \log n)$  algorithm to test whether any two circles in a set of size  $n$  intersect.

►4. (5 pts total) Staircases

You are given a set of points in the first quadrant. A *left-up* point of this set is defined to be a point that has no points both greater than it in both coordinates. The left-up subset of a set of points then forms a *staircase* (see figure).



- (a) (3 pts) Give an  $O(n \log n)$  algorithm to find the staircase of a set of points.

- (b) (2 pts) Assume that points are chosen uniformly at random within a rectangle. What is the average number of points in a staircase? Justify. Hint: you will be able to give an exact answer rather than just asymptotics. You have seen the same analysis before.

### Only 1U Grad Problems

►1. (6 pts, 2 each) Ghostbusters and Ghosts

A group of  $n$  ghostbusters is battling  $n$  ghosts. Each ghostbuster can shoot a single energy beam at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits a ghost. The ghostbusters must all fire at the same time and no two energy beams may cross. The positions of the ghosts and ghostbusters is fixed in the plane (assume that no three points are collinear).

- (a) Prove that for any configuration ghosts and ghostbusters there exists such a non-crossing matching.
- (b) Show that there exists a line passing through one ghostbuster and one ghost such that the number of ghostbusters on one side of the line equals the number of ghosts on the same side. Give an efficient algorithm to find such a line.
- (c) Give an efficient divide and conquer algorithm to pair ghostbusters and ghosts so that no two streams cross.

## Practice Problems

1. Basic Computation (assume two dimensions and *exact* arithmetic)
  - (a) Intersection: Extend the basic algorithm to determine if two line segments intersect by taking care of *all* degenerate cases.
  - (b) Simplicity: Give an  $O(n \log n)$  algorithm to determine whether an  $n$ -vertex polygon is simple.
  - (c) Area: Give an algorithm to compute the area of a simple  $n$ -polygon (not necessarily convex) in  $O(n)$  time.
  - (d) Inside: Give an algorithm to determine whether a point is within a simple  $n$ -polygon (not necessarily convex) in  $O(n)$  time.
  
2. External Diagonals and Mouths
  - (a) A pair of polygon vertices defines an *external diagonal* if the line segment between them is completely outside the polygon. Show that every nonconvex polygon has at least one external diagonal.
  - (b) Three consecutive polygon vertices  $p, q, r$  form a *mouth* if  $p$  and  $r$  define an external diagonal. Show that every nonconvex polygon has at least one mouth.
  
3. On-Line Convex Hull
 

We are given the set of points one point at a time. After receiving each point, we must compute the convex hull of all those points so far. Give an algorithm to solve this problem in  $O(n^2)$  (We could obviously use Graham's scan  $n$  times for an  $O(n^2 \log n)$  algorithm). Hint: How do you maintain the convex hull?
  
4. Another On-Line Convex Hull Algorithm
  - (a) Given an  $n$ -polygon and a point outside the polygon, give an algorithm to find a tangent.
  - \*(b) Suppose you have found both tangents. Give an algorithm to remove the points from the polygon that are within the angle formed by the tangents (as segments!) and the opposite side of the polygon.
  - (c) Use the above to give an algorithm to compute the convex hull on-line in  $O(n \log n)$
  
- ★5. Order of the size of the convex hull
 

The convex hull on  $n \geq 3$  points can have anywhere from 3 to  $n$  points. The average case depends on the distribution.

  - (a) Prove that if a set of points is chosen randomly within a given rectangle, then the average size of the convex hull is  $O(\log n)$ .
  - (b) Prove that if a set of points is chosen randomly within a given circle, then the average size of the convex hull is  $O(\sqrt{n})$ .

# CS 373: Combinatorial Algorithms, Spring 1999

<http://www-courses.cs.uiuc.edu/~cs373>

## Homework 5 (due Thu. Apr. 22, 1999 by noon)

Name:	
Net ID:	Alias:

**Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit.** (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

**Note:** You will be held accountable for the appropriate responses for answers (e.g. give models, proofs, analyses, etc)

---

### Undergrad/.75U Grad/1U Grad Problems

- ▶1. (5 pts) Show how to find the occurrences of pattern  $P$  in text  $T$  by computing the prefix function of the string  $PT$  (the concatenation of  $P$  and  $T$ ).
  
- ▶2. (10 pts total) Fibonacci strings and KMP matching  
*Fibonacci strings* are defined as follows:

$$F_1 = \text{"b"}, \quad F_2 = \text{"a"}, \quad \text{and } F_n = F_{n-1}F_{n-2}, (n > 2)$$

where the recursive rule uses concatenation of strings, so  $F_2$  is "ab",  $F_3$  is "aba". Note that the length of  $F_n$  is the  $n$ th Fibonacci number.

- (a) (2 pts) Prove that in any Fibonacci string there are no two b's adjacent and no three a's.
- (b) (2 pts) Give the unoptimized and optimized 'prefix' (fail) function for  $F_7$ .
- (c) (3 pts) Prove that, in searching for a Fibonacci string of length  $m$  using unoptimized KMP, it may shift up to  $\lceil \log_\phi m \rceil$  times, where  $\phi = (1 + \sqrt{5})/2$ , is the golden ratio. (Hint: Another way of saying the above is that we are given the string  $F_n$  and we may have to shift  $n$  times. Find an example text T that gives this number of shifts).
- (d) (3 pts) What happens here when you use the optimized prefix function? Explain.
  
- ▶3. (5 pts) Prove that finding the second smallest of  $n$  elements takes  $n + \lceil \lg n \rceil - 2$  comparisons in the worst case. Prove for both upper and lower bounds. Hint: find the (first) smallest using an elimination tournament.
  
- ▶4. (4 pts, 2 each) Lower Bounds on Adjacency Matrix Representations of Graphs
  - (a) Prove that the time to determine if an undirected graph has a cycle is  $\Omega(V^2)$ .

- (b) Prove that the time to determine if there is a path between two nodes in an undirected graph is  $\Omega(V^2)$ .

## Only 1U Grad Problems

- ▷ 1. (5 pts) Prove that  $\lceil 3n/2 \rceil - 2$  comparisons are necessary in the worst case to find both the minimum and maximum of  $n$  numbers. Hint: Consider how many are potentially either the min or max.

## Practice Problems

### 1. String matching with wild-cards

Suppose you have an alphabet for patterns that includes a ‘gap’ or wild-card character; any length string of any characters can match this additional character. For example if ‘\*’ is the wild-card, then the pattern ‘foo\*bar\*nad’ can be found in ‘foofoowangbarnad’. Modify the computation of the prefix function to correctly match strings using KMP.

2. Prove that there is no comparison sort whose running time is linear for at least  $1/2$  of the  $n!$  inputs of length  $n$ . What about at least  $1/n$ ? What about at least  $1/2^n$ ?
3. Prove that  $2n - 1$  comparisons are necessary in the worst case to merge two sorted lists containing  $n$  elements each.

4. Find asymptotic upper and lower bounds to  $\lg(n!)$  without Stirling’s approximation (Hint: use integration).
5. Given a sequence of  $n$  elements of  $n/k$  blocks ( $k$  elements per block) all elements in a block are less than those to the right in sequence, show that you cannot have the whole sequence sorted in better than  $\Omega(n \lg k)$ . Note that the entire sequence would be sorted if each of the  $n/k$  blocks were individually sorted in place. Also note that combining the lower bounds for each block is not adequate (that only gives an upper bound).

### 6. Some elementary reductions

- (a) Prove that if you can decide whether a graph  $G$  has a clique of size  $k$  (or less) then you can decide whether a graph  $G'$  has an independent set of size  $k$  (or more).
- (b) Prove that if you can decide whether one graph  $G_1$  is a subgraph of another graph  $G_2$  then you can decide whether a graph  $G$  has a clique of size  $k$  (or less).

### 7. There is no Proof but We are pretty Sure

Justify (prove) the following logical rules of inference:

- (a) Classical - If  $a \rightarrow b$  and  $a$  hold, then  $b$  holds.
- (b) Fuzzy - Prove: If  $a \rightarrow b$  holds, and  $a$  holds with probability  $p$ , then  $b$  holds with probability less than  $p$ . Assume all probabilities are independent.
- (c) Give formulas for computing the probabilities of the fuzzy logical operators ‘and’, ‘or’, ‘not’, and ‘implies’, and fill out truth tables with the values T (true,  $p = 1$ ), L (likely,  $p = 0.9$ ), M (maybe,  $p = 0.5$ ), N (not likely,  $p = 0.1$ ), and F (false,  $p = 0$ ).

- (d) If you have a poly time (algorithmic) reduction from problem  $B$  to problem  $A$  (i.e. you can solve  $B$  using  $A$  with a poly time conversion), and it is very unlikely that  $A$  has better than lower bound  $\Omega(2^n)$  algorithm, what can you say about problem  $A$ . Hint: a solution to  $A$  implies a solution to  $B$ .

# CS 373: Combinatorial Algorithms, Spring 1999

## Midterm 1 (February 23, 1999)

Name:

Net ID:

Alias:

This is a closed-book, closed-notes exam!

If you brought anything with you besides writing instruments and your  $8\frac{1}{2}'' \times 11''$  cheat sheet, please leave it at the front of the classroom.

---

- **Don't panic!**

- Print your name, netid, and alias in the boxes above, and print your name at the top of every page.
  - **Answer four of the five questions on the exam.** Each question is worth 10 points. If you answer more than four questions, the one with the lowest score will be ignored. **1-unit graduate students must answer question #5.**
  - Please write your answers on the front of the exam pages. Use the backs of the pages as scratch paper. Let us know if you need more paper.
  - Read the entire exam before writing anything. Make sure you understand what the questions are asking. If you give a beautiful answer to the wrong question, you'll get no credit. If any question is unclear, please ask one of us for clarification.
  - Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.
  - Write *something* down for every problem. Don't panic and erase large chunks of work. Even if you think it's nonsense, it might be worth partial credit.
- **Don't panic!**
- 

#	Score	Grader
1		
2		
3		
4		
5		

**1. Multiple Choice**

Every question below has one of the following answers.

- (a)  $\Theta(1)$       (b)  $\Theta(\log n)$       (c)  $\Theta(n)$       (d)  $\Theta(n \log n)$       (e)  $\Theta(n^2)$

For each question, write the letter that corresponds to your answer. You do not need to justify your answer. Each correct answer earns you 1 point, but each incorrect answer *costs* you  $\frac{1}{2}$  point. (You cannot score below zero.)

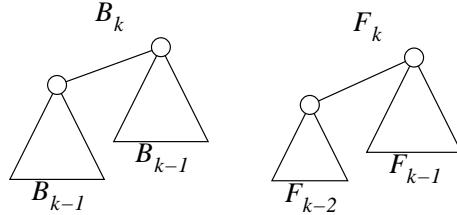
- What is  $\sum_{i=1}^n i$ ?
- What is  $\sum_{i=1}^n \frac{1}{i}$ ?
- What is the solution of the recurrence  $T(n) = T(\sqrt{n}) + n$ ?
- What is the solution of the recurrence  $T(n) = T(n - 1) + \lg n$ ?
- What is the solution of the recurrence  $T(n) = 2T\left(\lceil \frac{n+27}{2} \rceil\right) + 5n - 7\sqrt{\lg n} + \frac{1999}{n}$ ?
- The amortized time for inserting one item into an  $n$ -node splay tree is  $O(\log n)$ . What is the worst-case time for a sequence of  $n$  insertions into an initially empty splay tree?
- The expected time for inserting one item into an  $n$ -node randomized treap is  $O(\log n)$ . What is the worst-case time for a sequence of  $n$  insertions into an initially empty treap?
- What is the worst-case running time of randomized quicksort?
- How many bits are there in the binary representation of the  $n$ th Fibonacci number?
- What is the worst-case cost of merging two arbitrary splay trees with  $n$  items total into a single splay tree with  $n$  items.
- Suppose you correctly identify three of the answers to this question as obviously wrong. If you pick one of the two remaining answers at random, what is your expected score for this problem?

2. (a) [5 pt] Recall that a binomial tree of order  $k$ , denoted  $B_k$ , is defined recursively as follows.  $B_0$  is a single node. For any  $k > 0$ ,  $B_k$  consists of two copies of  $B_{k-1}$  linked together.

Prove that the degree of any node in a binomial tree is equal to its height.

- (b) [5 pt] Recall that a Fibonacci tree of order  $k$ , denoted  $F_k$ , is defined recursively as follows.  $F_1$  and  $F_2$  are both single nodes. For any  $k > 2$ ,  $F_k$  consists of an  $F_{k-2}$  linked to an  $F_{k-1}$ .

Prove that for any node  $v$  in a Fibonacci tree,  $\text{height}(v) = \lceil \text{degree}(v)/2 \rceil$ .



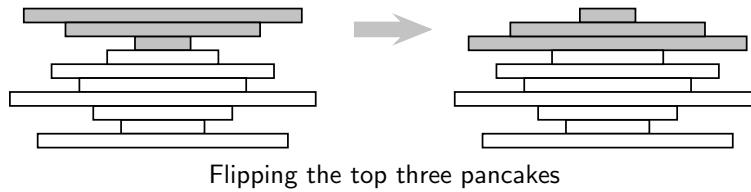
Recursive definitions of binomial trees and Fibonacci trees.

3. Consider the following randomized algorithm for computing the smallest element in an array.

```
RANDOMMIN( $A[1..n]$ ):  
     $min \leftarrow \infty$   
    for  $i \leftarrow 1$  to  $n$  in random order  
        if  $A[i] < min$   
             $min \leftarrow A[i]$  (*)  
    return  $min$ 
```

- (a) [1 pt] In the worst case, how many times does RANDOMMIN execute line (\*)?
- (b) [3 pt] What is the probability that line (\*) is executed during the  $n$ th iteration of the for loop?
- (c) [6 pt] What is the exact expected number of executions of line (\*)? (A correct  $\Theta()$  bound is worth 4 points.)

4. Suppose we have a stack of  $n$  pancakes of different sizes. We want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation we can perform is a *flip* — insert a spatula under the top  $k$  pancakes, for some  $k$  between 1 and  $n$ , and flip them all over.



- (a) [3 pt] Describe an algorithm to sort an arbitrary stack of  $n$  pancakes.
- (b) [3 pt] Prove that your algorithm is correct.
- (c) [2 pt] Exactly how many flips does your algorithm perform in the worst case? (A correct  $\Theta()$  bound is worth one point.)
- (d) [2 pt] Suppose one side of each pancake is burned. Exactly how many flips do you need to sort the pancakes *and* have the burned side of every pancake on the bottom? (A correct  $\Theta()$  bound is worth one point.)

5. You are given an array  $A[1..n]$  of integers. Describe and analyze an algorithm that finds the largest sum of elements in a contiguous subarray  $A[i..j]$ .

For example, if the array contains the numbers  $(-6, 12, -7, 0, 14, -7, 5)$ , then the largest sum is  $19 = 12 - 7 + 0 + 14$ .

-6	12	-7	0	14	-7	5
19						

To get full credit, your algorithm must run in  $\Theta(n)$  time — there are at least three different ways to do this. An algorithm that runs in  $\Theta(n^2)$  time is worth 7 points.

# CS 373: Combinatorial Algorithms, Spring 1999

## Midterm 2 (April 6, 1999)

Name:

Net ID:

Alias:

This is a closed-book, closed-notes exam!

If you brought anything with you besides writing instruments and your  $8\frac{1}{2}'' \times 11''$  cheat sheet, please leave it at the front of the classroom.

- 
- Don't panic!
  - Print your name, netid, and alias in the boxes above, and print your name at the top of every page.
  - Answer four of the five questions on the exam. Each question is worth 10 points. If you answer more than four questions, the one with the lowest score will be ignored. **1-unit graduate students must answer question #5.**
  - Please write your answers on the front of the exam pages. You can use the backs of the pages as scratch paper. Let us know if you need more paper.
  - Read the entire exam before writing anything. Make sure you understand what the questions are asking. If you give a beautiful answer to the wrong question, you'll get no credit. If any question is unclear, please ask one of us for clarification.
  - Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.
  - Write *something* down for every problem. Don't panic and erase large chunks of work. Even if you think it's nonsense, it might be worth partial credit.
  - Don't panic!
- 

#	Score	Grader
1		
2		
3		
4		
5		

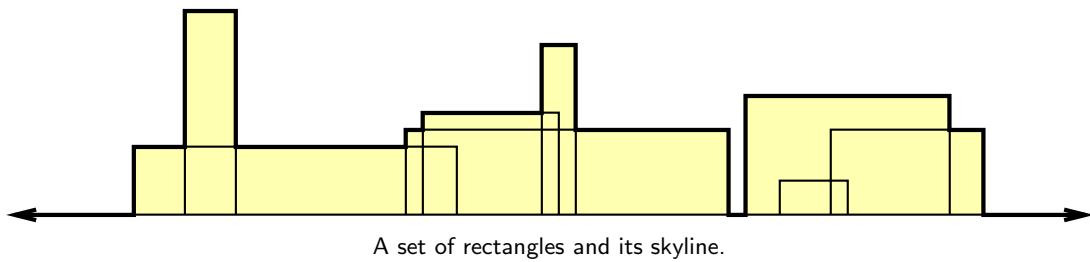
**1. Bipartite Graphs**

A graph  $(V, E)$  is *bipartite* if the vertices  $V$  can be partitioned into two subsets  $L$  and  $R$ , such that every edge has one vertex in  $L$  and the other in  $R$ .

- (a) Prove that every tree is a bipartite graph.
- (b) Describe and analyze an efficient algorithm that determines whether a given connected, undirected graph is bipartite.

**2. Manhattan Skyline**

The purpose of the following problem is to compute the outline of a projection of rectangular buildings. You are given the height, width, and left  $x$ -coordinate of  $n$  rectangles. The bottom of each rectangle is on the  $x$ -axis. Describe and analyze an efficient algorithm to compute the vertices of the “skyline”.



**3. Least Cost Vertex Weighted Path**

Suppose you want to drive from Champaign to Los Angeles via a network of roads connecting cities. You don't care how long it takes, how many cities you visit, or how much gas you use. All you care about is how much money you spend on food. Each city has a possibly different, but fixed, value for food.

More formally, you are given a directed graph  $G = (V, E)$  with nonnegative weights on the vertices  $w: V \rightarrow \mathbb{R}^+$ , a source vertex  $s \in V$ , and a target vertex  $t \in V$ . Describe and analyze an efficient algorithm to find a minimum-weight path from  $s$  to  $t$ . [Hint: Modify the graph.]

#### 4. Union-Find with Alternate Rule

In the UNION-FIND data structure described in CLR and in class, each set is represented by a rooted tree. The UNION algorithm, given two sets, decides which set is to be the parent of the other by comparing their ranks, where the rank of a set is an upper bound on the height of its tree.

Instead of rank, we propose using the *weight* of the set, which is just the number of nodes in the set. Here's the modified UNION algorithm:

```
UNION(A, B):
    if weight(A) > weight(B)
        parent(B) ← A
        weight(A) ← weight(A) + weight(B)
    else
        parent(A) ← B
        weight(B) ← weight(A) + weight(B)
```

Prove that if we use this method, then after any sequence of  $n$  MAKESETS, UNIONS, and FINDS (with path compression), the *height* of the tree representing any set is  $O(\log n)$ .

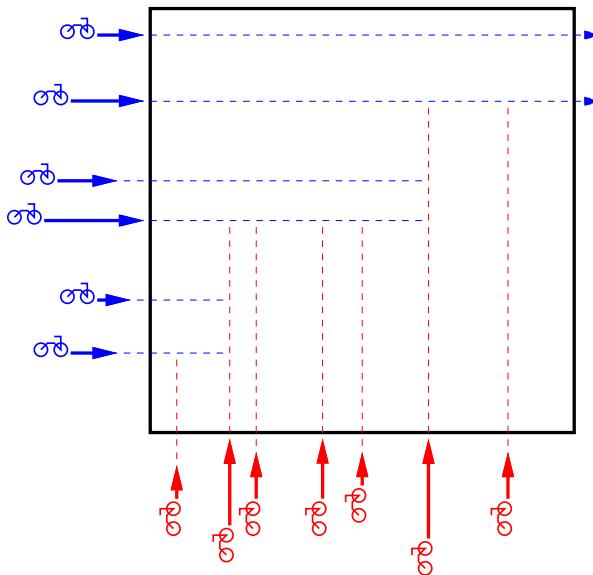
[Hint: First prove it without path compression, and then argue that path compression doesn't matter (for this problem).]

### 5. Motorcycle Collision

One gang, Hell's Ordinates, start west of the arena facing directly east; the other, The Vicious Abscissas of Death, start south of the arena facing due north. All the motorcycles start moving simultaneously at a prearranged signal. Each motorcycle moves at a fixed speed—no speeding up, slowing down, or turning is allowed. Each motorcycle leaves an oil trail behind it. If another motorcycle crosses that trail, it falls over and stops leaving a trail.

More formally, we are given two sets  $H$  and  $V$ , each containing  $n$  motorcycles. Each motorcycle is represented by three numbers  $(s, x, y)$ : its speed and the  $x$ - and  $y$ -coordinates of its initial location. Bikes in  $H$  move horizontally; bikes in  $V$  move vertically.

Assume that the bikes are infinitely small points, that the bike trails are infinitely thin lie segments, that a bike crashes stops *exactly* when it hits a oil trail, and that no two bikes collide with each other.



Two sets of motorcycles and the oil trails they leave behind.

- (a) Solve the case  $n = 1$ . Given only two motorcycles moving perpendicular to each other, determine which one of them falls over and where in  $O(1)$  time.
- (b) Describe an efficient algorithm to find the set of all points where motorcycles fall over.

**5. Motorcycle Collision (continued)**

Incidentally, the movie *Tron* is being shown during Roger Ebert's Forgotten Film Festival at the Virginia Theater in Champaign on April 25. Get your tickets now!

# CS 373: Combinatorial Algorithms, Spring 1999

## Final Exam (May 7, 1999)

Name:	
Net ID:	Alias:

This is a closed-book, closed-notes exam!

If you brought anything with you besides writing instruments and your two  $8\frac{1}{2}'' \times 11''$  cheat sheets, please leave it at the front of the classroom.

- 
- Print your name, netid, and alias in the boxes above, and print your name at the top of every page.
  - **Answer six of the seven questions on the exam.** Each question is worth 10 points. If you answer every question, the one with the lowest score will be ignored. **1-unit graduate students must answer question #7.**
  - Please write your answers on the front of the exam pages. Use the backs of the pages as scratch paper. Let us know if you need more paper.
  - Read the entire exam before writing anything. Make sure you understand what the questions are asking. If you give a beautiful answer to the wrong question, you'll get no credit. If any question is unclear, please ask one of us for clarification.
  - Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.
  - Write *something* down for every problem. Don't panic and erase large chunks of work. Even if you think it's nonsense, it might be worth partial credit.
- 

#	Score	Grader
1		
2		
3		
4		
5		
6		
7		

## 1. Short Answer

sorting	induction	Master theorem	divide and conquer
randomized algorithm	amortization	brute force	hashing
binary search	depth-first search	splay tree	Fibonacci heap
convex hull	sweep line	minimum spanning tree	shortest paths
shortest path	adversary argument	NP-hard	reduction
string matching	evasive graph property	dynamic programming	$H_n$

Choose from the list above the best method for solving each of the following problems. We do *not* want complete solutions, just a short description of the proper solution technique! Each item is worth 1 point.

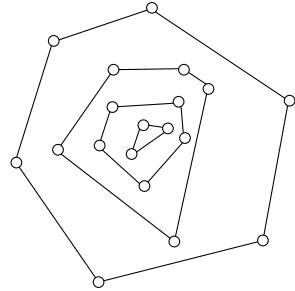
- (a) Given a Champaign phone book, find your own phone number.
- (b) Given a collection of  $n$  rectangles in the plane, determine whether any two intersect in  $O(n \log n)$  time.
- (c) Given an undirected graph  $G$  and an integer  $k$ , determine if  $G$  has a complete subgraph with  $k$  edges.
- (d) Given an undirected graph  $G$ , determine if  $G$  has a triangle — a complete subgraph with three vertices.
- (e) Prove that any  $n$ -vertex graph with minimum degree at least  $n/2$  has a Hamiltonian cycle.
- (f) Given a graph  $G$  and three distinguished vertices  $u$ ,  $v$ , and  $w$ , determine whether  $G$  contains a path from  $u$  to  $v$  that passes through  $w$ .
- (g) Given a graph  $G$  and two distinguished vertices  $u$  and  $v$ , determine whether  $G$  contains a path from  $u$  to  $v$  that passes through at most 17 edges.
- (h) Solve the recurrence  $T(n) = 5T(n/17) + O(n^{4/3})$ .
- (i) Solve the recurrence  $T(n) = 1/n + T(n - 1)$ , where  $T(0) = 0$ .
- (j) Given an array of  $n$  integers, find the integer that appears most frequently in the array.

- (a) \_\_\_\_\_ (f) \_\_\_\_\_
- (b) \_\_\_\_\_ (g) \_\_\_\_\_
- (c) \_\_\_\_\_ (h) \_\_\_\_\_
- (d) \_\_\_\_\_ (i) \_\_\_\_\_
- (e) \_\_\_\_\_ (j) \_\_\_\_\_

**2. Convex Layers**

Given a set  $Q$  of points in the plane, define the *convex layers* of  $Q$  inductively as follows: The first convex layer of  $Q$  is just the convex hull of  $Q$ . For all  $i > 1$ , the  $i$ th convex layer is the convex hull of  $Q$  after the vertices of the first  $i - 1$  layers have been removed.

Give an  $O(n^2)$ -time algorithm to find all convex layers of a given set of  $n$  points. [Partial credit for a correct slower algorithm; extra credit for a correct faster algorithm.]



A set of points with four convex layers.

3. Suppose you are given an array of  $n$  numbers, sorted in increasing order.

- (a) [3 pts] Describe an  $O(n)$ -time algorithm for the following problem:

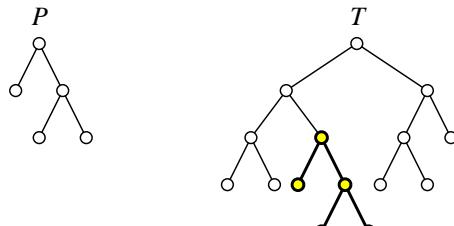
Find two numbers from the list that add up to zero, or report that there is no such pair.  
In other words, find two numbers  $a$  and  $b$  such that  $a + b = 0$ .

- (b) [7 pts] Describe an  $O(n^2)$ -time algorithm for the following problem:

Find *three* numbers from the list that add up to zero, or report that there is no such triple.  
In other words, find three numbers  $a$ ,  $b$ , and  $c$ , such that  $a + b + c = 0$ . [Hint: Use something similar to part (a) as a subroutine.]

**4. Pattern Matching**

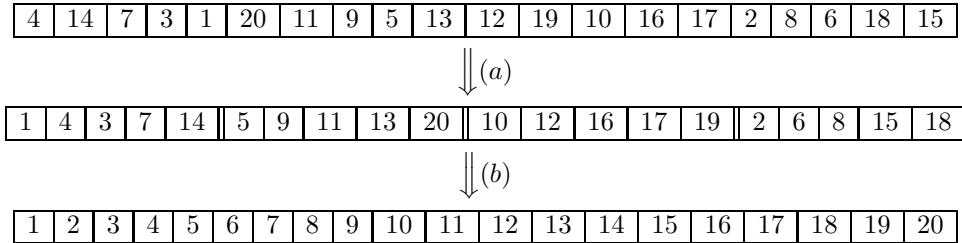
- (a) [4 pts] A *cyclic rotation* of a string is obtained by chopping off a prefix and gluing it at the end of the string. For example, ALGORITHM is a cyclic shift of RITHMALGO. Describe and analyze an algorithm that determines whether one string  $P[1..m]$  is a cyclic rotation of another string  $T[1..n]$ .
- (b) [6 pts] Describe and analyze an algorithm that decides, given any two binary trees  $P$  and  $T$ , whether  $P$  equals a subtree of  $T$ . [Hint: First transform both trees into strings.]



$P$  occurs exactly once as a subtree of  $T$ .

**5. Two-stage Sorting**

- (a) [1 pt] Suppose we are given an array  $A[1..n]$  of distinct integers. Describe an algorithm that splits  $A$  into  $n/k$  subarrays, each with  $k$  elements, such that the elements of each subarray  $A[(i-1)k+1..ik]$  are sorted. Your algorithm should run in  $O(n \log k)$  time.
- (b) [2 pts] Given an array  $A[1..n]$  that is already split into  $n/k$  sorted subarrays as in part (a), describe an algorithm that sorts the entire array in  $O(n \log(n/k))$  time.
- (c) [3 pts] Prove that your algorithm from part (a) is optimal.
- (d) [4 pts] Prove that your algorithm from part (b) is optimal.



**6. SAT Reduction**

Suppose you are have a black box that magically solves SAT (the formula satisfiability problem) in constant time. That is, given a boolean formula of variables and logical operators ( $\wedge, \vee, \neg$ ), the black box tells you, in constant time, whether or not the formula can be satisfied. Using this black box, design and analyze a **polynomial-time** algorithm that computes an assignment to the variables that satisfies the formula.

**7. Knapsack**

You're hiking through the woods when you come upon a treasure chest filled with objects. Each object has a different size, and each object has a price tag on it, giving its value. There is no correlation between an object's size and its value. You want to take back as valuable a subset of the objects as possible (in one trip), but also making sure that you will be able to carry it in your knapsack which has a limited size.

In other words, you have an integer capacity  $K$  and a target value  $V$ , and you want to decide whether there is a subset of the objects whose total size is *at most*  $K$  and whose total value is *at least*  $V$ .

- (a) [5 pts] Show that this problem is NP-hard. [Hint: Restate the problem more formally, then reduce from the NP-hard problem PARTITION: Given a set  $S$  of nonnegative integers, is there a partition of  $S$  into disjoint subsets  $A$  and  $B$  (where  $A \cup B = S$ ) whose sums are equal, *i.e.*,  $\sum_{a \in A} a = \sum_{b \in B} b$ .]
- (b) [5 pts] Describe and analyze a dynamic programming algorithm to solve the knapsack problem in  $O(nK)$  time. Prove your algorithm is correct.

# CS 373: Combinatorial Algorithms, Fall 2000

## Homework 0, due August 31, 2000 at the beginning of class

Name:	
Net ID:	Alias:

Neatly print your name (first name first, with no comma), your network ID, and a short alias into the boxes above. **Do not sign your name. Do not write your Social Security number.** Staple this sheet of paper to the top of your homework.

Grades will be listed on the course web site by alias give us, so your alias should not resemble your name or your Net ID. If you don't give yourself an alias, we'll give you one that you won't like.

---

Before you do anything else, read the Homework Instructions and FAQ on the CS 373 course web page (<http://www-courses.cs.uiuc.edu/~cs373/hw/faq.html>), and then check the box below. This web page gives instructions on how to write and submit homeworks—staple your solutions together in order, write your name and netID on every page, don't turn in source code, analyze everything, use good English and good logic, and so forth.

I have read the CS 373 Homework Instructions and FAQ.

---

This homework tests your familiarity with the prerequisite material from CS 173, CS 225, and CS 273—many of these problems have appeared on homeworks or exams in those classes—primarily to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Parberry and Chapters 1–6 of CLR should be sufficient review, but you may want to consult other texts as well.

---

### Required Problems

- Sort the following 25 functions from asymptotically smallest to asymptotically largest, indicating ties if there are any:

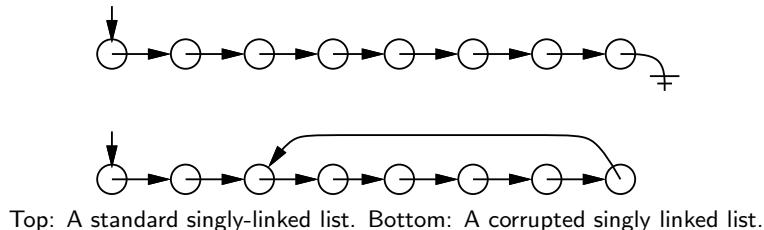
$$\begin{array}{ccccc} 1 & n & n^2 & \lg n & \lg(n \lg n) \\ \lg^* n & \lg^* 2^n & 2^{\lg^* n} & \lg \lg^* n & \lg^* \lg n \\ n^{\lg n} & (\lg n)^n & (\lg n)^{\lg n} & n^{1/\lg n} & n^{\lg \lg n} \\ \log_{1000} n & \lg^{1000} n & \lg^{(1000)} n & \left(1 + \frac{1}{n}\right)^n & n^{1/1000} \end{array}$$

To simplify notation, write  $f(n) \ll g(n)$  to mean  $f(n) = o(g(n))$  and  $f(n) \equiv g(n)$  to mean  $f(n) = \Theta(g(n))$ . For example, the functions  $n^2$ ,  $n$ ,  $\binom{n}{2}$ ,  $n^3$  could be sorted either as  $n \ll n^2 \equiv \binom{n}{2} \ll n^3$  or as  $n \ll \binom{n}{2} \equiv n^2 \ll n^3$ .

2. (a) Prove that any positive integer can be written as the sum of distinct powers of 2. For example:  $42 = 2^5 + 2^3 + 2^1$ ,  $25 = 2^4 + 2^3 + 2^0$ ,  $17 = 2^4 + 2^0$ . [Hint: “Write the number in binary” is *not* a proof; it just restates the problem.]
- (b) Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers—if  $F_n$  appears in the sum, then neither  $F_{n+1}$  nor  $F_{n-1}$  will. For example:  $42 = F_9 + F_6$ ,  $25 = F_8 + F_4 + F_2$ ,  $17 = F_7 + F_4 + F_2$ .
- (c) Prove that *any* integer (positive, negative, or zero) can be written in the form  $\sum_i \pm 3^i$ , where the exponents  $i$  are distinct non-negative integers. For example:  $42 = 3^4 - 3^3 - 3^2 - 3^1$ ,  $25 = 3^3 - 3^1 + 3^0$ ,  $17 = 3^3 - 3^2 - 3^0$ .
3. Solve the following recurrences. State tight asymptotic bounds for each function in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ . You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway just for practice. If no base cases are given, assume something reasonable but nontrivial. Extra credit will be given for more exact solutions.
- (a)  $A(n) = 3A(n/2) + n$
  - (b)  $B(n) = \max_{n/3 < k < 2n/3} (B(k) + B(n - k) + n)$
  - (c)  $C(n) = 4C(\lfloor n/2 \rfloor + 5) + n^2$
  - \*(d)  $D(n) = 2D(n/2) + n/\lg n$
  - \*(e)  $E(n) = \frac{E(n-1)}{E(n-2)}$ , where  $E(1) = 1$  and  $E(2) = 2$ .
4. Penn and Teller have a special deck of fifty-two cards, with no face cards and nothing but clubs—the ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ..., 52 of clubs. (They’re big cards.) Penn shuffles the deck until each each of the  $52!$  possible orderings of the cards is equally likely. He then takes cards one at a time from the top of the deck and gives them to Teller, stopping as soon as he gives Teller the three of clubs.
- (a) On average, how many cards does Penn give Teller?
  - (b) On average, what is the smallest-numbered card that Penn gives Teller?
  - \*(c) On average, what is the largest-numbered card that Penn gives Teller?

[Hint: Solve for an  $n$ -card deck, and then set  $n = 52$ .] Prove that your answers are correct. If you have to appeal to “intuition” or “common sense”, your answers are probably wrong!

5. Suppose you have a pointer to the head of singly linked list. Normally, each node in the list only has a pointer to the next element, and the last node's pointer is `NULL`. Unfortunately, your list might have been corrupted by a bug in somebody else's code<sup>1</sup>, so that the last node has a pointer back to some other node in the list instead.

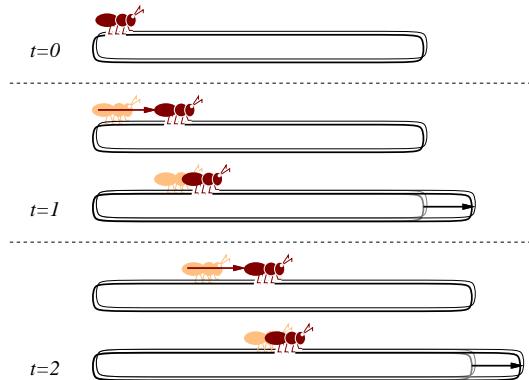


Top: A standard singly-linked list. Bottom: A corrupted singly linked list.

Describe an algorithm<sup>2</sup> that determines whether the linked list is corrupted or not. Your algorithm must not modify the list. For full credit, your algorithm should run in  $O(n)$  time, where  $n$  is the number of nodes in the list, and use  $O(1)$  extra space (not counting the list itself).

6. [This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]

An ant is walking along a rubber band, starting at the left end. Once every second, the ant walks one inch to the right, and then you make the rubber band one inch longer by pulling on the right end. The rubber band stretches uniformly, so stretching the rubber band also pulls the ant to the right. The initial length of the rubber band is  $n$  inches, so after  $t$  seconds, the rubber band is  $n + t$  inches long.



Every second, the ant walks an inch, and then the rubber band is stretched an inch longer.

- (a) How far has the ant moved after  $t$  seconds, as a function of  $n$  and  $t$ ? Set up a recurrence and (for full credit) give an *exact* closed-form solution. [Hint: What fraction of the rubber band's length has the ant walked?]
- \*(b) How long does it take the ant to get to the right end of the rubber band? For full credit, give an answer of the form  $f(n) + \Theta(1)$  for some explicit function  $f(n)$ .

---

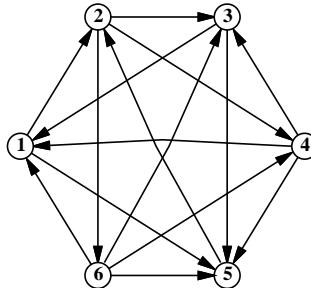
<sup>1</sup>After all, *your* code is always completely 100% bug-free. Isn't that right, Mr. Gates?

<sup>2</sup>Since you've read the Homework Instructions, you know what the phrase "describe an algorithm" means. Right?

## Practice Problems

These remaining practice problems are entirely for your benefit. Don't turn in solutions—we'll just throw them out—but feel free to ask us about these questions during office hours and review sessions. Think of these as potential exam questions (hint, hint).

1. Recall the standard recursive definition of the Fibonacci numbers:  $F_0 = 0$ ,  $F_1 = 1$ , and  $F_n = F_{n-1} + F_{n-2}$  for all  $n \geq 2$ . Prove the following identities for all positive integers  $n$  and  $m$ .
  - (a)  $F_n$  is even if and only if  $n$  is divisible by 3.
  - (b)  $\sum_{i=0}^n F_i = F_{n+2} - 1$
  - (c)  $F_n^2 - F_{n+1}F_{n-1} = (-1)^{n+1}$
  - ★(d) If  $n$  is an integer multiple of  $m$ , then  $F_n$  is an integer multiple of  $F_m$ .
  
2. A *tournament* is a directed graph with exactly one edge between every pair of vertices. (Think of the nodes as players in a round-robin tournament, where each edge points from the winner to the loser.) A *Hamiltonian path* is a sequence of directed edges, joined end to end, that visits every vertex exactly once. Prove that every tournament contains at least one Hamiltonian path.



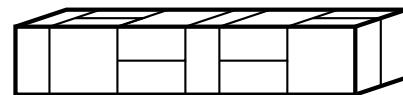
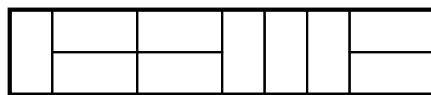
A six-vertex tournament containing the Hamiltonian path  $6 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1$ .

3. (a) Prove the following identity by induction:

$$\binom{2n}{n} = \sum_{k=0}^n \binom{n}{k} \binom{n}{n-k}.$$

- (b) Give a non-inductive combinatorial proof of the same identity, by showing that the two sides of the equation count exactly the same thing in two different ways. There is a correct one-sentence proof.

4. (a) Prove that  $2^{\lceil \lg n \rceil + \lfloor \lg n \rfloor} / n = \Theta(n)$ .  
 (b) Is  $2^{\lfloor \lg n \rfloor} = \Theta(2^{\lceil \lg n \rceil})$ ? Justify your answer.  
 (c) Is  $2^{2^{\lfloor \lg \lg n \rfloor}} = \Theta(2^{2^{\lceil \lg \lg n \rceil}})$ ? Justify your answer.  
 (d) Prove that if  $f(n) = O(g(n))$ , then  $2^{f(n)} = O(2^{g(n)})$ . Justify your answer.  
 (e) Prove that  $f(n) = O(g(n))$  does *not* imply that  $\log(f(n)) = O(\log(g(n)))$ ?.  
 \*(f) Prove that  $\log^k n = o(n^{1/k})$  for any positive integer  $k$ .
5. Solve the following recurrences. State tight asymptotic bounds for each function in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ . You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway just for practice. If no base cases are given, assume something reasonable (but nontrivial). Extra credit will be given for more exact solutions.
- (a)  $A(n) = A(n/2) + n$   
 (b)  $B(n) = 2B(n/2) + n$   
 (c)  $C(n) = \min_{0 < k < n} (C(k) + C(n - k) + 1)$ , where  $C(1) = 1$ .  
 (d)  $D(n) = D(n - 1) + 1/n$   
 \*(e)  $E(n) = 8E(n - 1) - 15E(n - 2) + 1$   
 \*(f)  $F(n) = (n - 1)(F(n - 1) + F(n - 2))$ , where  $F(0) = F(1) = 1$   
 ★(g)  $G(n) = G(n/2) + G(n/4) + G(n/6) + G(n/12) + n$  [Hint:  $\frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \frac{1}{12} = 1$ .]
6. (a) A *domino* is a  $2 \times 1$  or  $1 \times 2$  rectangle. How many different ways are there to completely fill a  $2 \times n$  rectangle with  $n$  dominos? Set up a recurrence relation and give an *exact* closed-form solution.  
 (b) A *slab* is a three-dimensional box with dimensions  $1 \times 2 \times 2$ ,  $2 \times 1 \times 2$ , or  $2 \times 2 \times 1$ . How many different ways are there to fill a  $2 \times 2 \times n$  box with  $n$  slabs? Set up a recurrence relation and give an *exact* closed-form solution.



A  $2 \times 10$  rectangle filled with ten dominos, and a  $2 \times 2 \times 10$  box filled with ten slabs.

7. Professor George O'Jungle has a favorite 26-node binary tree, whose nodes are labeled by letters of the alphabet. The preorder and postorder sequences of nodes are as follows:

preorder: M N H C R S K W T G D X I Y A J P O E Z V B U L Q F  
 postorder: C W T K S G R H D N A O E P J Y Z I B Q L F U V X M

Draw Professor O'Jungle's binary tree, and give the inorder sequence of nodes.

8. Alice and Bob each have a fair  $n$ -sided die. Alice rolls her die once. Bob then repeatedly throws his die until he rolls a number at least as big as the number Alice rolled. Each time Bob rolls, he pays Alice \$1. (For example, if Alice rolls a 5, and Bob rolls a 4, then a 3, then a 1, then a 5, the game ends and Alice gets \$4. If Alice rolls a 1, then no matter what Bob rolls, the game will end immediately, and Alice will get \$1.)

*Exactly* how much money does Alice expect to win at this game? Prove that your answer is correct. If you have to appeal to “intuition” or “common sense”, your answer is probably wrong!

9. Prove that for any nonnegative parameters  $a$  and  $b$ , the following algorithms terminate and produce identical output.

<u>SLOWEUCLID(<math>a, b</math>) :</u>
if $b > a$
return SLOWEUCLID( $b, a$ )
else if $b = 0$
return $a$
else
return SLOWEUCLID( $b, a - b$ )

<u>FASTEUCLID(<math>a, b</math>) :</u>
if $b = 0$
return $a$
else
return FASTEUCLID( $b, a \bmod b$ )

# CS 373: Combinatorial Algorithms, Fall 2000

Homework 1 (due September 12, 2000 at midnight)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

---

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U,  $\frac{3}{4}$ , or 1, respectively. Staple this sheet to the top of your homework.

---

## Required Problems

1. Suppose we want to display a paragraph of text on a computer screen. The text consists of  $n$  words, where the  $i$ th word is  $p_i$  pixels wide. We want to break the paragraph into several lines, each exactly  $P$  pixels long. Depending on which words we put on each line, we will need to insert different amounts of white space between the words. The paragraph should be fully justified, meaning that the first word on each line starts at its leftmost pixel, and *except for the last line*, the last character on each line ends at its rightmost pixel. There must be at least one pixel of whitespace between any two words on the same line.

Define the *slop* of a paragraph layout as the sum over all lines, *except the last*, of the cube of the number of extra white-space pixels in each line (not counting the one pixel required between every adjacent pair of words). Specifically, if a line contains words  $i$  through  $j$ , then the amount of extra white space on that line is  $P - j + i - \sum_{k=i}^j p_k$ . Describe a dynamic programming algorithm to print the paragraph with minimum slop.

2. Consider the following sorting algorithm:

```

STUPIDSORT( $A[0..n-1]$ ) :
  if  $n = 2$  and  $A[0] > A[1]$ 
    swap  $A[0] \leftrightarrow A[1]$ 
  else if  $n > 2$ 
     $m \leftarrow \lceil 2n/3 \rceil$ 
    STUPIDSORT( $A[0..m-1]$ )
    STUPIDSORT( $A[n-m..n-1]$ )
    STUPIDSORT( $A[0..m-1]$ )

```

- (a) Prove that STUPIDSORT actually sorts its input.
  - (b) Would the algorithm still sort correctly if we replaced the line  $m \leftarrow \lceil 2n/3 \rceil$  with  $m \leftarrow \lfloor 2n/3 \rfloor$ ? Justify your answer.
  - (c) State a recurrence (including the base case(s)) for the number of comparisons executed by STUPIDSORT.
  - (d) Solve the recurrence, and prove that your solution is correct. [Hint: Ignore the ceiling.] Does the algorithm deserve its name?
  - \*(e) Show that the number of swaps executed by STUPIDSORT is at most  $\binom{n}{2}$ .
3. The following randomized algorithm selects the  $r$ th smallest element in an unsorted array  $A[1..n]$ . For example, to find the smallest element, you would call RANDOMSELECT( $A, 1$ ); to find the median element, you would call RANDOMSELECT( $A, \lfloor n/2 \rfloor$ ). Recall from lecture that PARTITION splits the array into three parts by comparing the pivot element  $A[p]$  to every other element of the array, using  $n - 1$  comparisons altogether, and returns the new index of the pivot element.

```

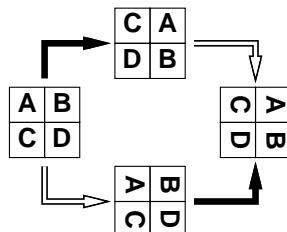
RANDOMSELECT( $A[1..n], r$ ) :
   $p \leftarrow \text{RANDOM}(1, p)$ 
   $k \leftarrow \text{PARTITION}(A[1..n], p)$ 
  if  $r < k$ 
    return RANDOMSELECT( $A[1..k-1], r$ )
  else if  $r > k$ 
    return RANDOMSELECT( $A[k+1..n], r-k$ )
  else
    return  $A[k]$ 

```

- (a) State a recurrence for the expected running time of RANDOMSELECT, as a function of  $n$  and  $r$ .
- (b) What is the *exact* probability that RANDOMSELECT compares the  $i$ th smallest and  $j$ th smallest elements in the input array? The correct answer is a simple function of  $i$ ,  $j$ , and  $r$ . [Hint: Check your answer by trying a few small examples.]
- \*(c) What is the expected running time of RANDOMSELECT, as a function of  $n$  and  $r$ ? You can use either the recurrence from part (a) or the probabilities from part (b). For extra credit, give the *exact* expected number of comparisons.
- (d) What is the expected number of times that RANDOMSELECT calls itself recursively?

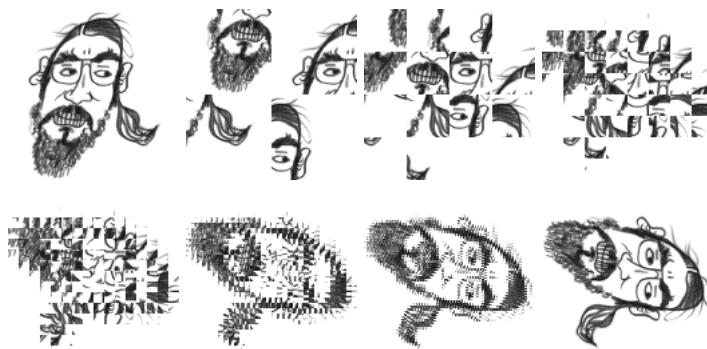
4. Some graphics hardware includes support for an operation called *blit*, or **block transfer**, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

Suppose we want to rotate an  $n \times n$  pixelmap 90° clockwise. One way to do this is to split the pixelmap into four  $n/2 \times n/2$  blocks, move each block to its proper position using a sequence of five blits, and then recursively rotate each block. Alternately, we can first recursively rotate the blocks and blit them into place afterwards.



Two algorithms for rotating a pixelmap.  
Black arrows indicate blitting the blocks into place.  
White arrows indicate recursively rotating the blocks.

The following sequence of pictures shows the first algorithm (blit then recurse) in action.



In the following questions, assume  $n$  is a power of two.

- (a) Prove that both versions of the algorithm are correct. [Hint: If you exploit all the available symmetries, your proof will only be a half of a page long.]
  - (b) *Exactly* how many blits does the algorithm perform?
  - (c) What is the algorithm's running time if a  $k \times k$  blit takes  $O(k^2)$  time?
  - (d) What if a  $k \times k$  blit takes only  $O(k)$  time?

5. The traditional Devonian/Cornish drinking song “The Barley Mow” has the following pseudolyrics<sup>1</sup>, where  $\text{container}[i]$  is the name of a container that holds  $2^i$  ounces of beer.<sup>2</sup>

```
BARLEYMOW( $n$ ):
  "Here's a health to the barley-mow, my brave boys,"
  "Here's a health to the barley-mow!"

  "We'll drink it out of the jolly brown bowl,"
  "Here's a health to the barley-mow!"
  "Here's a health to the barley-mow, my brave boys,"
  "Here's a health to the barley-mow!"

  for  $i \leftarrow 1$  to  $n$ 
    "We'll drink it out of the container[ $i$ ], boys,"
    "Here's a health to the barley-mow!"
    for  $j \leftarrow i$  downto 1
      "The container[j],"
      "And the jolly brown bowl!"
      "Here's a health to the barley-mow!"
      "Here's a health to the barley-mow, my brave boys,"
      "Here's a health to the barley-mow!"
```

- (a) Suppose each container name  $\text{container}[i]$  is a single word, and you can sing four words a second. How long would it take you to sing  $\text{BARLEYMOW}(n)$ ? (Give a tight asymptotic bound.)
- (b) If you want to sing this song for  $n > 20$ , you’ll have to make up your own container names, and to avoid repetition, these names will get progressively longer as  $n$  increases<sup>3</sup>. Suppose  $\text{container}[n]$  has  $\Theta(\log n)$  syllables, and you can sing six syllables per second. Now how long would it take you to sing  $\text{BARLEYMOW}(n)$ ? (Give a tight asymptotic bound.)
- (c) Suppose each time you mention the name of a container, you drink the corresponding amount of beer: one ounce for the jolly brown bowl, and  $2^i$  ounces for each  $\text{container}[i]$ . Assuming for purposes of this problem that you are at least 21 years old, *exactly* how many ounces of beer would you drink if you sang  $\text{BARLEYMOW}(n)$ ? (Give an *exact* answer, not just an asymptotic bound.)

---

<sup>1</sup>Pseudolyrics are to lyrics as pseudocode is to code.

<sup>2</sup>One version of the song uses the following containers: nipperkin, gill pot, half-pint, pint, quart, pottle, gallon, half-anker, anker, firkin, half-barrel, barrel, hogshead, pipe, well, river, and ocean. Every container in this list is twice as big as its predecessor, except that a firkin is actually 2.25 ankers, and the last three units are just silly.

<sup>3</sup>“We'll drink it out of the hemisemidemiyottapint, boys!”

6. [This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]

A company is planning a party for its employees. The employees in the company are organized into a strict hierarchy, that is, a tree with the company president at the root. The organizers of the party have assigned a real number to each employee measuring how ‘fun’ the employee is. In order to keep things social, there is one restriction on the guest list: an employee cannot attend the party if their immediate supervisor is present. On the other hand, the president of the company *must* attend the party, even though she has a negative fun rating; it’s her company, after all. Give an algorithm that makes a guest list for the party that maximizes the sum of the ‘fun’ ratings of the guests.

## Practice Problems

1. Give an  $O(n^2)$  algorithm to find the longest increasing subsequence of a sequence of numbers. The elements of the subsequence need not be adjacent in the sequence. For example, the sequence  $\langle 1, 5, 3, 2, 4 \rangle$  has longest increasing subsequence  $\langle 1, 3, 4 \rangle$ .
  
2. You are at a political convention with  $n$  delegates. Each delegate is a member of exactly one political party. It is impossible to tell which political party a delegate belongs to. However, you can check whether any two delegates are in the *same* party or not by introducing them to each other. (Members of the same party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.)
  - (a) Suppose a majority (more than half) of the delegates are from the same political party. Give an efficient algorithm that identifies a member of the majority party.
  - (b) Suppose exactly  $k$  political parties are represented at the convention and one party has a *plurality*: more delegates belong to that party than to any other. Present a practical procedure to pick a person from the plurality party as parsimoniously as possible. (Please.)
  
3. Give an algorithm that finds the *second* smallest of  $n$  elements in at most  $n + \lceil \lg n \rceil - 2$  comparisons. [Hint: divide and conquer to find the smallest; where is the second smallest?]
  
4. Suppose that you have an array of records whose keys to be sorted consist only of 0’s and 1’s. Give a simple, linear-time  $O(n)$  algorithm to sort the array in place (using storage of no more than constant size in addition to that of the array).

5. Consider the problem of making change for  $n$  cents using the least number of coins.
- Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.
  - Suppose that the available coins have the values  $c^0, c^1, \dots, c^k$  for some integers  $c > 1$  and  $k \geq 1$ . Show that the obvious greedy algorithm always yields an optimal solution.
  - Give a set of 4 coin values for which the greedy algorithm does not yield an optimal solution.
  - Describe a dynamic programming algorithm that yields an optimal solution for an arbitrary set of coin values.
  - Suppose we have only two types of coins whose values  $a$  and  $b$  are relatively prime. Prove that any value of greater than  $(a - 1)(b - 1)$  can be made with these two coins.
- ★(f) For only three coins  $a, b, c$  whose greatest common divisor is 1, give an algorithm to determine the smallest value  $n$  such that change *can* be given for all values greater than  $n$ . [Note: this problem is currently unsolved for more than four coins!]
6. Suppose you have a subroutine that can find the median of a set of  $n$  items (i.e., the  $\lfloor n/2 \rfloor$  smallest) in  $O(n)$  time. Give an algorithm to find the  $k$ th biggest element (for arbitrary  $k$ ) in  $O(n)$  time.
7. You're walking along the beach and you stub your toe on something in the sand. You dig around it and find that it is a treasure chest full of gold bricks of different (integral) weight. Your knapsack can only carry up to weight  $n$  before it breaks apart. You want to put as much in it as possible without going over, but you *cannot* break the gold bricks up.
- Suppose that the gold bricks have the weights  $1, 2, 4, 8, \dots, 2^k$ ,  $k \geq 1$ . Describe and prove correct a greedy algorithm that fills the knapsack as much as possible without going over.
  - Give a set of 3 weight values for which the greedy algorithm does *not* yield an optimal solution and show why.
  - Give a dynamic programming algorithm that yields an optimal solution for an arbitrary set of gold brick values.

# CS 373: Combinatorial Algorithms, Fall 2000

## Homework 2 (due September 28, 2000 at midnight)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

---

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U,  $\frac{3}{4}$ , or 1, respectively. Staple this sheet to the top of your homework.

---

### Required Problems

#### 1. Faster Longest Increasing Subsequence (15 pts)

Give an  $O(n \log n)$  algorithm to find the longest increasing subsequence of a sequence of numbers. [Hint: In the dynamic programming solution, you don't really have to look back at all previous items. There was a practice problem on HW 1 that asked for an  $O(n^2)$  algorithm for this. If you are having difficulty, look at the HW 1 solutions.]

#### 2. SELECT(A, k) (10 pts)

Say that a binary search tree is *augmented* if every node  $v$  also stores  $|v|$ , the size of its subtree.

- Show that a rotation in an augmented binary tree can be performed in constant time.
- Describe an algorithm `SCAPEGOATSELECT( $k$ )` that selects the  $k$ th smallest item in an augmented scapegoat tree in  $O(\log n)$  worst-case time.
- Describe an algorithm `SPLAYSELECT( $k$ )` that selects the  $k$ th smallest item in an augmented splay tree in  $O(\log n)$  amortized time.

- (d) Describe an algorithm  $\text{TREAPSELECT}(k)$  that selects the  $k$ th smallest item in an augmented treap in  $O(\log n)$  *expected* time.

[Hint: The answers for (b), (c), and (d) are almost exactly the same!]

3. Scapegoat trees (15 pts)

- (a) Prove that only one subtree gets rebalanced in a scapegoat tree insertion.
- (b) Prove that  $I(v) = 0$  in every node of a perfectly balanced tree. (Recall that  $I(v) = \max\{0, |T| - |s| - 1\}$ , where  $T$  is the child of greater height and  $s$  the child of lesser height, and  $|v|$  is the number of nodes in subtree  $v$ . A perfectly balanced tree has two perfectly balanced subtrees, each with as close to half the nodes as possible.)
- \*(c) Show that you can rebuild a fully balanced binary tree from an unbalanced tree in  $O(n)$  time using only  $O(\log n)$  additional memory. For 5 extra credit points, use only  $O(1)$  additional memory.

4. Memory Management (10 pts)

Suppose we can insert or delete an element into a hash table in constant time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:

- After an insertion, if the table is more than  $3/4$  full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
- After a deletion, if the table is less than  $1/4$  full, we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do *not* use the potential method—it makes the problem much too hard!

5. Fibonacci Heaps:  $\text{SECONDMIN}$  (10 pts)

- (a) Implement  $\text{SECONDMIN}$  by using a Fibonacci heap as a black box. Remember to justify its correctness and running time.
- \*(b) Modify the Fibonacci Heap data structure to implement the  $\text{SECONDMIN}$  operation in constant time, without degrading the performance of any other Fibonacci heap operation.

## Practice Problems

### 1. Amortization

- (a) Modify the binary double-counter (see class notes Sept 12) to support a new operation SIGN, which determines whether the number being stored is positive, negative, or zero, in constant time. The amortized time to increment or decrement the counter should still be a constant.

[Hint: Suppose  $p$  is the number of significant bits in  $P$ , and  $n$  is the number of significant bits in  $N$ . For example, if  $P = 17 = 10001_2$  and  $N = 0$ , then  $p = 5$  and  $n = 0$ . Then  $p - n$  always has the same sign as  $P - N$ . Assume you can update  $p$  and  $n$  in  $O(1)$  time.]

- \*(b) Do the same but now you can't assume that  $p$  and  $n$  can be updated in  $O(1)$  time.

### \*2. Amortization

Suppose instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of ‘fits’, where the  $i$ th least significant fit indicates whether the sum includes the  $i$ th Fibonacci number  $F_i$ . For example, the fit string 101110 represents the number  $F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14$ . Describe algorithms to increment and decrement a fit string in constant amortized time. [Hint: Most numbers can be represented by more than one fit string. This is not the same representation as on Homework 0!]]

### 3. Rotations

- (a) Show that it is possible to transform any  $n$ -node binary search tree into any other  $n$ -node binary search tree using at most  $2n - 2$  rotations.
- \*(b) Use fewer than  $2n - 2$  rotations. Nobody knows how few rotations are required in the worst case. There is an algorithm that can transform any tree to any other in at most  $2n - 6$  rotations, and there are pairs of trees that are  $2n - 10$  rotations apart. These are the best bounds known.
4. Give an efficient implementation of the operation CHANGEKEY( $x, k$ ), which changes the key of a node  $x$  in a Fibonacci heap to the value  $k$ . The changes you make to Fibonacci heap data structure to support your implementation should not affect the amortized running time of any other Fibonacci heap operations. Analyze the amortized running time of your implementation for cases in which  $k$  is greater than, less than, or equal to  $\text{key}[x]$ .

### 5. Detecting overlap

- (a) You are given a list of ranges represented by min and max (e.g., [1,3], [4,5], [4,9], [6,8], [7,10]). Give an  $O(n \log n)$ -time algorithm that decides whether or not a set of ranges contains a pair that overlaps. You need not report all intersections. If a range completely covers another, they are overlapping, even if the boundaries do not intersect.

- (b) You are given a list of rectangles represented by min and max  $x$ - and  $y$ -coordinates. Give an  $O(n \log n)$ -time algorithm that decides whether or not a set of rectangles contains a pair that overlaps (with the same qualifications as above). [Hint: sweep a vertical line from left to right, performing some processing whenever an end-point is encountered. Use a balanced search tree to maintain any extra info you might need.]

6. Comparison of Amortized Analysis Methods

A sequence of  $n$  operations is performed on a data structure. The  $i$ th operation costs  $i$  if  $i$  is an exact power of 2, and 1 otherwise. That is operation  $i$  costs  $f(i)$ , where:

$$f(i) = \begin{cases} i, & i = 2^k, \\ 1, & \text{otherwise} \end{cases}$$

Determine the amortized cost per operation using the following methods of analysis:

- (a) Aggregate method
- (b) Accounting method
- \*(c) Potential method

# CS 373: Combinatorial Algorithms, Fall 2000

Homework 3 (due October 17, 2000 at midnight)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

---

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U,  $\frac{3}{4}$ , or 1, respectively. Staple this sheet to the top of your homework.

---

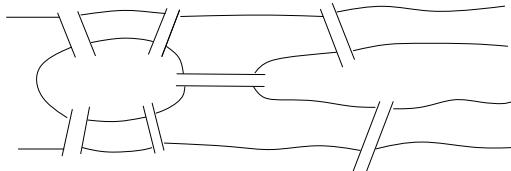
## Required Problems

1. Suppose you have to design a dictionary that holds 2048 items.
  - (a) How many probes are used for an unsuccessful search if the dictionary is implemented as a sorted array? Assume the use of Binary Search.
  - (b) How large a hashtable do you need if your goal is to have 2 as the expected number of probes for an unsuccessful search?
  - (c) How much more space is needed by the hashtable compared to the sorted array? Assume that each pointer in a linked list takes 1 word of storage.
2. In order to facilitate recompiling programs from multiple source files when only a small number of files have been updated, there is a UNIX utility called ‘make’ that only recompiles those files that were changed after the most recent compilation, *and* any intermediate files in the compilation that depend on those that were changed. A Makefile is typically composed of a list of source files that must be compiled. Each of these source files is dependent on some of

the other files which are listed. Thus a source file must be recompiled if a file on which it depends is changed.

Assuming you have a list of which files have been recently changed, as well as a list for each source file of the files on which it depends, design an algorithm to recompile only those necessary. Don't worry about the details of parsing a Makefile.

3. A person wants to fly from city  $A$  to city  $B$  in the shortest possible time. She turns to the traveling agent who knows all the departure and arrival times of all the flights on the planet. Give an algorithm that will allow the agent to choose a route with the minimum total travel time—initial takeoff to final landing, including layovers. [*Hint: Modify the data and call a shortest-path algorithm.*]
  
4. During the eighteenth century the city of Königsberg in East Prussia was divided into four sections by the Pregel river. Seven bridges connected these regions, as shown below. It was said that residents spent their Sunday walks trying to find a way to walk about the city so as to cross each bridge exactly once and then return to their starting point.



- (a) Show how the residents of the city could accomplish such a walk or prove no such walk exists.
- (b) Given any undirected graph  $G = (V, E)$ , give an algorithm that finds a cycle in the graph that visits every edge exactly once, or says that it can't be done.
  
5. Suppose you have a graph  $G$  and an MST of that graph (i.e. the MST has already been constructed).
  - (a) Give an algorithm to update the MST when an edge is added to  $G$ .
  - (b) Give an algorithm to update the MST when an edge is deleted from  $G$ .
  - (c) Give an algorithm to update the MST when a vertex (and possibly edges to it) is added to  $G$ .
  
6. [*This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.*]

You are given an unlimited number of each of  $n$  different types of envelopes. The dimensions of envelope type  $i$  are  $x_i \times y_i$ . In nesting envelopes inside one another, you can place envelope  $A$  inside envelope  $B$  if and only if the dimensions  $A$  are *strictly smaller* than the dimensions of  $B$ . Design and analyze an algorithm to determine the largest number of envelopes that can be nested inside one another.

## Practice Problems

- ★1. Let the hash function for a table of size  $m$  be

$$h(x) = \lfloor Ax \rfloor \bmod m$$

where  $A = \hat{\phi} = \frac{\sqrt{5}-1}{2}$ . Show that this gives the best possible spread, i.e. if the  $x$  are hashed in order,  $x+1$  will be hashed in the largest remaining contiguous interval.

2. The incidence matrix of an undirected graph  $G = (V, E)$  is a  $|V| \times |E|$  matrix  $B = (b_{ij})$  such that

$$b_{ij} = [(i, j) \in E] = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{if } (i, j) \notin E. \end{cases}$$

- (a) Describe what all the entries of the matrix product  $BB^T$  represent ( $B^T$  is the matrix transpose).
  - (b) Describe what all the entries of the matrix product  $B^TB$  represent.
  - ★(c) Let  $C = BB^T - 2A$ , where  $A$  is the adjacency matrix of  $G$ , with zeroes on the diagonal. Let  $C'$  be  $C$  with the first row and column removed. Show that  $\det C'$  is the number of spanning trees.
3. (a) Give an  $O(V)$  algorithm to decide whether a directed graph contains a *sink* in an adjacency matrix representation. A sink is a vertex with in-degree  $V - 1$ .
- (b) An undirected graph is a scorpion if it has a vertex of degree 1 (the sting) connected to a vertex of degree two (the tail) connected to a vertex of degree  $V - 2$  (the body) connected to the other  $V - 3$  vertices (the feet). Some of the feet may be connected to other feet.  
Design an algorithm that decides whether a given adjacency matrix represents a scorpion by examining only  $O(V)$  of the entries.
- (c) Show that it is impossible to decide whether  $G$  has at least one edge in  $O(V)$  time.
4. Given an *undirected* graph  $G = (V, E)$ , and a weight function  $f : E \rightarrow \mathbb{R}$  on the edges, give an algorithm that finds (in time polynomial in  $V$  and  $E$ ) a cycle of smallest weight in  $G$ .
5. Let  $G = (V, E)$  be a graph with  $n$  vertices. A *simple path* of  $G$ , is a path that does not contain the same vertex twice. Use dynamic programming to design an algorithm (not polynomial time) to find a simple path of maximum length in  $G$ . Hint: It can be done in  $O(n^c 2^n)$  time, for some constant  $c$ .
6. Suppose all edge weights in a graph  $G$  are equal. Give an algorithm to compute a minimum spanning tree of  $G$ .

7. Give an algorithm to construct a *transitive reduction* of a directed graph  $G$ , i.e. a graph  $G^{TR}$  with the fewest edges (but with the same vertices) such that there is a path from  $a$  to  $b$  in  $G$  iff there is also such a path in  $G^{TR}$ .

8. (a) What is  $5^{29^5^0 + 23^4^1 + 17^3^2 + 11^2^3 + 5^1^4} \pmod{6}$ ?
- (b) What is the capital of Nebraska? Hint: It is not Omaha. It is named after a famous president of the United States that was not George Washington. The distance from the Earth to the Moon averages roughly 384,000 km.

# CS 373: Combinatorial Algorithms, Fall 2000

## Homework 4 (due October 26, 2000 at midnight)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

---

Homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grad. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U,  $\frac{3}{4}$ , or 1, respectively. Staple this sheet to the top of your homework.

---

### Required Problems

- (10 points) A certain algorithms professor once claimed that the height of an  $n$ -node Fibonacci heap is of height  $O(\log n)$ . Disprove his claim by showing that for a positive integer  $n$ , a sequence of Fibonacci heap operations that creates a Fibonacci heap consisting of just one tree that is a (downward) linear chain of  $n$  nodes.
- (20 points) *Fibonacci strings* are defined as follows:

$$\begin{aligned}F_1 &= b \\F_2 &= a \\F_n &= F_{n-1}F_{n-2} \quad \text{for all } n > 2\end{aligned}$$

where the recursive rule uses concatenation of strings, so  $F_3 = ab$ ,  $F_4 = aba$ , and so on. Note that the length of  $F_n$  is the  $n$ th Fibonacci number.

- Prove that in any Fibonacci string there are no two b's adjacent and no three a's.

- (b) Give the unoptimized and optimized failure function for  $F_7$ .
- (c) Prove that, in searching for the Fibonacci string  $F_k$ , the unoptimized KMP algorithm may shift  $\lceil k/2 \rceil$  times on the same text character. In other words, prove that there is a chain of failure links  $j \rightarrow \text{fail}[j] \rightarrow \text{fail}[\text{fail}[j]] \rightarrow \dots$  of length  $\lceil k/2 \rceil$ , and find an example text  $T$  that would cause KMP to traverse this entire chain on the same position in the text.
- (d) What happens here when you use the optimized prefix function? Explain.
3. (10 points) Show how to extend the Rabin-Karp fingerprinting method to handle the problem of looking for a given  $m \times m$  pattern in an  $n \times n$  array of characters. The pattern may be shifted horizontally and vertically, but it may not be rotated.
4. (10 points)
- A *cyclic rotation* of a string is obtained by chopping off a prefix and gluing it at the end of the string. For example, ALGORITHM is a cyclic shift of RITHMALGO. Describe and analyze an algorithm that determines whether one string  $P[1..m]$  is a cyclic rotation of another string  $T[1..n]$ .
  - Describe and analyze an algorithm that decides, given any two binary trees  $P$  and  $T$ , whether  $P$  equals a subtree of  $T$ . We want an algorithm that compares the *shapes* of the trees. There is no data stored in the nodes, just pointers to the left and right children. [Hint: First transform both trees into strings.]
- 
- $P$
- $T$
- $P$  occurs exactly once as a subtree of  $T$ .
5. (10 points) [This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]

Refer to the notes for lecture 11 for this problem. The GENERICSSSP algorithm described in class can be implemented using a stack for the ‘bag’. Prove that the resulting algorithm can be forced to perform in  $\Omega(2^n)$  relaxation steps. To do this, you need to describe, for any positive integer  $n$ , a specific weighted directed  $n$ -vertex graph that forces this exponential behavior. The easiest way to describe such a family of graphs is using an *algorithm*!

## Practice Problems

### 1. String matching with wild-cards

Suppose you have an alphabet for patterns that includes a ‘gap’ or wild-card character; any length string of any characters can match this additional character. For example if ‘\*’ is the wild-card, then the pattern foo\*bar\*nad can be found in foofoowangbarnad. Modify the computation of the prefix function to correctly match strings using KMP.

### 2. Prove that there is no comparison sort whose running time is linear for at least $1/2$ of the $n!$ inputs of length $n$ . What about at least $1/n$ ? What about at least $1/2^n$ ?

### 3. Prove that $2n - 1$ comparisons are necessary in the worst case to merge two sorted lists containing $n$ elements each.

### 4. Find asymptotic upper and lower bounds to $\lg(n!)$ without Stirling’s approximation (Hint: use integration).

### 5. Given a sequence of $n$ elements of $n/k$ blocks ( $k$ elements per block) all elements in a block are less than those to the right in sequence, show that you cannot have the whole sequence sorted in better than $\Omega(n \lg k)$ . Note that the entire sequence would be sorted if each of the $n/k$ blocks were individually sorted in place. Also note that combining the lower bounds for each block is not adequate (that only gives an upper bound).

### 6. Show how to find the occurrences of pattern $P$ in text $T$ by computing the prefix function of the string $PT$ (the concatenation of $P$ and $T$ ).

### 7. Lower Bounds on Adjacency Matrix Representations of Graphs

#### (a) Prove that the time to determine if an undirected graph has a cycle is $\Omega(V^2)$ .

#### (b) Prove that the time to determine if there is a path between two nodes in an undirected graph is $\Omega(V^2)$ .

# CS 373: Combinatorial Algorithms, Fall 2000

## Homework 1 (due November 16, 2000 at midnight)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

---

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

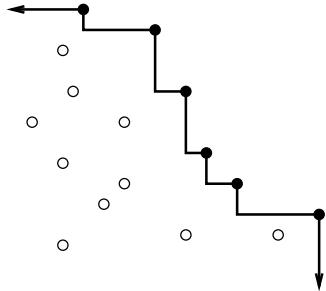
Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U,  $\frac{3}{4}$ , or 1, respectively. Staple this sheet to the top of your homework.

---

### Required Problems

1. Give an  $O(n^2 \log n)$  algorithm to determine whether any three points of a set of  $n$  points are collinear. Assume two dimensions and *exact* arithmetic.
  
2. We are given an array of  $n$  bits, and we want to determine if it contains two consecutive 1 bits. Obviously, we can check every bit, but is this always necessary?
  - (a) (4 pts) Show that when  $n \bmod 3 = 0$  or 2, we must examine every bit in the array. That is, give an adversary strategy that forces any algorithm to examine every bit when  $n = 2, 3, 5, 6, 8, 9, \dots$ .
  - (b) (4 pts) Show that when  $n = 3k + 1$ , we only have to examine  $n - 1$  bits. That is, describe an algorithm that finds two consecutive 1s or correctly reports that there are none after examining at most  $n - 1$  bits, when  $n = 1, 4, 7, 10, \dots$ .
  - (c) (2 pts) How many  $n$ -bit strings are there with two consecutive ones? For which  $n$  is this number even or odd?

3. You are given a set of points in the plane. A point is *maximal* if there is no other point both above and to the right. The subset of maximal points of points then forms a *staircase*.

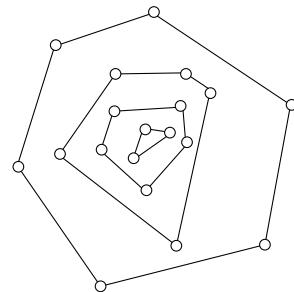


The staircase of a set of points. Maximal points are black.

- (a) (0 pts) Prove that maximal points are *not* necessarily on the convex hull.
- (b) (6 pts) Give an  $O(n \log n)$  algorithm to find the maximal points.
- (c) (4 pts) Assume that points are chosen uniformly at random within a rectangle. What is the average number of maximal points? Justify. Hint: you will be able to give an exact answer rather than just asymptotics. You have seen the same analysis before.

4. Given a set  $Q$  of points in the plane, define the *convex layers* of  $Q$  inductively as follows: The first convex layer of  $Q$  is just the convex hull of  $Q$ . For all  $i > 1$ , the  $i$ th convex layer is the convex hull of  $Q$  after the vertices of the first  $i - 1$  layers have been removed.

Give an  $O(n^2)$ -time algorithm to find all convex layers of a given set of  $n$  points.



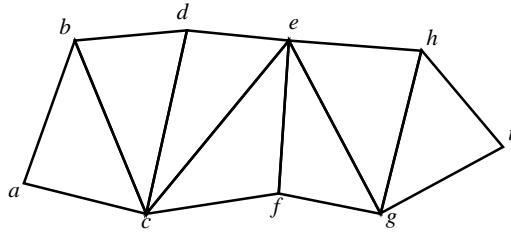
A set of points with four convex layers.

5. Prove that finding the second smallest of  $n$  elements takes  $n + \lceil \lg n \rceil - 2$  comparisons in the worst case. Prove for both upper and lower bounds. Hint: find the (first) smallest using an elimination tournament.

6. [This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]

Almost all computer graphics systems, at some level, represent objects as collections of triangles. In order to minimize storage space and rendering time, many systems allow objects to be stored as a set of *triangle strips*. A triangle strip is a sequence of vertices  $\langle v_1, v_2, \dots, v_k \rangle$ , where each contiguous triple of vertices  $v_i, v_{i+1}, v_{i+2}$  represents a triangle. As the rendering system reads the sequence of vertices and draws the triangles, it keeps the two most recent vertices in a cache.

Some systems allow triangle strips to contain *swaps*: special flags indicating that the order of the two cached vertices should be reversed. For example, the triangle strip  $\langle a, b, c, d, \text{swap}, e, f, \text{swap}, g, h, i \rangle$  represents the sequence of triangles  $(a, b, c), (b, c, d), (d, c, e), (c, e, f), (f, e, g)$ .



Two triangle strips are *disjoint* if they share no triangles (although they may share vertices). The *length* of a triangle strip is the length of its vertex sequence, including swaps; for example, the example strip above has length 11. A *pure* triangle strip is one with no swaps. The adjacency graph of a triangle strip is a simple path. If the strip is pure, this path alternates between left and right turns.

Suppose you are given a set  $S$  of interior-disjoint triangles whose adjacency graph is a tree. (In other words,  $S$  is a triangulation of a simple polygon.) Describe a linear-time algorithm to decompose  $S$  into a set of disjoint triangle strips of minimum total length.

## Practice Problems

1. Consider the following generic recurrence for convex hull algorithms that divide and conquer:

$$T(n, h) = T(n_1, h_1) + T(n_2, h_2) + O(n)$$

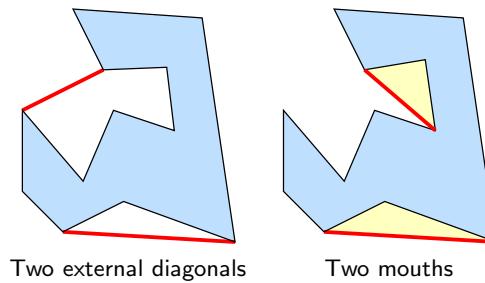
where  $n \geq n_1 + n_2$ ,  $h = h_1 + h_2$  and  $n \geq h$ . This means that the time to compute the convex hull is a function of both  $n$ , the number of input points, and  $h$ , the number of convex hull vertices. The splitting and merging parts of the divide-and-conquer algorithm take  $O(n)$  time. When  $n$  is a constant,  $T(n, h) = O(1)$ , but when  $h$  is a constant,  $T(n, h) = O(n)$ . Prove that for both of the following restrictions, the solution to the recurrence is  $O(n \log h)$ :

- (a)  $h_1, h_2 < \frac{3}{4}h$
- (b)  $n_1, n_2 < \frac{3}{4}n$

2. Circle Intersection

Give an  $O(n \log n)$  algorithm to test whether any two circles in a set of size  $n$  intersect.

3. Basic polygon computations (assume *exact* arithmetic)
- Intersection: Extend the basic algorithm to determine if two line segments intersect by taking care of *all* degenerate cases.
  - Simplicity: Give an  $O(n \log n)$  algorithm to determine whether an  $n$ -vertex polygon is simple.
  - Area: Give an algorithm to compute the area of a simple  $n$ -polygon (not necessarily convex) in  $O(n)$  time.
  - Inside: Give an algorithm to determine whether a point is within a simple  $n$ -polygon (not necessarily convex) in  $O(n)$  time.
4. We are given the set of points one point at a time. After receiving each point, we must compute the convex hull of all those points so far. Give an algorithm to solve this problem in  $O(n^2)$  total time. (We could obviously use Graham's scan  $n$  times for an  $O(n^2 \log n)$ -time algorithm). Hint: How do you maintain the convex hull?
5. \*(a) Given an  $n$ -polygon and a point outside the polygon, give an algorithm to find a tangent.  
(b) Suppose you have found both tangents. Give an algorithm to remove the points from the polygon that are within the angle formed by the tangents (as segments!) and the opposite side of the polygon.  
(c) Use the above to give an algorithm to compute the convex hull on-line in  $O(n \log n)$
6. (a) A pair of polygon vertices defines an *external diagonal* if the line segment between them is completely outside the polygon. Show that every nonconvex polygon has at least one external diagonal.  
(b) Three consecutive polygon vertices  $p, q, r$  form a *mouth* if  $p$  and  $r$  define an external diagonal. Show that every nonconvex polygon has at least one mouth.



7. A group of  $n$  ghostbusters is battling  $n$  ghosts. Each ghostbuster can shoot a single energy beam at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits the ghost. The ghostbusters all fire at the same time and no two energy beams may cross. The positions of the ghosts and ghostbusters are fixed points in the plane.
- Prove that for any configuration of ghosts and ghostbusters, there is such a non-crossing matching. (Assume that no three points are collinear.)

- (b) Show that there is a line passing through one ghostbuster and one ghost such that the number of ghostbusters on one side of the line equals the number of ghosts on the same side. Give an efficient algorithm to find such a line.
- (c) Give an efficient divide and conquer algorithm to pair ghostbusters and ghosts so that no two streams cross.

# CS 373: Combinatorial Algorithms, Fall 2000

## Homework 6 (due December 7, 2000 at midnight)

Name:		
Net ID:	Alias:	<input type="checkbox"/> U <input checked="" type="checkbox"/> <input type="checkbox"/> 1

Name:		
Net ID:	Alias:	<input type="checkbox"/> U <input checked="" type="checkbox"/> <input type="checkbox"/> 1

Name:		
Net ID:	Alias:	<input type="checkbox"/> U <input checked="" type="checkbox"/> <input type="checkbox"/> 1

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U,  $\frac{3}{4}$ , or 1, respectively. Staple this sheet to the top of your homework.

---

### Required Problems

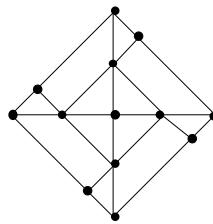
1. (a) Prove that  $P \subseteq \text{co-NP}$ .  
(b) Show that if  $NP \neq \text{co-NP}$ , then *no* NP-complete problem is a member of co-NP.
2. 2SAT is a special case of the formula satisfiability problem, where the input formula is in conjunctive normal form and every clause has at most two literals. Prove that 2SAT is in P.
3. Describe an algorithm that solves the following problem, called 3SUM, as quickly as possible: Given a set of  $n$  numbers, does it contain three elements whose sum is zero? For example, your algorithm should answer TRUE for the set  $\{-5, -17, 7, -4, 3, -2, 4\}$ , since  $-5+7+(-2)=0$ , and FALSE for the set  $\{-6, 7, -4, -13, -2, 5, 13\}$ .

4. (a) Show that the problem of deciding whether one undirected graph is a subgraph of another is NP-complete.  
(b) Show that the problem of deciding whether an unweighted undirected graph has a path of length greater than  $k$  is NP-complete.
  5. (a) Consider the following problem: Given a set of axis-aligned rectangles in the plane, decide whether any point in the plane is covered by  $k$  or more rectangles. Now also consider the CLIQUE problem. Describe and analyze a reduction of one problem to the other.  
(b) Finding the largest clique in an arbitrary graph is NP-hard. What does this fact imply about the complexity of finding a point that lies inside the largest number of rectangles?
6. [This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]

PARTITION is the problem of deciding, given a set  $S = \{s_1, s_2, \dots, s_n\}$  of numbers, whether there is a subset  $T$  containing half the 'weight' of  $S$ , i.e., such that  $\sum T = \frac{1}{2} \sum S$ . SUBSETSUM is the problem of deciding, given a set  $S = \{s_1, s_2, \dots, s_n\}$  of numbers and a target sum  $t$ , whether there is a subset  $T \subseteq S$  such that  $\sum T = t$ . Give two reductions between these two problems, one in each direction.

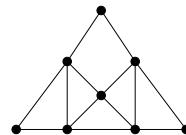
## Practice Problems

1. What is the *exact* worst case number of comparisons needed to find the median of 5 numbers?  
For 6 numbers?
2. The EXACTCOVERBYTHREES problem is defined as follows: given a finite set  $X$  and a collection  $C$  of 3-element subsets of  $X$ , does  $C$  contain an *exact cover* for  $X$ , that is, a subcollection  $C' \subseteq C$  where every element of  $X$  occurs in exactly one member of  $C'$ ? Given that EXACTCOVERBYTHREES is NP-complete, show that the similar problem EXACTCOVERBYFOURS is also NP-complete.
3. Using 3COLOR and the ‘gadget’ below, prove that the problem of deciding whether a planar graph can be 3-colored is NP-complete. [Hint: Show that the gadget can be 3-colored, and then replace any crossings in a planar embedding with the gadget appropriately.]



Crossing gadget for PLANAR3COLOR.

4. Using the previous result, and the ‘gadget’ below, prove that the problem of deciding whether a planar graph with no vertex of degree greater than four can be 3-colored is NP-complete. [Hint: Show that you can replace any vertex with degree greater than 4 with a collection of gadgets connected in such a way that no degree is greater than four.]



Degree gadget for DEGREE4PLANAR3COLOR

5. Show that an algorithm that makes at most a constant number of calls to polynomial-time subroutines runs in polynomial time, but that a polynomial number of calls to polynomial-time subroutines may result in an exponential-time algorithm.
6. (a) Prove that if  $G$  is an undirected bipartite graph with an odd number of vertices, then  $G$  is nonhamiltonian. Give a polynomial time algorithm for finding a hamiltonian cycle in an undirected bipartite graph or establishing that it does not exist.  
(b) Show that the hamiltonian-path problem can be solved in polynomial time on directed acyclic graphs.  
(c) Explain why the results in previous questions do not contradict the fact that both HAMILTONIANCYCLE and HAMILTONIANPATH are NP-complete problems.
7. Consider the following pairs of problems:

- (a) MIN SPANNING TREE and MAX SPANNING TREE
- (b) SHORTEST PATH and LONGEST PATH
- (c) TRAVELING SALESMAN and VACATION TOUR (the longest tour is sought).
- (d) MIN CUT and MAX CUT (between  $s$  and  $t$ )
- (e) EDGE COVER and VERTEX COVER
- (f) TRANSITIVE REDUCTION and MIN EQUIVALENT DIGRAPH

(All of these seem dual or opposites, except the last, which are just two versions of minimal representation of a graph.) Which of these pairs are polytime equivalent and which are not?

\*8. Consider the problem of deciding whether one graph is isomorphic to another.

- (a) Give a brute force algorithm to decide this.
- (b) Give a dynamic programming algorithm to decide this.
- (c) Give an efficient probabilistic algorithm to decide this.
- ★(d) Either prove that this problem is NP-complete, give a poly time algorithm for it, or prove that neither case occurs.

\*9. Prove that PRIMALITY (Given  $n$ , is  $n$  prime?) is in  $\text{NP} \cap \text{co-NP}$ . Showing that PRIMALITY is in co-NP is easy. (What's a certificate for showing that a number is composite?) For NP, consider a certificate involving primitive roots and recursively their primitive roots. Show that this tree of primitive roots can be checked to be correct and used to show that  $n$  is prime, and that this check takes polynomial time.

10. How much wood would a woodchuck chuck if a woodchuck could chuck wood?

# CS 373: Combinatorial Algorithms, Fall 2000

Midterm 1 — October 3, 2000

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

This is a closed-book, closed-notes exam!

If you brought anything with you besides writing instruments and your  $8\frac{1}{2}'' \times 11''$  cheat sheet, please leave it at the front of the classroom.

- 
- Print your name, netid, and alias in the boxes above. Circle U if you are an undergrad,  $\frac{3}{4}$  if you are a 3/4-unit grad student, or 1 if you are a 1-unit grad student. Print your name at the top of every page (in case the staple falls out!).
  - **Answer four of the five questions on the exam.** Each question is worth 10 points. If you answer more than four questions, the one with the lowest score will be ignored. **1-unit graduate students must answer question 5.**
  - Please write your final answers on the front of the exam pages. Use the backs of the pages as scratch paper. Let us know if you need more paper.
  - Unless we specifically say otherwise, proofs are not required. However, they may help us give you partial credit.
  - Read the entire exam before writing anything. Make sure you understand what the questions are asking. If you give a beautiful answer to the wrong question, you'll get no credit. If any question is unclear, please ask one of us for clarification.
  - Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.
  - Write something down for every problem. Don't panic and erase large chunks of work. Even if you think it's absolute nonsense, it might be worth partial credit.
  - Relax. Breathe. Kick some ass.

---

#	Score	Grader
1		
2		
3		
4		
5		
Total		

**1. Multiple Choice**

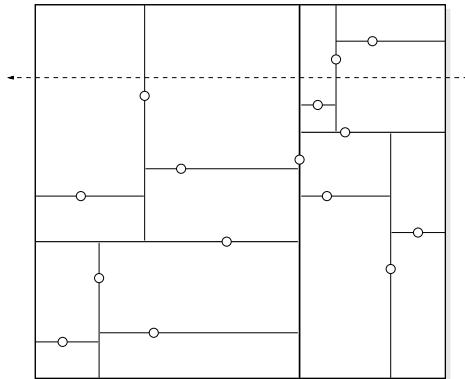
Every question below has one of the following answers.

- (a)  $\Theta(1)$       (b)  $\Theta(\log n)$       (c)  $\Theta(n)$       (d)  $\Theta(n \log n)$       (e)  $\Theta(n^2)$

For each question, write the letter that corresponds to your answer. You do not need to justify your answers. Each correct answer earns you 1 point, but each incorrect answer *costs* you  $\frac{1}{2}$  point. You cannot score below zero.

- What is  $\sum_{i=1}^n \log i$ ?
- What is  $\sum_{i=1}^n \frac{n}{i}$ ?
- How many digits do you need to write  $2^n$  in decimal?
- What is the solution of the recurrence  $T(n) = 25T(n/5) + n$ ?
- What is the solution of the recurrence  $T(n) = T(n - 1) + \frac{1}{2^n}$ ?
- What is the solution of the recurrence  $T(n) = 3T\left(\lceil \frac{n+51}{3} \rceil\right) + 17n - \sqrt[3]{\lg \lg n} - 2^{2^{\log^* n}} + \pi$ ?
- What is the worst-case running time of randomized quicksort?
- The expected time for inserting one item into an  $n$ -node randomized treap is  $O(\log n)$ .  
What is the worst-case time for a sequence of  $n$  insertions into an initially empty treap?
- The amortized time for inserting one item into an  $n$ -node scapegoat tree is  $O(\log n)$ .  
What is the worst-case time for a sequence of  $n$  insertions into an initially empty scapegoat tree?
- In the worst case, how many nodes can be in the root list of a Fibonacci heap storing  $n$  keys, immediately after a DECREASEKEY operation?
- Every morning, an Amtrak train leaves Chicago for Champaign, 200 miles away. The train can accelerate or decelerate at 10 miles per hour per second, and it has a maximum speed of 60 miles an hour. Every 50 miles, the train must stop for five minutes while a school bus crosses the tracks. Every hour, the conductor stops the train for a union-mandated 10-minute coffee break. How long does it take the train to reach Champaign?

2. Suppose we have  $n$  points scattered inside a two-dimensional box. A *kd-tree* recursively subdivides the rectangle as follows. First we split the box into two smaller boxes with a *vertical* line, then we split each of those boxes with *horizontal* lines, and so on, always alternating between horizontal and vertical splits. Each time we split a box, the splitting line passes through some point inside the box (*not* on the boundary) and partitions the rest of the interior points as evenly as possible. If a box doesn't contain any points, we don't split it any more; these final empty boxes are called *cells*.

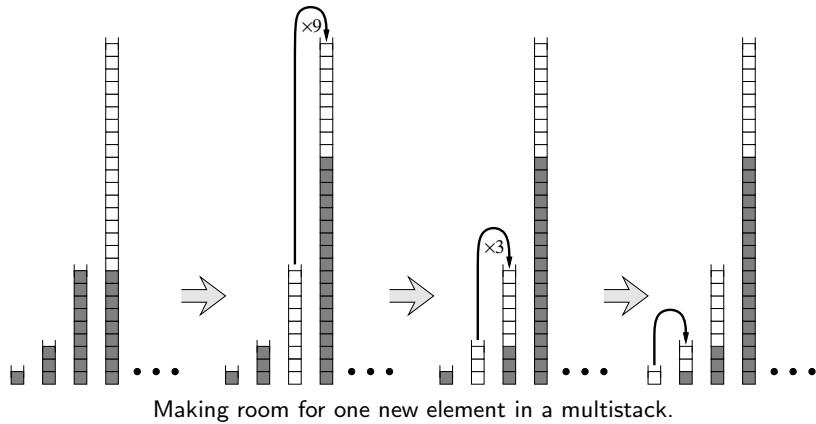


A kd-tree for 15 points. The dashed line crosses four cells.

- (a) [2 points] How many cells are there, as a function of  $n$ ? Prove your answer is correct.
- (b) [8 points] In the worst case, exactly how many cells can a horizontal line cross, as a function of  $n$ ? Prove your answer is correct. Assume that  $n = 2^k - 1$  for some integer  $k$ .  
[For full credit, you must give an exact answer. A tight asymptotic bound (with proof) is worth 5 points. A correct recurrence is worth 3 points.]
- (c) [5 points extra credit] In the worst case, how many cells can a diagonal line cross?

Incidentally, ‘kd-tree’ originally meant ‘ $k$ -dimensional tree’—for example, the specific data structure described here used to be called a ‘2d-tree’—but current usage ignores this etymology. The phrase ‘ $d$ -dimensional kd-tree’ is now considered perfectly standard, even though it’s just as redundant as ‘ATM machine’, ‘PIN number’, ‘HIV virus’, ‘PDF format’, ‘Mt. Fujiyama’, ‘Sahara Desert’, ‘The La Brea Tar Pits’, or ‘and etc.’ On the other hand, ‘BASIC code’ is *not* redundant; ‘Beginner’s All-Purpose Instruction Code’ is a backronym. Hey, aren’t you supposed to be taking a test?

3. A *multistack* consists of an infinite series of stacks  $S_0, S_1, S_2, \dots$ , where the  $i$ th stack  $S_i$  can hold up to  $3^i$  elements. Whenever a user attempts to push an element onto any full stack  $S_i$ , we first move all the elements in  $S_i$  to stack  $S_{i+1}$  to make room. But if  $S_{i+1}$  is already full, we first move all its members to  $S_{i+2}$ , and so on. Moving a single element from one stack to the next takes  $O(1)$  time.



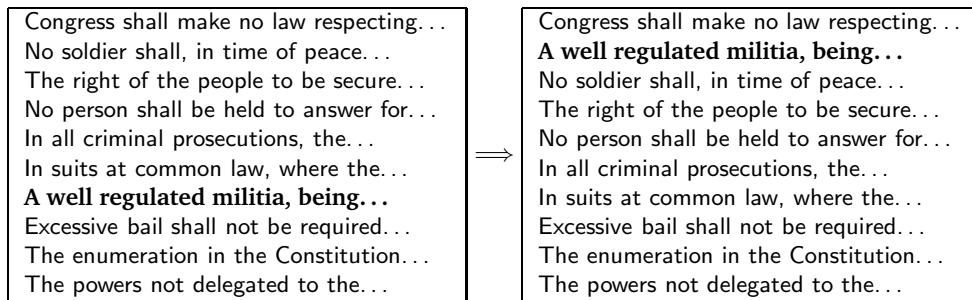
- (a) [1 point] In the worst case, how long does it take to push one more element onto a multistack containing  $n$  elements?
- (b) [9 points] Prove that the amortized cost of a push operation is  $O(\log n)$ , where  $n$  is the maximum number of elements in the multistack. You can use any method you like.

4. After graduating with a computer science degree, you find yourself working for a software company that publishes a word processor. The program stores a document containing  $n$  characters, grouped into  $p$  paragraphs. Your manager asks you to implement a ‘Sort Paragraphs’ command that rearranges the paragraphs into alphabetical order.

Design and analyze an efficient paragraph-sorting algorithm, using the following pair of routines as black boxes.

- COMPAREPARAGRAPHS( $i, j$ ) compares the  $i$ th and  $j$ th paragraphs, and returns  $i$  or  $j$  depending on which paragraph should come first in the final sorted output. (Don’t worry about ties.) This function runs in  $O(1)$  time, since almost any two paragraphs can be compared by looking at just their first few characters!
- MOVEPARAGRAPH( $i, j$ ) ‘cuts’ out the  $i$ th paragraph and ‘pastes’ it back in as the  $j$ th paragraph. This function runs in  $O(n_i)$  time, where  $n_i$  is the number of characters in the  $i$ th paragraph. (So in particular,  $n_1 + n_2 + \dots + n_p = n$ .)

Here is an example of MOVEPARAGRAPH(7, 2):



[Hint: For full credit, your algorithm should run in  $o(n \log n)$  time when  $p = o(n)$ .]

5. [1-unit grad students must answer this question.]

Describe and analyze an algorithm to randomly shuffle an array of  $n$  items, so that each of the  $n!$  possible permutations is equally likely. Assume that you have a function RANDOM( $i, j$ ) that returns a random integer from the set  $\{i, i + 1, \dots, j\}$  in constant time.

[Hint: As a sanity check, you might want to confirm that for  $n = 3$ , all six permutations have probability  $1/6$ . For full credit, your algorithm must run in  $\Theta(n)$  time. A correct algorithm that runs in  $\Theta(n \log n)$  time is worth 7 points.]

From: "Josh Pepper" <jwpepper@uiuc.edu>  
To: "Chris Neihengen" <neihenge@uiuc.edu>  
Subject: FW: proof  
Date: Fri, 29 Sep 2000 09:34:56 -0500

thought you might like this.

Problem: To prove that computer science 373 is indeed the work of Satan.

Proof: First, let us assume that everything in "Helping Yourself with Numerology", by Helyn Hitchcock, is true.

Second, let us apply divide and conquer to this problem. There are main parts:

1. The name of the course: "Combinatorial Algorithms"
2. The most important individual in the course, the "Recursion Fairy"
3. The number of this course: 373.

We examine these sequentially.

The name of the course. "Combinatorial Algorithms" can actually be expressed as a single integer - 23 - since it has 23 letters.

The most important individual, the Recursion Fairy, can also be expressed as a single integer - 14 - since it has 14 letters. In other words:

COMBINATORIAL ALGORITHMS = 23  
RECUSION FAIRY = 14

As a side note, a much shorter proof has already been published showing that the Recursion Fairy is Lucifer, and that any class involving the Fairy is from Lucifer, however, that proofs numerical significance is slight.

Now we can move on to an analysis of the number of course, which holds great meaning. The first assumption we make is that the number of the course, 373, is not actually a base 10 number. We can prove this inductively by making a reasonable guess for the actual base, then finding a new way to express the nature of the course, and if the answer

confirms what we assumed, then we're right. That's the way induction works.

What is a reasonable guess for the base of the course? The answer is trivial, since the basest of all beings is the Recursion Fairy, the base is 14. So a true base 10 representation of 373 (base 14) is 689. So we see:

373 (base 14) = 689 (base 10)

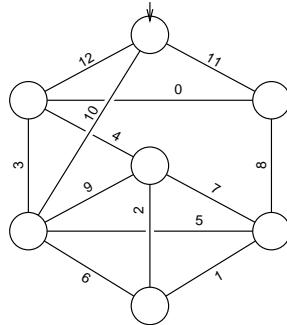
Now since the nature of the course has absolutely nothing to do with combinatorial algorithms (instead having much to do with the work of the devil), we can subtract from the above result everything having to do with combinatorial algorithms just by subtracting 23. Here we see that:

689 - 23 = 666

QED.

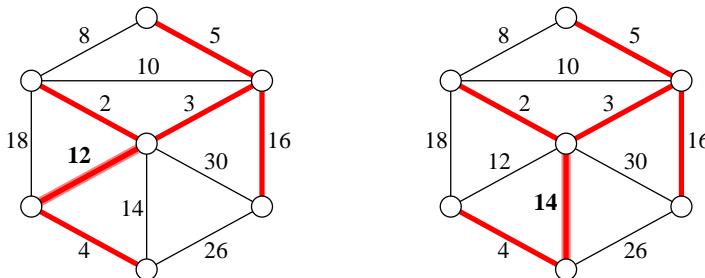
1. Using any method you like, compute the following subgraphs for the weighted graph below. Each subproblem is worth 3 points. Each incorrect edge costs you 1 point, but you cannot get a negative score for any subproblem.

- (a) a depth-first search tree, starting at the top vertex;
- (b) a breadth-first search tree, starting at the top vertex;
- (c) a shortest path tree, starting at the top vertex;
- (d) the minimum spanning tree.



2. Suppose you are given a weighted undirected graph  $G$  (represented as an adjacency list) and its minimum spanning tree  $T$  (which you already know how to compute). Describe and analyze an algorithm to find the *second-minimum spanning tree* of  $G$ , i.e., the spanning tree of  $G$  with smallest total weight except for  $T$ .

The minimum spanning tree and the second-minimum spanning tree differ by exactly one edge. But *which* edge is different, and *how* is it different? That's what your algorithm has to figure out!

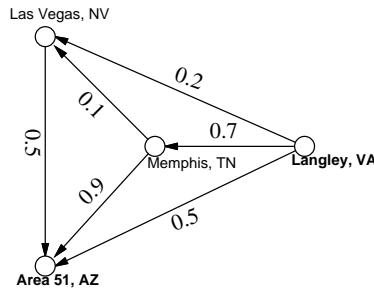


The minimum spanning tree and the second-minimum spanning tree of a graph.

3. (a) [4 pts] Prove that a connected acyclic graph with  $V$  vertices has exactly  $V - 1$  edges. (“It’s a tree!” is not a proof.)
- (b) [4 pts] Describe and analyze an algorithm that determines whether a given graph is a tree, where the graph is represented by an adjacency list.
- (c) [2 pts] What is the running time of your algorithm from part (b) if the graph is represented by an adjacency matrix?

4. Mulder and Scully have computed, for every road in the United States, the exact probability that someone driving on that road *won't* be abducted by aliens. Agent Mulder needs to drive from Langley, Virginia to Area 51, Nevada. What route should he take so that he has the least chance of being abducted?

More formally, you are given a directed graph  $G = (V, E)$ , where every edge  $e$  has an independent safety probability  $p(e)$ . The *safety* of a path is the product of the safety probabilities of its edges. Design and analyze an algorithm to determine the safest path from a given start vertex  $s$  to a given target vertex  $t$ .



With the probabilities shown above, if Mulder tries to drive directly from Langley to Area 51, he has a 50% chance of getting there without being abducted. If he stops in Memphis, he has a  $0.7 \times 0.9 = 63\%$  chance of arriving safely. If he stops first in Memphis and then in Las Vegas, he has a  $1 - 0.7 \times 0.1 \times 0.5 = 96.5\%$  chance of being abducted!<sup>1</sup>

5. [1-unit grad students must answer this question.]

Many string matching applications allow the following *wild card* characters in the pattern.

- The wild card ? represents an arbitrary single character. For example, the pattern s?r?ng matches the strings string, sprung, and sarong.
- The wild card \* represents an arbitrary string of zero or more characters. For example, the pattern te\*st\* matches the strings test, tensest, and technostructuralism.

Both wild cards can occur in a single pattern. For example, the pattern f\*a?? matches the strings face, football, and flippyfloppydingdongdang. On the other hand, neither wild card can occur in the text.

Describe how to modify the Knuth-Morris-Pratt algorithm to support patterns with these wild cards, and analyze the modified algorithm. Your algorithm should find the first substring in the text that matches the pattern. An algorithm that supports only one of the two wild cards is worth 5 points.

---

<sup>1</sup>That's how they got Elvis, you know.

**1. True, False, or Maybe**

Indicate whether each of the following statements is always true, sometimes true, always false, or unknown. Some of these questions are deliberately tricky, so read them carefully. Each correct choice is worth +1, and each incorrect choice is worth -1. **Guessing will hurt you!**

- (a) Suppose SMARTALGORITHM runs in  $\Theta(n^2)$  time and DUMBALGORITHM runs in  $\Theta(2^n)$  time for all inputs of size  $n$ . (Thus, for each algorithm, the best-case and worst-case running times are the same.) SMARTALGORITHM is faster than DUMBALGORITHM.

True    False    Sometimes    Nobody Knows

- (b) QUICKSORT runs in  $O(n^6)$  time.

True    False    Sometimes    Nobody Knows

- (c)  $\lfloor \log_2 n \rfloor \geq \lceil \log_2 n \rceil$

True    False    Sometimes    Nobody Knows

- (d) The recurrence  $F(n) = n + 2\sqrt{n} \cdot F(\sqrt{n})$  has the solution  $F(n) = \Theta(n \log n)$ .

True    False    Sometimes    Nobody Knows

- (e) A Fibonacci heap with  $n$  nodes has depth  $\Omega(\log n)$ .

True    False    Sometimes    Nobody Knows

- (f) Suppose a graph  $G$  is represented by an adjacency matrix. It is possible to determine whether  $G$  is an independent set without looking at every entry of the adjacency matrix.

True    False    Sometimes    Nobody Knows

- (g)  $NP \neq co\text{-}NP$

True    False    Sometimes    Nobody Knows

- (h) Finding the smallest clique in a graph is NP-hard.

True    False    Sometimes    Nobody Knows

- (i) A polynomial-time reduction from X to 3SAT proves that X is NP-hard.

True    False    Sometimes    Nobody Knows

- (j) The correct answer for exactly three of these questions is “False”.

True    False

**2. Convex Hull**

Suppose you are given the convex hull of a set of  $n$  points, and one additional point  $(x, y)$ . The convex hull is represented by an array of vertices in counterclockwise order, starting from the leftmost vertex. Describe how to test in  $O(\log n)$  time whether or not the additional point  $(x, y)$  is inside the convex hull.

**3. Finding the Largest Block**

In your new job, you are working with screen images. These are represented using two dimensional arrays where each element is a 1 or a 0, indicating whether that position of the screen is illuminated. Design and analyze an efficient algorithm to find the largest rectangular block of ones in such an array. For example, the largest rectangular block of ones in the array shown below is in rows 2–4 and columns 2–3. [Hint: Use dynamic programming.]

1	0	1	0	0
1	1	1	0	1
0	1	1	1	1
1	1	1	0	0

**4. The Hogwarts Sorting Hat**

Every year, upon their arrival at Hogwarts School of Witchcraft and Wizardry, new students are sorted into one of four houses (Gryffindor, Hufflepuff, Ravenclaw, or Slytherin) by the Hogwarts Sorting Hat. The student puts the Hat on their head, and the Hat tells the student which house they will join. This year, a failed experiment by Fred and George Weasley filled almost all of Hogwarts with sticky brown goo, mere moments before the annual Sorting. As a result, the Sorting had to take place in the basement hallways, where there was so little room to move that the students had to stand in a long line.

After everyone learned what house they were in, the students tried to group together by house, but there was too little room in the hallway for more than one student to move at a time. Fortunately, the Sorting Hat took CS 373 many years ago, so it knew how to group the students as quickly as possible. What method did the Sorting Hat use?

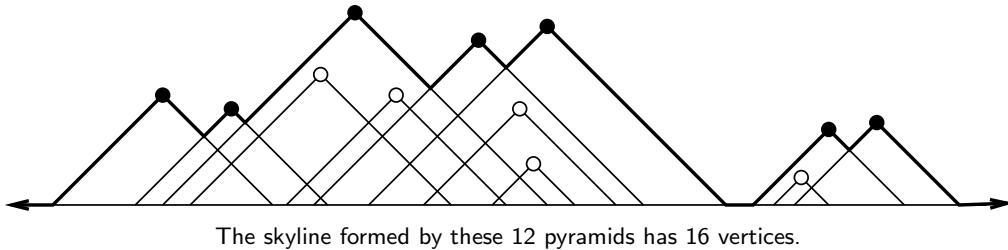
More formally, you are given an array of  $n$  items, where each item has one of four possible values, possibly with a pointer to some additional data. Design and analyze an algorithm that rearranges the items into four clusters in  $O(n)$  time using only  $O(1)$  extra space.

The diagram illustrates the sorting process for 20 items. The initial array (top) contains 20 cells, each with a letter (G, H, R, S) and a name. The final array (bottom) shows the same 20 cells, but the letters are rearranged to group them by value. An arrow points from the top array to the bottom array, indicating the transformation.

G Harry	H Ann	R Bob	R Tina	G Chad	G Bill	R Lisa	G Ekta	H Bart	H Jim	H John	R Jeff	S Liz	R Mary	H Dawn	G Nick	S Kim	H Fox	G Dana	G Mel
G Harry	G Ekta	G Bill	G Chad	G Nick	G Mel	G Dana	H Fox	H Ann	H Jim	H Dawn	H Bart	R Lisa	R Tina	R John	R Bob	R Liz	R Mary	S Kim	S Jeff

### 5. The Egyptian Skyline

Suppose you are given a set of  $n$  pyramids in the plane. Each pyramid is a isosceles triangle with two  $45^\circ$  edges and a horizontal edge on the  $x$ -axis. Each pyramid is represented by the  $x$ - and  $y$ -coordinates of its topmost point. Your task is to compute the “skyline” formed by these pyramids (the dark line shown below).



- (a) Describe and analyze an algorithm that determines which pyramids are visible on the skyline. These are the pyramids with black points in the figure above; the pyramids with white points are not visible. [Hint: You've seen this problem before.]
- (b) Once you know which pyramids are visible, how would you compute the *shape* of the skyline? Describe and analyze an algorithm to compute the left-to-right sequence of skyline vertices, including the vertices between the pyramids and on the ground.

### 6. DNF-SAT

A boolean formula is in *disjunctive normal form* (DNF) if it consists of clauses of conjunctions (ANDs) joined together by disjunctions (ORs). For example, the formula

$$(\bar{a} \wedge b \wedge \bar{c}) \vee (b \wedge c) \vee (a \wedge \bar{b} \wedge \bar{c})$$

is in disjunctive normal form. DNF-SAT is the problem that asks, given a boolean formula in disjunctive normal form, whether that formula is satisfiable.

- (a) Show that DNF-SAT is in P.
- (b) What is wrong with the following argument that P=NP?

Suppose we are given a boolean formula in conjunctive normal form with at most three literals per clause, and we want to know if it is satisfiable. We can use the distributive law to construct an equivalent formula in disjunctive normal form. For example,

$$(a \vee b \vee \bar{c}) \wedge (\bar{a} \vee \bar{b}) \iff (a \wedge \bar{b}) \vee (b \wedge \bar{a}) \vee (\bar{c} \wedge \bar{a}) \vee (\bar{c} \wedge \bar{b})$$

Now we can use the answer to part (a) to determine, in polynomial time, whether the resulting DNF formula is satisfiable. We have just solved 3SAT in polynomial time! Since 3SAT is NP-hard, we must conclude that P=NP.

### 7. Magic 3-Coloring [1-unit graduate students must answer this question.]

The recursion fairy's distant cousin, the reduction genie, shows up one day with a magical gift for you—a box that determines in constant time whether or not a graph is 3-colorable. (A graph is 3-colorable if you can color each of the vertices red, green, or blue, so that every edge has different colors.) The magic box does not tell you *how* to color the graph, just whether or not it can be done. Devise and analyze an algorithm to 3-color any graph **in polynomial time** using this magic box.

# CS 373: Combinatorial Algorithms, Spring 2001

## Homework 0, due January 23, 2001 at the beginning of class

Name:	
Net ID:	Alias:

Neatly print your name (first name first, with no comma), your network ID, and a short alias into the boxes above. **Do not sign your name. Do not write your Social Security number.** Staple this sheet of paper to the top of your homework.

Grades will be listed on the course web site by alias give us, so your alias should not resemble your name or your Net ID. If you don't give yourself an alias, we'll give you one that you won't like.

---

This homework tests your familiarity with the prerequisite material from CS 173, CS 225, and CS 273—many of these problems have appeared on homeworks or exams in those classes—primarily to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Parberry and Chapters 1–6 of CLR should be sufficient review, but you may want to consult other texts as well.

---

Before you do anything else, read the Homework Instructions and FAQ on the CS 373 course web page (<http://www-courses.cs.uiuc.edu/~cs373/hw/faq.html>), and then check the box below. This web page gives instructions on how to write and submit homeworks—staple your solutions together in order, write your name and netID on every page, don't turn in source code, analyze everything, use good English and good logic, and so forth.

I have read the CS 373 Homework Instructions and FAQ.

---

### Required Problems

1. (a) Prove that any positive integer can be written as the sum of distinct powers of 2. For example:  $42 = 2^5 + 2^3 + 2^1$ ,  $25 = 2^4 + 2^3 + 2^0$ ,  $17 = 2^4 + 2^0$ . [Hint: ‘Write the number in binary’ is not a proof; it just restates the problem.]
- (b) Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers—if  $F_n$  appears in the sum, then neither  $F_{n+1}$  nor  $F_{n-1}$  will. For example:  $42 = F_9 + F_6$ ,  $25 = F_8 + F_4 + F_2$ ,  $17 = F_7 + F_4 + F_2$ .
- (c) Prove that *any* integer (positive, negative, or zero) can be written in the form  $\sum_i \pm 3^i$ , where the exponents  $i$  are distinct non-negative integers. For example:  $42 = 3^4 - 3^3 - 3^2 - 3^1$ ,  $25 = 3^3 - 3^1 + 3^0$ ,  $17 = 3^3 - 3^2 - 3^0$ .

2. Sort the following 20 functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway just for practice.

$$\begin{array}{ccccc}
 1 & n & n^2 & \lg n & \lg^* n \\
 2^{2^{\lg \lg n + 1}} & \lg^* 2^n & 2^{\lg^* n} & \lfloor \lg(n!) \rfloor & \lfloor \lg n \rfloor! \\
 n^{\lg n} & (\lg n)^n & (\lg n)^{\lg n} & n^{1/\lg n} & n^{\lg \lg n} \\
 \log_{1000} n & \lg^{1000} n & \lg^{(1000)} n & \left(1 + \frac{1}{1000}\right)^n & n^{1/1000}
 \end{array}$$

To simplify notation, write  $f(n) \ll g(n)$  to mean  $f(n) = o(g(n))$  and  $f(n) \equiv g(n)$  to mean  $f(n) = \Theta(g(n))$ . For example, the functions  $n^2$ ,  $n$ ,  $\binom{n}{2}$ ,  $n^3$  could be sorted either as  $n \ll n^2 \equiv \binom{n}{2} \ll n^3$  or as  $n \ll \binom{n}{2} \equiv n^2 \ll n^3$ .

3. Solve the following recurrences. State tight asymptotic bounds for each function in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ . You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway just for practice. Assume reasonable but nontrivial base cases if none are supplied. Extra credit will be given for more exact solutions.

- (a)  $A(n) = 5A(n/3) + n \log n$
- (b)  $B(n) = \min_{0 < k < n} (B(k) + B(n - k) + 1)$ .
- (c)  $C(n) = 4C(\lfloor n/2 \rfloor + 5) + n^2$
- (d)  $D(n) = D(n - 1) + 1/n$
- \*(e)  $E(n) = n + 2\sqrt{n} \cdot E(\sqrt{n})$

4. This problem asks you to simplify some recursively defined boolean formulas as much as possible. In each case, prove that your answer is correct. Each proof can be just a few sentences long, but it must be a *proof*.

- (a) Suppose  $\alpha_0 = p$ ,  $\alpha_1 = q$ , and  $\alpha_n = (\alpha_{n-2} \wedge \alpha_{n-1})$  for all  $n \geq 2$ . Simplify  $\alpha_n$  as much as possible. [Hint: What is  $\alpha_5$ ?]
- (b) Suppose  $\beta_0 = p$ ,  $\beta_1 = q$ , and  $\beta_n = (\beta_{n-2} \Leftrightarrow \beta_{n-1})$  for all  $n \geq 2$ . Simplify  $\beta_n$  as much as possible. [Hint: What is  $\beta_5$ ?]
- (c) Suppose  $\gamma_0 = p$ ,  $\gamma_1 = q$ , and  $\gamma_n = (\gamma_{n-2} \Rightarrow \gamma_{n-1})$  for all  $n \geq 2$ . Simplify  $\gamma_n$  as much as possible. [Hint: What is  $\gamma_5$ ?]
- (d) Suppose  $\delta_0 = p$ ,  $\delta_1 = q$ , and  $\delta_n = (\delta_{n-2} \bowtie \delta_{n-1})$  for all  $n \geq 2$ , where  $\bowtie$  is some boolean function with two arguments. Find a boolean function  $\bowtie$  such that  $\delta_n = \delta_m$  if and only if  $n - m$  is a multiple of 4. [Hint: There is only one such function.]

5. Every year, upon their arrival at Hogwarts School of Witchcraft and Wizardry, new students are sorted into one of four houses (Gryffindor, Hufflepuff, Ravenclaw, or Slytherin) by the Hogwarts Sorting Hat. The student puts the Hat on their head, and the Hat tells the student which house they will join. This year, a failed experiment by Fred and George Weasley filled almost all of Hogwarts with sticky brown goo, mere moments before the annual Sorting. As a result, the Sorting had to take place in the basement hallways, where there was so little room to move that the students had to stand in a long line.

After everyone learned what house they were in, the students tried to group together by house, but there was too little room in the hallway for more than one student to move at a time. Fortunately, the Sorting Hat took CS 373 many years ago, so it knew how to group the students as quickly as possible. What method did the Sorting Hat use?

More formally, you are given an array of  $n$  items, where each item has one of four possible values, possibly with a pointer to some additional data. Describe an algorithm<sup>1</sup> that rearranges the items into four clusters in  $O(n)$  time using only  $O(1)$  extra space.

$G$ Harry	$H$ Ann	$R$ Bob	$R$ Tina	$G$ Chad	$G$ Bill	$R$ Lisa	$G$ Ekta	$H$ Bart	$H$ Jim	$R$ John	$S$ Jeff	$R$ Liz	$R$ Mary	$H$ Dawn	$G$ Nick	$S$ Kim	$H$ Fox	$G$ Dana	$G$ Mel
↓																			
$G$ Harry	$G$ Ekta	$G$ Bill	$G$ Chad	$G$ Nick	$G$ Mel	$G$ Dana	$H$ Fox	$H$ Ann	$H$ Jim	$H$ Dawn	$H$ Bart	$R$ Lisa	$R$ Tina	$R$ John	$R$ Bob	$R$ Liz	$R$ Mary	$S$ Kim	$S$ Jeff

6. [This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]

Penn and Teller have a special deck of fifty-two cards, with no face cards and nothing but clubs—the ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, . . . , 52 of clubs. (They’re big cards.) Penn shuffles the deck until each each of the 52! possible orderings of the cards is equally likely. He then takes cards one at a time from the top of the deck and gives them to Teller, stopping as soon as he gives Teller the three of clubs.

- (a) On average, how many cards does Penn give Teller?
- (b) On average, what is the smallest-numbered card that Penn gives Teller?
- \*(c) On average, what is the largest-numbered card that Penn gives Teller?

[Hint: Solve for an  $n$ -card deck and then set  $n = 52$ .] In each case, give *exact* answers and prove that they are correct. If you have to appeal to “intuition” or “common sense”, your answers are probably wrong!

---

<sup>1</sup>Since you’ve read the Homework Instructions, you know what the phrase ‘describe an algorithm’ means. Right?

## Practice Problems

The remaining problems are entirely for your benefit; similar questions will appear in every homework. Don't turn in solutions—we'll just throw them out—but feel free to ask us about practice questions during office hours and review sessions. Think of them as potential exam questions (hint, hint). We'll post solutions to *some* of the practice problems after the homeworks are due.

1. Recall the standard recursive definition of the Fibonacci numbers:  $F_0 = 0$ ,  $F_1 = 1$ , and  $F_n = F_{n-1} + F_{n-2}$  for all  $n \geq 2$ . Prove the following identities for all positive integers  $n$  and  $m$ .

(a)  $F_n$  is even if and only if  $n$  is divisible by 3.

(b)  $\sum_{i=0}^n F_i = F_{n+2} - 1$

(c)  $F_n^2 - F_{n+1}F_{n-1} = (-1)^{n+1}$

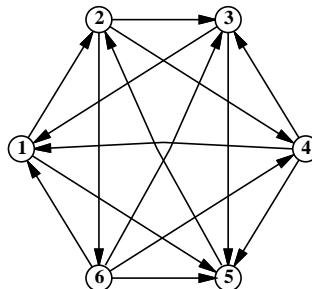
- ★(d) If  $n$  is an integer multiple of  $m$ , then  $F_n$  is an integer multiple of  $F_m$ .

2. (a) Prove the following identity by induction:

$$\binom{2n}{n} = \sum_{k=0}^n \binom{n}{k} \binom{n}{n-k}.$$

- (b) Give a non-inductive combinatorial proof of the same identity, by showing that the two sides of the equation count exactly the same thing in two different ways. There is a correct one-sentence proof.

3. A *tournament* is a directed graph with exactly one edge between every pair of vertices. (Think of the nodes as players in a round-robin tournament, where each edge points from the winner to the loser.) A *Hamiltonian path* is a sequence of directed edges, joined end to end, that visits every vertex exactly once. Prove that every tournament contains at least one Hamiltonian path.



A six-vertex tournament containing the Hamiltonian path  $6 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1$ .

4. Solve the following recurrences. State tight asymptotic bounds for each function in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ . You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway just for practice. Assume reasonable but nontrivial base cases if none are supplied. Extra credit will be given for more exact solutions.

- (a)  $A(n) = A(n/2) + n$
- (b)  $B(n) = 2B(n/2) + n$
- ★(c)  $C(n) = n + \frac{1}{2}(C(n-1) + C(3n/4))$
- (d)  $D(n) = \max_{n/3 < k < 2n/3} (D(k) + D(n-k) + n)$
- \*(e)  $E(n) = 2E(n/2) + n/\lg n$
- \*(f)  $F(n) = \frac{F(n-1)}{F(n-2)}$ , where  $F(1) = 1$  and  $F(2) = 2$ .
- \*(g)  $G(n) = G(n/2) + G(n/4) + G(n/6) + G(n/12) + n$  [Hint:  $\frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \frac{1}{12} = 1$ .]
- \*(h)  $H(n) = n + \sqrt{n} \cdot H(\sqrt{n})$
- \*(i)  $I(n) = (n-1)(I(n-1) + I(n-2))$ , where  $I(0) = I(1) = 1$
- \*(j)  $J(n) = 8J(n-1) - 15J(n-2) + 1$

5. (a) Prove that  $2^{\lceil \lg n \rceil + \lfloor \lg n \rfloor} = \Theta(n^2)$ .

- (b) Prove or disprove:  $2^{\lfloor \lg n \rfloor} = \Theta(2^{\lceil \lg n \rceil})$ .
- (c) Prove or disprove:  $2^{2^{\lfloor \lg \lg n \rfloor}} = \Theta(2^{2^{\lceil \lg \lg n \rceil}})$ .
- (d) Prove or disprove: If  $f(n) = O(g(n))$ , then  $\log(f(n)) = O(\log(g(n)))$ .
- (e) Prove or disprove: If  $f(n) = O(g(n))$ , then  $2^{f(n)} = O(2^{g(n)})$ .
- \*(f) Prove that  $\log^k n = o(n^{1/k})$  for any positive integer  $k$ .

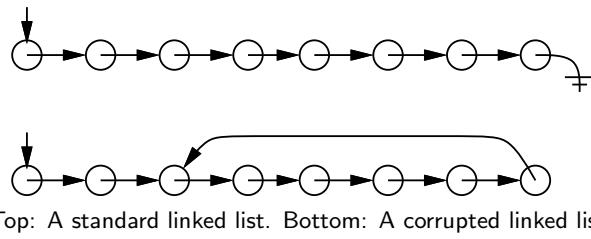
6. Evaluate the following summations; simplify your answers as much as possible. Significant partial credit will be given for answers in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ .

$$(a) \sum_{i=1}^n \sum_{j=1}^i \sum_{k=j}^i \frac{1}{i}$$

$$*(b) \sum_{i=1}^n \sum_{j=1}^i \sum_{k=j}^i \frac{1}{j}$$

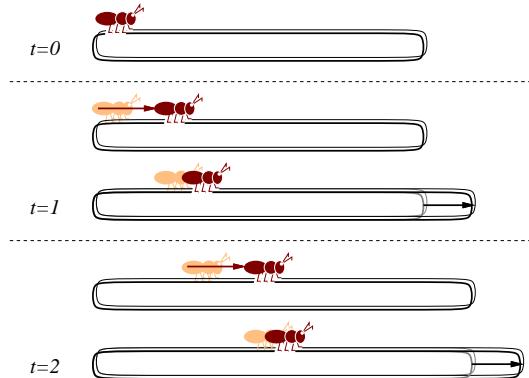
$$(c) \sum_{i=1}^n \sum_{j=1}^i \sum_{k=j}^i \frac{1}{k}$$

7. Suppose you have a pointer to the head of singly linked list. Normally, each node in the list only has a pointer to the next element, and the last node's pointer is `NULL`. Unfortunately, your list might have been corrupted by a bug in somebody else's code<sup>2</sup>, so that the last node has a pointer back to some other node in the list instead.



Describe an algorithm that determines whether the linked list is corrupted or not. Your algorithm must not modify the list. For full credit, your algorithm should run in  $O(n)$  time, where  $n$  is the number of nodes in the list, and use  $O(1)$  extra space (not counting the list itself).

- \*8. An ant is walking along a rubber band, starting at the left end. Once every second, the ant walks one inch to the right, and then you make the rubber band one inch longer by pulling on the right end. The rubber band stretches uniformly, so stretching the rubber band also pulls the ant to the right. The initial length of the rubber band is  $n$  inches, so after  $t$  seconds, the rubber band is  $n + t$  inches long.

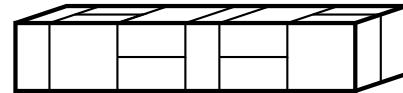


Every second, the ant walks an inch, and then the rubber band is stretched an inch longer.

- (a) How far has the ant moved after  $t$  seconds, as a function of  $n$  and  $t$ ? Set up a recurrence and (for full credit) give an *exact* closed-form solution. [Hint: What *fraction* of the rubber band's length has the ant walked?]
- (b) How long does it take the ant to get to the right end of the rubber band? For full credit, give an answer of the form  $f(n) + \Theta(1)$  for some explicit function  $f(n)$ .
9. (a) A *domino* is a  $2 \times 1$  or  $1 \times 2$  rectangle. How many different ways are there to completely fill a  $2 \times n$  rectangle with  $n$  dominos? Set up a recurrence relation and give an *exact* closed-form solution.

<sup>2</sup>After all, *your* code is always completely 100% bug-free. Isn't that right, Mr. Gates?

- (b) A *slab* is a three-dimensional box with dimensions  $1 \times 2 \times 2$ ,  $2 \times 1 \times 2$ , or  $2 \times 2 \times 1$ . How many different ways are there to fill a  $2 \times 2 \times n$  box with  $n$  slabs? Set up a recurrence relation and give an *exact* closed-form solution.



A  $2 \times 10$  rectangle filled with ten dominos, and a  $2 \times 2 \times 10$  box filled with ten slabs.

10. Professor George O'Jungle has a favorite 26-node binary tree, whose nodes are labeled by letters of the alphabet. The preorder and postorder sequences of nodes are as follows:

preorder: M N H C R S K W T G D X I Y A J P O E Z V B U L Q F  
 postorder: C W T K S G R H D N A O E P J Y Z I B Q L F U V X M

Draw Professor O'Jungle's binary tree, and give the inorder sequence of nodes.

11. Alice and Bob each have a fair  $n$ -sided die. Alice rolls her die once. Bob then repeatedly throws his die until he rolls a number at least as big as the number Alice rolled. Each time Bob rolls, he pays Alice \$1. (For example, if Alice rolls a 5, and Bob rolls a 4, then a 3, then a 1, then a 5, the game ends and Alice gets \$4. If Alice rolls a 1, then no matter what Bob rolls, the game will end immediately, and Alice will get \$1.)

*Exactly* how much money does Alice expect to win at this game? Prove that your answer is correct. If you have to appeal to ‘intuition’ or ‘common sense’, your answer is probably wrong!

12. Prove that for any nonnegative parameters  $a$  and  $b$ , the following algorithms terminate and produce identical output.

**SLOWEUCLID( $a, b$ ) :**

```

if  $b > a$ 
    return SLOWEUCLID( $b, a$ )
else if  $b = 0$ 
    return  $a$ 
else
    return SLOWEUCLID( $b, a - b$ )
  
```

**FASTEUCLID( $a, b$ ) :**

```

if  $b = 0$ 
    return  $a$ 
else
    return FASTEUCLID( $b, a \bmod b$ )
  
```

# CS 373: Combinatorial Algorithms, Spring 2001

## Homework 1 (due Thursday, February 1, 2001 at 11:59:59 p.m.)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

---

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U,  $\frac{3}{4}$ , or 1, respectively. Staple this sheet to the top of your homework.

---

### Required Problems

1. Suppose you are a simple shopkeeper living in a country with  $n$  different types of coins, with values  $1 = c[1] < c[2] < \dots < c[n]$ . (In the U.S., for example,  $n = 6$  and the values are 1, 5, 10, 25, 50 and 100 cents.) Your beloved and benevolent dictator, El Generalissimo, has decreed that whenever you give a customer change, you must use the smallest possible number of coins, so as not to wear out the image of El Generalissimo lovingly engraved on each coin by servants of the Royal Treasury.
  - (a) In the United States, there is a simple greedy algorithm that always results in the smallest number of coins: subtract the largest coin and recursively give change for the remainder. El Generalissimo does not approve of American capitalist greed. Show that there is a set of coin values for which the greedy algorithm does *not* always give the smallest possible of coins.
  - (b) Describe and analyze a dynamic programming algorithm to determine, given a target amount  $A$  and a sorted array  $c[1..n]$  of coin values, the smallest number of coins needed to make  $A$  cents in change. You can assume that  $c[1] = 1$ , so that it is possible to make change for any amount  $A$ .

2. Consider the following sorting algorithm:

```
STUPIDSORT( $A[0..n-1]$ ) :
  if  $n = 2$  and  $A[0] > A[1]$ 
    swap  $A[0] \leftrightarrow A[1]$ 
  else if  $n > 2$ 
     $m \leftarrow \lceil 2n/3 \rceil$ 
    STUPIDSORT( $A[0..m-1]$ )
    STUPIDSORT( $A[n-m..n-1]$ )
    STUPIDSORT( $A[0..m-1]$ )
```

- (a) Prove that STUPIDSORT actually sorts its input.
- (b) Would the algorithm still sort correctly if we replaced the line  $m \leftarrow \lceil 2n/3 \rceil$  with  $m \leftarrow \lfloor 2n/3 \rfloor$ ? Justify your answer.
- (c) State a recurrence (including the base case(s)) for the number of comparisons executed by STUPIDSORT.
- (d) Solve the recurrence, and prove that your solution is correct. [Hint: Ignore the ceiling.] Does the algorithm deserve its name?
- \*(e) Show that the number of *swaps* executed by STUPIDSORT is at most  $\binom{n}{2}$ .

3. The following randomized algorithm selects the  $r$ th smallest element in an unsorted array  $A[1..n]$ . For example, to find the smallest element, you would call RANDOMSELECT( $A, 1$ ); to find the median element, you would call RANDOMSELECT( $A, \lfloor n/2 \rfloor$ ). Recall from lecture that PARTITION splits the array into three parts by comparing the pivot element  $A[p]$  to every other element of the array, using  $n - 1$  comparisons altogether, and returns the new index of the pivot element.

```
RANDOMSELECT( $A[1..n], r$ ) :
   $p \leftarrow \text{RANDOM}(1, n)$ 
   $k \leftarrow \text{PARTITION}(A[1..n], p)$ 
  if  $r < k$ 
    return RANDOMSELECT( $A[1..k-1], r$ )
  else if  $r > k$ 
    return RANDOMSELECT( $A[k+1..n], r-k$ )
  else
    return  $A[k]$ 
```

- (a) State a recurrence for the expected running time of RANDOMSELECT, as a function of  $n$  and  $r$ .
- (b) What is the *exact* probability that RANDOMSELECT compares the  $i$ th smallest and  $j$ th smallest elements in the input array? The correct answer is a simple function of  $i$ ,  $j$ , and  $r$ . [Hint: Check your answer by trying a few small examples.]
- (c) Show that for any  $n$  and  $r$ , the expected running time of RANDOMSELECT is  $\Theta(n)$ . You can use either the recurrence from part (a) or the probabilities from part (b). For extra credit, find the exact expected number of comparisons, as a function of  $n$  and  $r$ .
- (d) What is the expected number of times that RANDOMSELECT calls itself recursively?

4. What excitement! The Champaign Spinners and the Urbana Dreamweavers have advanced to meet each other in the World Series of Basketweaving! The World Champions will be decided by a best-of-  $2n - 1$  series of head-to-head weaving matches, and the first to win  $n$  matches will take home the coveted Golden Basket (for example, a best-of-7 series requiring four match wins, but we will keep the generalized case). We know that for any given match there is a constant probability  $p$  that Champaign will win, and a subsequent probability  $q = 1 - p$  that Urbana will win.

Let  $P(i, j)$  be the probability that Champaign will win the series given that they still need  $i$  more victories, whereas Urbana needs  $j$  more victories for the championship.  $P(0, j) = 1$ ,  $1 \leq j \leq n$ , because Champaign needs no more victories to win.  $P(i, 0) = 0$ ,  $1 \leq i \leq n$ , as Champaign cannot possibly win if Urbana already has.  $P(0, 0)$  is meaningless. Champaign wins any particular match with probability  $p$  and loses with probability  $q$ , so

$$P(i, j) = p \cdot P(i - 1, j) + q \cdot P(i, j - 1)$$

for any  $i \geq 1$  and  $j \geq 1$ .

Create and analyze an  $O(n^2)$ -time dynamic programming algorithm that takes the parameters  $n$ ,  $p$  and  $q$  and returns the probability that Champaign will win the series (that is, calculate  $P(n, n)$ ).

5. The traditional Devonian/Cornish drinking song “The Barley Mow” has the following pseudolyrics<sup>1</sup>, where  $\text{container}[i]$  is the name of a container that holds  $2^i$  ounces of beer.<sup>2</sup>

```
BARLEYMOW( $n$ ):
    "Here's a health to the barley-mow, my brave boys,"
    "Here's a health to the barley-mow!"

    "We'll drink it out of the jolly brown bowl,"
    "Here's a health to the barley-mow!"
    "Here's a health to the barley-mow, my brave boys,"
    "Here's a health to the barley-mow!"

for  $i \leftarrow 1$  to  $n$ 
    "We'll drink it out of the container[ $i$ ], boys,"
    "Here's a health to the barley-mow!"
    for  $j \leftarrow i$  downto 1
        "The container[j],"
        "And the jolly brown bowl!"
        "Here's a health to the barley-mow!"
        "Here's a health to the barley-mow, my brave boys,"
        "Here's a health to the barley-mow!"
```

- (a) Suppose each container name  $\text{container}[i]$  is a single word, and you can sing four words a second. How long would it take you to sing BARLEYMOW( $n$ )? (Give a tight asymptotic bound.)
- (b) If you want to sing this song for  $n > 20$ , you’ll have to make up your own container names, and to avoid repetition, these names will get progressively longer as  $n$  increases<sup>3</sup>. Suppose  $\text{container}[n]$  has  $\Theta(\log n)$  syllables, and you can sing six syllables per second. Now how long would it take you to sing BARLEYMOW( $n$ )? (Give a tight asymptotic bound.)
- (c) Suppose each time you mention the name of a container, you drink the corresponding amount of beer: one ounce for the jolly brown bowl, and  $2^i$  ounces for each  $\text{container}[i]$ . Assuming for purposes of this problem that you are at least 21 years old, *exactly* how many ounces of beer would you drink if you sang BARLEYMOW( $n$ )? (Give an *exact* answer, not just an asymptotic bound.)

---

<sup>1</sup>Pseudolyrics are to lyrics as pseudocode is to code.

<sup>2</sup>One version of the song uses the following containers: nipperkin, gill pot, half-pint, pint, quart, pottle, gallon, half-anker, anker, firkin, half-barrel, barrel, hogshead, pipe, well, river, and ocean. Every container in this list is twice as big as its predecessor, except that a firkin is actually 2.25 ankers, and the last three units are just silly.

<sup>3</sup>“We'll drink it out of the hemisemidemiyottapint, boys!”

6. [This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]

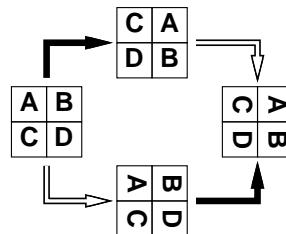
Suppose we want to display a paragraph of text on a computer screen. The text consists of  $n$  words, where the  $i$ th word is  $p_i$  pixels wide. We want to break the paragraph into several lines, each exactly  $P$  pixels long. Depending on which words we put on each line, we will need to insert different amounts of white space between the words. The paragraph should be fully justified, meaning that the first word on each line starts at its leftmost pixel, and *except for the last line*, the last character on each line ends at its rightmost pixel. There must be at least one pixel of whitespace between any two words on the same line.

Define the *slop* of a paragraph layout as the sum over all lines, *except the last*, of the cube of the number of extra white-space pixels in each line (not counting the one pixel required between every adjacent pair of words). Specifically, if a line contains words  $i$  through  $j$ , then the amount of extra white space on that line is  $P - j + i - \sum_{k=i}^j p_k$ . Describe a dynamic programming algorithm to print the paragraph with minimum slop.

## Practice Problems

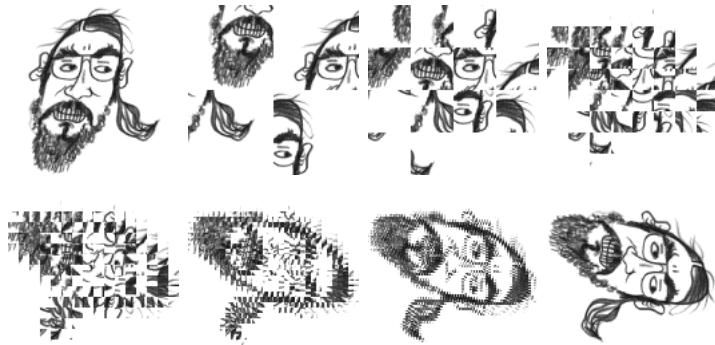
1. Give an  $O(n^2)$  algorithm to find the longest increasing subsequence of a sequence of numbers. The elements of the subsequence need not be adjacent in the sequence. For example, the sequence  $\langle 1, 5, 3, 2, 4 \rangle$  has longest increasing subsequence  $\langle 1, 3, 4 \rangle$ .
  
2. You are at a political convention with  $n$  delegates. Each delegate is a member of exactly one political party. It is impossible to tell which political party a delegate belongs to. However, you can check whether any two delegates are in the *same* party or not by introducing them to each other. (Members of the same party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.)
  - (a) Suppose a majority (more than half) of the delegates are from the same political party. Give an efficient algorithm that identifies a member of the majority party.
  - (b) Suppose exactly  $k$  political parties are represented at the convention and one party has a *plurality*: more delegates belong to that party than to any other. Present a practical procedure to pick a person from the plurality party as parsimoniously as possible. (Please.)
  
3. Give an algorithm that finds the *second* smallest of  $n$  elements in at most  $n + \lceil \lg n \rceil - 2$  comparisons. [Hint: divide and conquer to find the smallest; where is the second smallest?]
  
4. Some graphics hardware includes support for an operation called *blit*, or **block transfer**, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.
 

Suppose we want to rotate an  $n \times n$  pixelmap 90° clockwise. One way to do this is to split the pixelmap into four  $n/2 \times n/2$  blocks, move each block to its proper position using a sequence of five blits, and then recursively rotate each block. Alternately, we can first recursively rotate the blocks and blit them into place afterwards.



Two algorithms for rotating a pixelmap.  
 Black arrows indicate blitting the blocks into place.  
 White arrows indicate recursively rotating the blocks.

The following sequence of pictures shows the first algorithm (blit then recurse) in action.



In the following questions, assume  $n$  is a power of two.

- (a) Prove that both versions of the algorithm are correct. [Hint: If you exploit all the available symmetries, your proof will only be a half of a page long.]
  - (b) *Exactly* how many blits does the algorithm perform?
  - (c) What is the algorithm's running time if a  $k \times k$  blit takes  $O(k^2)$  time?
  - (d) What if a  $k \times k$  blit takes only  $O(k)$  time?
- 
5. A company is planning a party for its employees. The employees in the company are organized into a strict hierarchy, that is, a tree with the company president at the root. The organizers of the party have assigned a real number to each employee measuring how ‘fun’ the employee is. In order to keep things social, there is one restriction on the guest list: an employee cannot attend the party if their immediate supervisor is present. On the other hand, the president of the company *must* attend the party, even though she has a negative fun rating; it’s her company, after all. Give an algorithm that makes a guest list for the party that maximizes the sum of the ‘fun’ ratings of the guests.
  
  6. Suppose you have a subroutine that can find the median of a set of  $n$  items (*i.e.*, the  $\lfloor n/2 \rfloor$  smallest) in  $O(n)$  time. Give an algorithm to find the  $k$ th biggest element (for arbitrary  $k$ ) in  $O(n)$  time.
  
  7. You’re walking along the beach and you stub your toe on something in the sand. You dig around it and find that it is a treasure chest full of gold bricks of different (integral) weight. Your knapsack can only carry up to weight  $n$  before it breaks apart. You want to put as much in it as possible without going over, but you *cannot* break the gold bricks up.
    - (a) Suppose that the gold bricks have the weights  $1, 2, 4, 8, \dots, 2^k$ ,  $k \geq 1$ . Describe and prove correct a greedy algorithm that fills the knapsack as much as possible without going over.
    - (b) Give a set of 3 weight values for which the greedy algorithm does *not* yield an optimal solution and show why.
    - (c) Give a dynamic programming algorithm that yields an optimal solution for an arbitrary set of gold brick values.

# CS 373: Combinatorial Algorithms, Spring 2001

Homework 2 (due Thu. Feb. 15, 2001 at 11:59 PM)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

---

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U,  $\frac{3}{4}$ , or 1, respectively. Staple this sheet to the top of your homework.

---

## Required Problems

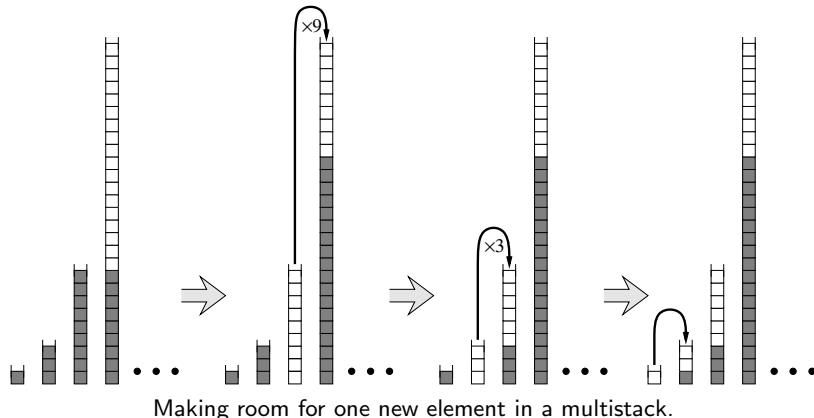
1. Suppose we are given two sorted arrays  $A[1..n]$  and  $B[1..n]$  and an integer  $k$ . Describe an algorithm to find the  $k$ th smallest element in the union of  $A$  and  $B$ . (For example, if  $k = 1$ , your algorithm should return the smallest element of  $A \cup B$ ; if  $k = n$ , our algorithm should return the median of  $A \cup B$ .) You can assume that the arrays contain no duplicates. For full credit, your algorithm should run in  $\Theta(\log n)$  time. [Hint: First try to solve the special case  $k = n$ .]
2. Say that a binary search tree is *augmented* if every node  $v$  also stores  $|v|$ , the size of its subtree.
  - (a) Show that a rotation in an augmented binary tree can be performed in constant time.
  - (b) Describe an algorithm SCAPEGOATSELECT( $k$ ) that selects the  $k$ th smallest item in an augmented scapegoat tree in  $O(\log n)$  worst-case time.
  - (c) Describe an algorithm SPLAISELECT( $k$ ) that selects the  $k$ th smallest item in an augmented splay tree in  $O(\log n)$  amortized time.

- (d) Describe an algorithm TREAPSELECT( $k$ ) that selects the  $k$ th smallest item in an augmented treap in  $O(\log n)$  *expected* time.
3. (a) Prove that only one subtree gets rebalanced in a scapegoat tree insertion.
- (b) Prove that  $I(v) = 0$  in every node of a perfectly balanced tree. (Recall that  $I(v) = \max\{0, |T| - |s| - 1\}$ , where  $T$  is the child of greater height and  $s$  the child of lesser height, and  $|v|$  is the number of nodes in subtree  $v$ . A perfectly balanced tree has two perfectly balanced subtrees, each with as close to half the nodes as possible.)
- \*(c) Show that you can rebuild a fully balanced binary tree from an unbalanced tree in  $O(n)$  time using only  $O(\log n)$  additional memory.
4. Suppose we can insert or delete an element into a hash table in constant time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:

- After an insertion, if the table is more than  $3/4$  full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
- After a deletion, if the table is less than  $1/4$  full, we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do *not* use the potential method (it makes it much more difficult).

5. A *multistack* consists of an infinite series of stacks  $S_0, S_1, S_2, \dots$ , where the  $i$ th stack  $S_i$  can hold up to  $3^i$  elements. Whenever a user attempts to push an element onto any full stack  $S_i$ , we first move all the elements in  $S_i$  to stack  $S_{i+1}$  to make room. But if  $S_{i+1}$  is already full, we first move all its members to  $S_{i+2}$ , and so on. Moving a single element from one stack to the next takes  $O(1)$  time.



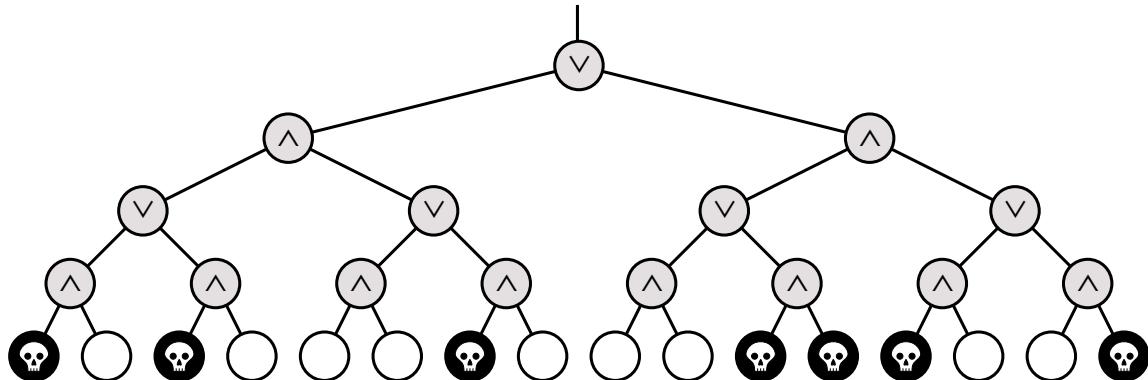
- (a) [1 point] In the worst case, how long does it take to push one more element onto a multistack containing  $n$  elements?
- (b) [9 points] Prove that the amortized cost of a push operation is  $O(\log n)$ , where  $n$  is the maximum number of elements in the multistack. You can use any method you like.

6. [This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]

Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with  $4^n$  leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after  $2n$  moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.

You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are AND gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

- (a) (2 pts) Describe and analyze a deterministic algorithm to determine whether or not you can win. [Hint: This is easy!]
- (b) (8 pts) Unfortunately, Death won't let you even look at every node in the tree. Describe a randomized algorithm that determines whether you can win in  $\Theta(3^n)$  expected time. [Hint: Consider the case  $n = 1$ .]



## Practice Problems

1. (a) Show that it is possible to transform any  $n$ -node binary search tree into any other  $n$ -node binary search tree using at most  $2n - 2$  rotations.

\*(b) Use fewer than  $2n - 2$  rotations. Nobody knows how few rotations are required in the worst case. There is an algorithm that can transform any tree to any other in at most  $2n - 6$  rotations, and there are pairs of trees that are  $2n - 10$  rotations apart. These are the best bounds known.

2. Faster Longest Increasing Subsequence(LIS)

Give an  $O(n \log n)$  algorithm to find the longest increasing subsequence of a sequence of numbers. [Hint: In the dynamic programming solution, you don't really have to look back at all previous items. There was a practice problem on HW 1 that asked for an  $O(n^2)$  algorithm for this. If you are having difficulty, look at the solution provided in the HW 1 solutions.]

3. Amortization

- (a) Modify the binary double-counter (see class notes Sept 12) to support a new operation SIGN, which determines whether the number being stored is positive, negative, or zero, in constant time. The amortized time to increment or decrement the counter should still be a constant.

[Hint: Suppose  $p$  is the number of significant bits in  $P$ , and  $n$  is the number of significant bits in  $N$ . For example, if  $P = 17 = 10001_2$  and  $N = 0$ , then  $p = 5$  and  $n = 0$ . Then  $p - n$  always has the same sign as  $P - N$ . Assume you can update  $p$  and  $n$  in  $O(1)$  time.]

- \*(b) Do the same but now you can't assume that  $p$  and  $n$  can be updated in  $O(1)$  time.

- \*4. Amortization

Suppose instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of 'fits', where the  $i$ th least significant fit indicates whether the sum includes the  $i$ th Fibonacci number  $F_i$ . For example, the fit string 101110 represents the number  $F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14$ . Describe algorithms to increment and decrement a fit string in constant amortized time. [Hint: Most numbers can be represented by more than one fit string. This is *not* the same representation as on Homework 0.]

5. Detecting overlap

- (a) You are given a list of ranges represented by min and max (*e.g.*, [1,3], [4,5], [4,9], [6,8], [7,10]). Give an  $O(n \log n)$ -time algorithm that decides whether or not a set of ranges contains a pair that overlaps. You need not report all intersections. If a range completely covers another, they are overlapping, even if the boundaries do not intersect.

- (b) You are given a list of rectangles represented by min and max  $x$ - and  $y$ -coordinates. Give an  $O(n \log n)$ -time algorithm that decides whether or not a set of rectangles contains a pair that overlaps (with the same qualifications as above). [Hint: sweep a vertical line from left to right, performing some processing whenever an end-point is encountered. Use a balanced search tree to maintain any extra info you might need.]

## 6. Comparison of Amortized Analysis Methods

A sequence of  $n$  operations is performed on a data structure. The  $i$ th operation costs  $i$  if  $i$  is an exact power of 2, and 1 otherwise. That is operation  $i$  costs  $f(i)$ , where:

$$f(i) = \begin{cases} i, & i = 2^k, \\ 1, & \text{otherwise} \end{cases}$$

Determine the amortized cost per operation using the following methods of analysis:

- (a) Aggregate method
- (b) Accounting method
- \*(c) Potential method

# CS 373: Combinatorial Algorithms, Spring 2001

## Homework 3 (due Thursday, March 8, 2001 at 11:59.99 p.m.)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

---

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U,  $\frac{3}{4}$ , or 1, respectively. Staple this sheet to the top of your homework.

---

### Required Problems

#### 1. Hashing:

A hash table of size  $m$  is used to store  $n$  items with  $n \leq m/2$ . Open addressing is used for collision resolution.

- Assuming uniform hashing, show that for  $i = 1, 2, \dots, n$ , the probability that the  $i^{th}$  insertion requires strictly more than  $k$  probes is at most  $2^{-k}$ .
- Show that for  $i = 1, 2, \dots, n$ , the probability that the  $i^{th}$  insertion requires more than  $2 \lg n$  probes is at most  $1/n^2$ .

Let the random variable  $X_i$  denote the number of probes required by the  $i^{th}$  insertion. You have shown in part (b) that  $\Pr\{X_i > 2 \lg n\} \leq 1/n^2$ . Let the random variable  $X = \max_{1 \leq i \leq n} X_i$  denote the maximum number of probes required by any of the  $n$  insertions.

- Show that  $\Pr\{X > 2 \lg n\} \leq 1/n$ .
- Show that the expected length of the longest probe sequence is  $E[X] = O(\lg n)$ .

## 2. Reliable Network:

Suppose you are given a graph of a computer network  $G = (V, E)$  and a function  $r(u, v)$  that gives a reliability value for every edge  $(u, v) \in E$  such that  $0 \leq r(u, v) \leq 1$ . The reliability value gives the probability that the network connection corresponding to that edge will *not* fail. Describe and analyze an algorithm to find the most reliable path from a given source vertex  $s$  to a given target vertex  $t$ .

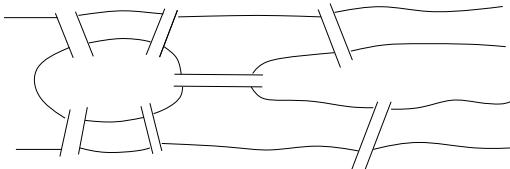
## 3. Aerophobia:

After graduating you find a job with Aerophobes-R'-Us, the leading traveling agency for aerophobic people. Your job is to build a system to help customers plan airplane trips from one city to another. All of your customers are afraid of flying so the trip should be as short as possible.

In other words, a person wants to fly from city  $A$  to city  $B$  in the shortest possible time. S/he turns to the traveling agent who knows all the departure and arrival times of all the flights on the planet. Give an algorithm that will allow the agent to choose an optimal route to minimize the total time in transit. Hint: rather than modify Dijkstra's algorithm, modify the data. The total transit time is from departure to arrival at the destination, so it will include layover time (time waiting for a connecting flight).

## 4. The Seven Bridges of Königsberg:

During the eighteenth century the city of Königsberg in East Prussia was divided into four sections by the Pregel river. Seven bridges connected these regions, as shown below. It was said that residents spent their Sunday walks trying to find a way to walk about the city so as to cross each bridge exactly once and then return to their starting point.



- (a) Show how the residents of the city could accomplish such a walk or prove no such walk exists.
  - (b) Given any undirected graph  $G = (V, E)$ , give an algorithm that finds a cycle in the graph that visits every edge exactly once, or says that it can't be done.
5. Minimum Spanning Tree changes:

Suppose you have a graph  $G$  and an MST of that graph (i.e. the MST has already been constructed).

- (a) Give an algorithm to update the MST when an edge is added to  $G$ .
- (b) Give an algorithm to update the MST when an edge is deleted from  $G$ .
- (c) Give an algorithm to update the MST when a vertex (and possibly edges to it) is added to  $G$ .

## 6. Nesting Envelopes

[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.] You are given an unlimited number of each of  $n$  different types of envelopes. The dimensions of envelope type  $i$  are  $x_i \times y_i$ . In nesting envelopes inside one another, you can place envelope  $A$  inside envelope  $B$  if and only if the dimensions  $A$  are *strictly smaller* than the dimensions of  $B$ . Design and analyze an algorithm to determine the largest number of envelopes that can be nested inside one another.

**Practice Problems**

## 1. Makefiles:

In order to facilitate recompiling programs from multiple source files when only a small number of files have been updated, there is a UNIX utility called ‘make’ that only recompiles those files that were changed after the most recent compilation, *and* any intermediate files in the compilation that depend on those that were changed. A Makefile is typically composed of a list of source files that must be compiled. Each of these source files is dependent on some of the other files which are listed. Thus a source file must be recompiled if a file on which it depends is changed.

Assuming you have a list of which files have been recently changed, as well as a list for each source file of the files on which it depends, design an algorithm to recompile only those necessary. DO NOT worry about the details of parsing a Makefile.

★2. Let the hash function for a table of size  $m$  be

$$h(x) = \lfloor Ax \rfloor \bmod m$$

where  $A = \hat{\phi} = \frac{\sqrt{5}-1}{2}$ . Show that this gives the best possible spread, i.e. if the  $x$  are hashed in order,  $x + 1$  will be hashed in the largest remaining contiguous interval.

3. The incidence matrix of an undirected graph  $G = (V, E)$  is a  $|V| \times |E|$  matrix  $B = (b_{ij})$  such that

$$b_{ij} = \begin{cases} 1 & (i, j) \in E, \\ 0 & (i, j) \notin E. \end{cases}$$

(a) Describe what all the entries of the matrix product  $BB^T$  represent ( $B^T$  is the matrix transpose). Justify.

(b) Describe what all the entries of the matrix product  $B^TB$  represent. Justify.

★(c) Let  $C = BB^T - 2A$ . Let  $C'$  be  $C$  with the first row and column removed. Show that  $\det C'$  is the number of spanning trees. ( $A$  is the adjacency matrix of  $G$ , with zeroes on the diagonal).

4.  $o(V^2)$  Adjacency Matrix Algorithms

(a) Give an  $O(V)$  algorithm to decide whether a directed graph contains a *sink* in an adjacency matrix representation. A sink is a vertex with in-degree  $V - 1$ .

- (b) An undirected graph is a scorpion if it has a vertex of degree 1 (the sting) connected to a vertex of degree two (the tail) connected to a vertex of degree  $V - 2$  (the body) connected to the other  $V - 3$  vertices (the feet). Some of the feet may be connected to other feet.

Design an algorithm that decides whether a given adjacency matrix represents a scorpion by examining only  $O(V)$  of the entries.

- (c) Show that it is impossible to decide whether  $G$  has at least one edge in  $O(V)$  time.

5. Shortest Cycle:

Given an **undirected** graph  $G = (V, E)$ , and a weight function  $f : E \rightarrow \mathbf{R}$  on the **edges**, give an algorithm that finds (in time polynomial in  $V$  and  $E$ ) a cycle of smallest weight in  $G$ .

6. Longest Simple Path:

Let graph  $G = (V, E)$ ,  $|V| = n$ . A *simple path* of  $G$ , is a path that does not contain the same vertex twice. Use dynamic programming to design an algorithm (not polynomial time) to find a simple path of maximum length in  $G$ . Hint: It can be done in  $O(n^c 2^n)$  time, for some constant  $c$ .

7. Minimum Spanning Tree:

Suppose all edge weights in a graph  $G$  are equal. Give an algorithm to compute an MST.

8. Transitive reduction:

Give an algorithm to construct a *transitive reduction* of a directed graph  $G$ , i.e. a graph  $G^{TR}$  with the fewest edges (but with the same vertices) such that there is a path from  $a$  to  $b$  in  $G$  iff there is also such a path in  $G^{TR}$ .

9. (a) What is  $5^{2^{29}5^0 + 23^41 + 17^32 + 11^23 + 5^14} \pmod{6}$ ?

- (b) What is the capital of Nebraska? Hint: It is not Omaha. It is named after a famous president of the United States that was not George Washington. The distance from the Earth to the Moon averages roughly 384,000 km.

# CS 373: Combinatorial Algorithms, Spring 2001

<http://www-courses.cs.uiuc.edu/~cs373>

Homework 4 (due Thu. March 29, 2001 at 11:59:59 pm)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

---

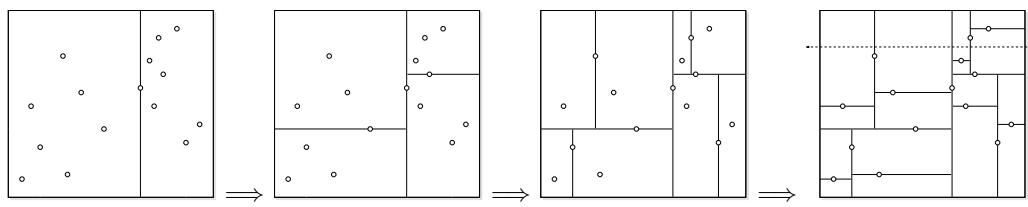
Homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U,  $\frac{3}{4}$ , or 1, respectively. Staple this sheet to the top of your homework.

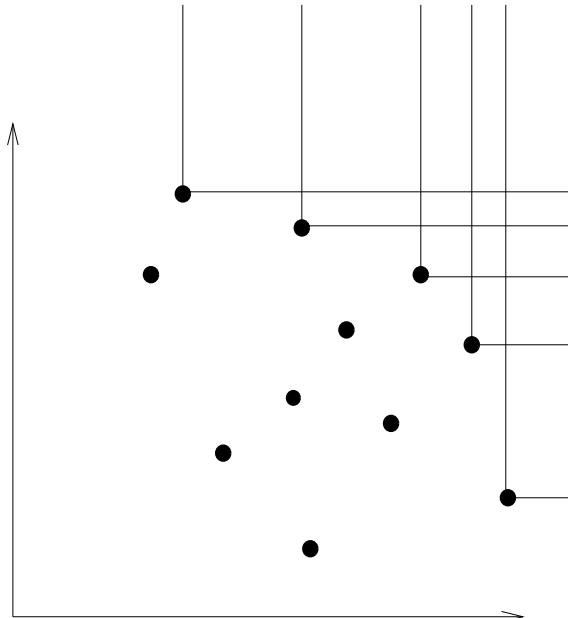
---

## Required Problems

- Suppose we have  $n$  points scattered inside a two-dimensional box. A *kd-tree* recursively subdivides the rectangle as follows. First we split the box into two smaller boxes with a *vertical* line, then we split each of those boxes with *horizontal* lines, and so on, always alternating between horizontal and vertical splits. Each time we split a box, the splitting line partitions the rest of the interior points *as evenly as possible* by passing through a median point inside the box (*not* on the boundary). If a box doesn't contain any points, we don't split it any more; these final empty boxes are called *cells*.



Successive divisions of a kd-tree for 15 points. The dashed line crosses four cells.



An example staircase as in problem 3.

- (a) How many cells are there, as a function of  $n$ ? Prove your answer is correct.
- (b) In the worst case, *exactly* how many cells can a horizontal line cross, as a function of  $n$ ? Prove your answer is correct. Assume that  $n = 2^k - 1$  for some integer  $k$ .
- (c) Suppose we have  $n$  points stored in a kd-tree. Describe an algorithm that counts the number of points above a horizontal line (such as the dashed line in the figure) in  $O(\sqrt{n})$  time.
- \*(d) [Optional: 5 pts extra credit] Find an algorithm that counts the number of points that lie inside a rectangle  $R$  and show that it takes  $O(\sqrt{n})$  time. You may assume that the sides of the rectangle are parallel to the sides of the box.

2. Circle Intersection [*This problem is worth 20 points*]

Describe an algorithm to decide, given  $n$  circles in the plane, whether any two of them intersect, in  $O(n \log n)$  time. Each circle is specified by three numbers: its radius and the  $x$ - and  $y$ -coordinates of its center.

We only care about intersections between circle boundaries; concentric circles do not intersect. What general position assumptions does your algorithm require? [*Hint: Modify an algorithm for detecting line segment intersections, but describe your modifications very carefully! There are at least two very different solutions.*]

3. Staircases

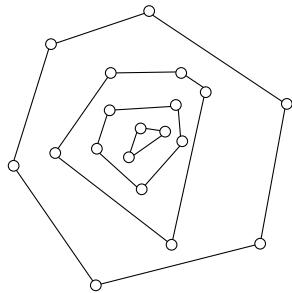
You are given a set of points in the first quadrant. A *left-up* point of this set is defined to be a point that has no points both greater than it in both coordinates. The left-up subset of a set of points then forms a *staircase* (see figure).

- (a) Prove that left-up points do not necessarily lie on the convex hull.
- (b) Give an  $O(n \log n)$  algorithm to find the staircase of a set of points.

- (c) Assume that points are chosen uniformly at random within a rectangle. What is the average number of points in a staircase? Justify. Hint: you will be able to give an exact answer rather than just asymptotics. You have seen the same analysis before.
4. Convex Layers

Given a set  $Q$  of points in the plane, define the *convex layers* of  $Q$  inductively as follows: The first convex layer of  $Q$  is just the convex hull of  $Q$ . For all  $i > 1$ , the  $i$ th convex layer is the convex hull of  $Q$  after the vertices of the first  $i - 1$  layers have been removed.

Give an  $O(n^2)$ -time algorithm to find all convex layers of a given set of  $n$  points.



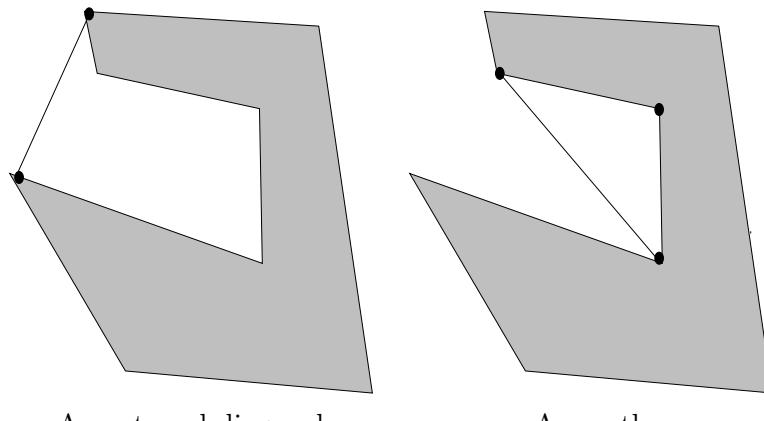
A set of points with four convex layers.

5. [This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.] Solve the travelling salesman problem for points in convex position (ie, the vertices of a convex polygon). Finding the shortest cycle that visits every point is easy – it's just the convex hull. Finding the shortest path that visits every point is a little harder, because the path can cross through the interior.

- (a) Show that the optimal path cannot be one that crosses itself.  
 (b) Describe an  $O(n^2)$  time dynamic programming algorithm to solve the problem.

## Practice Problems

1. Basic Computation (assume two dimensions and *exact* arithmetic)
  - (a) Intersection: Extend the basic algorithm to determine if two line segments intersect by taking care of *all* degenerate cases.
  - (b) Simplicity: Give an  $O(n \log n)$  algorithm to determine whether an  $n$ -vertex polygon is simple.
  - (c) Area: Give an algorithm to compute the area of a simple  $n$ -polygon (not necessarily convex) in  $O(n)$  time.
  - (d) Inside: Give an algorithm to determine whether a point is within a simple  $n$ -polygon (not necessarily convex) in  $O(n)$  time.
  
2. External Diagonals and Mouths
  - (a) A pair of polygon vertices defines an *external diagonal* if the line segment between them is completely outside the polygon. Show that every nonconvex polygon has at least one external diagonal.
  - (b) Three consecutive polygon vertices  $p, q, r$  form a *mouth* if  $p$  and  $r$  define an external diagonal. Show that every nonconvex polygon has at least one mouth.



3. On-Line Convex Hull

We are given the set of points one point at a time. After receiving each point, we must compute the convex hull of all those points so far. Give an algorithm to solve this problem in  $O(n^2)$  (We could obviously use Graham's scan  $n$  times for an  $O(n^2 \log n)$  algorithm). Hint: How do you maintain the convex hull?

4. Another On-Line Convex Hull Algorithm

- (a) Given an  $n$ -polygon and a point outside the polygon, give an algorithm to find a tangent.
- \*(b) Suppose you have found both tangents. Give an algorithm to remove the points from the polygon that are within the angle formed by the tangents (as segments!) and the opposite side of the polygon.

- (c) Use the above to give an algorithm to compute the convex hull on-line in  $O(n \log n)$
5. Order of the size of the convex hull  
The convex hull on  $n \geq 3$  points can have anywhere from 3 to  $n$  points. The average case depends on the distribution.
- (a) Prove that if a set of points is chosen randomly within a given rectangle then the average size of the convex hull is  $O(\log n)$ .
  - ★(b) Prove that if a set of points is chosen randomly within a given circle then the average size of the convex hull is  $O(n^{1/3})$ .
6. Ghostbusters and Ghosts  
A group of  $n$  ghostbusters is battling  $n$  ghosts. Each ghostbuster can shoot a single energy beam at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits a ghost. The ghostbusters must all fire at the same time and no two energy beams may cross (it would be bad). The positions of the ghosts and ghostbusters is fixed in the plane (assume that no three points are collinear).
- (a) Prove that for any configuration of ghosts and ghostbusters there exists such a non-crossing matching.
  - (b) Show that there exists a line passing through one ghostbuster and one ghost such that the number of ghostbusters on one side of the line equals the number of ghosts on the same side. Give an efficient algorithm to find such a line.
  - (c) Give an efficient divide and conquer algorithm to pair ghostbusters and ghosts so that no two streams cross.

# CS 373: Combinatorial Algorithms, Spring 2001

<http://www-courses.cs.uiuc.edu/~cs373>

**Homework 5 (due Tue. Apr. 17, 2001 at 11:59 pm)**

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

---

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate,  $\frac{3}{4}$ -unit grad student, or 1-unit grad student by circling U,  $\frac{3}{4}$ , or 1, respectively. Staple this sheet to the top of your homework.

---

## Required Problems

1. Prove that finding the second smallest of  $n$  elements takes EXACTLY  $n + \lceil \lg n \rceil - 2$  comparisons in the worst case. Prove for both upper and lower bounds. Hint: find the (first) smallest using an elimination tournament.
2. *Fibonacci strings* are defined as follows:

$$F_1 = "b", \quad F_2 = "a", \quad \text{and } F_n = F_{n-1}F_{n-2}, (n > 2)$$

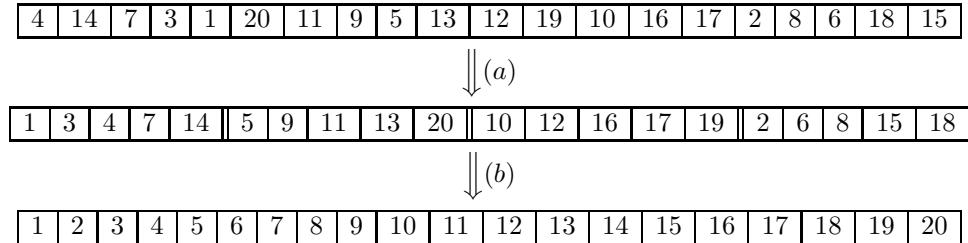
where the recursive rule uses concatenation of strings, so  $F_3$  is “ab”,  $F_4$  is “aba”. Note that the length of  $F_n$  is the  $n$ th Fibonacci number.

- (a) Prove that in any Fibonacci string there are no two b’s adjacent and no three a’s.
- (b) Give the unoptimized and optimized ‘prefix’ (fail) function for  $F_7$ .
- (c) Prove that, in searching for the Fibonacci string  $F_k$ , the unoptimized KMP algorithm can shift  $\lceil k/2 \rceil$  times in a row trying to match the last character of the pattern. In other words, prove that there is a chain of failure links  $m \rightarrow \text{fail}[m] \rightarrow \text{fail}[\text{fail}[m]] \rightarrow \dots$  of length  $\lceil k/2 \rceil$ , and find an example text  $T$  that would cause KMP to traverse this entire chain at a single text position.

- (d) Prove that the unoptimized KMP algorithm can shift  $k - 2$  times in a row at the same text position when searching for  $F_k$ . Again, you need to find an example text  $T$  that would cause KMP to traverse this entire chain on the same text character.
- (e) How do the failure chains in parts (c) and (d) change if we use the optimized failure function instead?

3. Two-stage sorting

- (a) Suppose we are given an array  $A[1..n]$  of distinct integers. Describe an algorithm that splits  $A$  into  $n/k$  subarrays, each with  $k$  elements, such that the elements of each subarray  $A[(i-1)k+1..ik]$  are sorted. Your algorithm should run in  $O(n \log k)$  time.
- (b) Given an array  $A[1..n]$  that is already split into  $n/k$  sorted subarrays as in part (a), describe an algorithm that sorts the entire array in  $O(n \log(n/k))$  time.
- (c) Prove that your algorithm from part (a) is optimal.
- (d) Prove that your algorithm from part (b) is optimal.

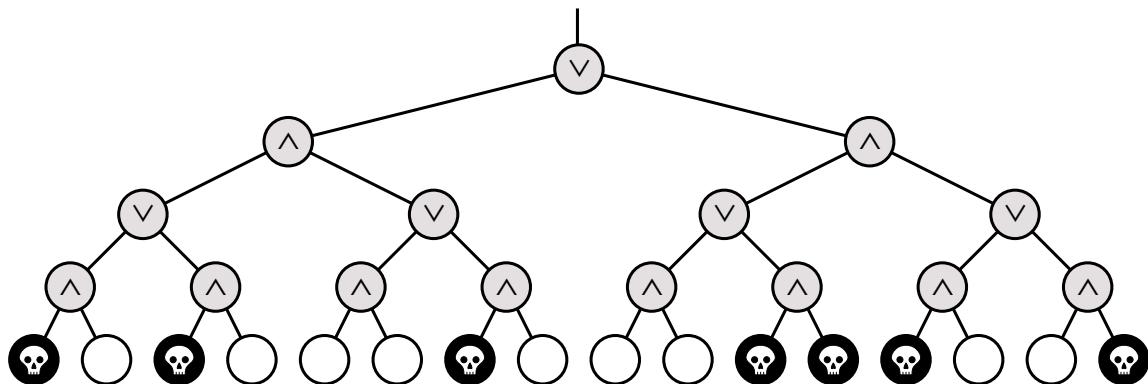


4. Show how to extend the Rabin-Karp fingerprinting method to handle the problem of looking for a given  $m \times m$  pattern in an  $n \times n$  array of characters. (The pattern may be shifted horizontally and vertically, but it may not be rotated.)

5. Death knocks on your door once more on a warm spring day. He remembers that you are an algorithms student and that you soundly defeated him last time and are now living out your immortality. Death is in a bit of a quandry. He has been losing a lot and doesn't know why. He wants you to prove a lower bound on your deterministic algorithm so that he can reap more souls. If you have forgotten, the game goes like this: It is a complete binary tree with  $4^n$  leaves, each colored black or white. There is a token at the root of the tree. To play the game, you and Death took turns moving the token from its current node to one of its children. The game ends after  $2n$  moves, when the token lands on a leaf. If the final leaf is black, the player dies; if it's white, you will live forever. You move first, so Death gets the last turn.

You decided whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are AND gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should've challenged Death to a game of Twister instead.

Prove that *any* deterministic algorithm must examine *every* leaf of the tree in the worst case. Since there are  $4^n$  leaves, this implies that any deterministic algorithm must take  $\Omega(4^n)$  time in the worst case. Use an adversary argument, or in other words, assume Death cheats.



6. [This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]

#### Lower Bounds on Adjacency Matrix Representations of Graphs

- Prove that the time to determine if an undirected graph has a cycle is  $\Omega(V^2)$ .
- Prove that the time to determine if there is a path between two nodes in an undirected graph is  $\Omega(V^2)$ .

## Practice Problems

- String matching with wild-cards

Suppose you have an alphabet for patterns that includes a ‘gap’ or wild-card character; any length string of any characters can match this additional character. For example if ‘\*’ is the wild-card, then the pattern ‘foo\*bar\*nad’ can be found in ‘foofoowangbarnad’. Modify the computation of the prefix function to correctly match strings using KMP.

2. Prove that there is no comparison sort whose running time is linear for at least  $1/2$  of the  $n!$  inputs of length  $n$ . What about at least  $1/n$ ? What about at least  $1/2^n$ ?
3. Prove that  $2n - 1$  comparisons are necessary in the worst case to merge two sorted lists containing  $n$  elements each.
4. Find asymptotic upper and lower bounds to  $\lg(n!)$  without Stirling's approximation (Hint: use integration).
5. Given a sequence of  $n$  elements of  $n/k$  blocks ( $k$  elements per block) all elements in a block are less than those to the right in sequence, show that you cannot have the whole sequence sorted in better than  $\Omega(n \lg k)$ . Note that the entire sequence would be sorted if each of the  $n/k$  blocks were individually sorted in place. Also note that combining the lower bounds for each block is not adequate (that only gives an upper bound).
6. Show how to find the occurrences of pattern  $P$  in text  $T$  by computing the prefix function of the string  $PT$  (the concatenation of  $P$  and  $T$ ).

# CS 373: Combinatorial Algorithms, Spring 2001

<http://www-courses.cs.uiuc.edu/~cs373>

**Homework 6 (due Tue. May 1, 2001 at 11:59.99 p.m.)**

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U,  $\frac{3}{4}$ , or 1, respectively. Staple this sheet to the top of your homework.

**Note:** You will be held accountable for the appropriate responses for answers (e.g. give models, proofs, analyses, etc). For NP-complete problems you should prove everything rigorously, i.e. for showing that it is in NP, give a description of a certificate and a poly time algorithm to verify it, and for showing NP-hardness, you must show that your reduction is polytime (by similarly proving something about the algorithm that does the transformation) and proving both directions of the ‘if and only if’ (a solution of one is a solution of the other) of the many-one reduction.

---

## Required Problems

1. Complexity
  - (a) Prove that  $P \subseteq \text{co-NP}$ .
  - (b) Show that if  $NP \neq \text{co-NP}$ , then *every* NP-complete problem is *not* a member of co-NP.
2. 2-CNF-SAT  
Prove that deciding satisfiability when all clauses have at most 2 literals is in P.
3. Graph Problems

## (a) SUBGRAPH-ISOMORPHISM

Show that the problem of deciding whether one graph is a subgraph of another is NP-complete.

## (b) LONGEST-PATH

Show that the problem of deciding whether an unweighted undirected graph has a path of length greater than  $k$  is NP-complete.

## 4. PARTITION, SUBSET-SUM

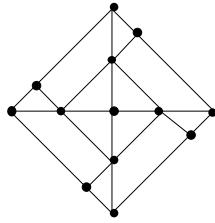
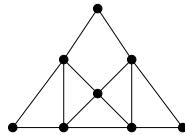
PARTITION is the problem of deciding, given a set of numbers, whether there exists a subset whose sum equals the sum of the complement, i.e. given  $S = s_1, s_2 \dots, s_n$ , does there exist a subset  $S'$  such that  $\sum_{s \in S'} s = \sum_{t \in S - S'} t$ . SUBSET-SUM is the problem of deciding, given a set of numbers and a target sum, whether there exists a subset whose sum equals the target, i.e. given  $S = s_1, s_2 \dots, s_n$  and  $k$ , does there exist a subset  $S'$  such that  $\sum_{s \in S'} s = k$ . Give two reduction, one in both directions.

5. BIN-PACKING Consider the bin-packing problem: given a finite set  $U$  of  $n$  items and the positive integer size  $s(u)$  of each item  $u \in U$ , can  $U$  be partitioned into  $k$  disjoint sets  $U_1, \dots, U_k$  such that the sum of the sizes of the items in each set does not exceed  $B$ ? Show that the bin-packing problem is NP-Complete. [Hint: Use the result from the previous problem.]

## 6. 3SUM

*[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

Describe an algorithm that solves the following problem as quickly as possible: Given a set of  $n$  numbers, does it contain three elements whose sum is zero? For example, your algorithm should answer TRUE for the set  $\{-5, -17, 7, -4, 3, -2, 4\}$ , since  $-5 + 7 + (-2) = 0$ , and FALSE for the set  $\{-6, 7, -4, -13, -2, 5, 13\}$ .

**Figure 1.** Gadget for PLANAR-3-COLOR.**Figure 2.** Gadget for DEGREE-4-PLANAR-3-COLOR.

## Practice Problems

1. Consider finding the median of 5 numbers by using only comparisons. What is the exact worst case number of comparisons needed to find the median. Justify (exhibit a set that cannot be done in one less comparisons). Do the same for 6 numbers.

2. EXACT-COVER-BY-4-SETS

The EXACT-COVER-BY-3-SETS problem is defined as the following: given a finite set  $X$  with  $|X| = 3q$  and a collection  $C$  of 3-element subsets of  $X$ , does  $C$  contain an *exact cover* for  $X$ , that is, a subcollection  $C' \subseteq C$  such that every element of  $X$  occurs in exactly one member of  $C'$ ?

Given that EXACT-COVER-BY-3-SETS is NP-complete, show that EXACT-COVER-BY-4-SETS is also NP-complete.

3. PLANAR-3-COLOR

Using 3-COLOR, and the ‘gadget’ in figure 3, prove that the problem of deciding whether a planar graph can be 3-colored is NP-complete. Hint: show that the gadget can be 3-colored, and then replace any crossings in a planar embedding with the gadget appropriately.

4. DEGREE-4-PLANAR-3-COLOR

Using the previous result, and the ‘gadget’ in figure 4, prove that the problem of deciding whether a planar graph with no vertex of degree greater than four can be 3-colored is NP-complete. Hint: show that you can replace any vertex with degree greater than 4 with a collection of gadgets connected in such a way that no degree is greater than four.

5. Poly time subroutines can lead to exponential algorithms

Show that an algorithm that makes at most a constant number of calls to polynomial-time subroutines runs in polynomial time, but that a polynomial number of calls to polynomial-time subroutines may result in an exponential-time algorithm.

6. (a) Prove that if  $G$  is an undirected bipartite graph with an odd number of vertices, then  $G$  is nonhamiltonian. Give a polynomial time algorithm for finding a **hamiltonian cycle** in an undirected bipartite graph or establishing that it does not exist.  
(b) Show that the **hamiltonian-path** problem can be solved in polynomial time on directed acyclic graphs by giving an efficient algorithm for the problem.  
(c) Explain why the results in previous questions do not contradict the facts that both HAM-CYCLE and HAM-PATH are NP-complete problems.
7. Consider the following pairs of problems:
  - (a) MIN SPANNING TREE and MAX SPANNING TREE
  - (b) SHORTEST PATH and LONGEST PATH
  - (c) TRAVELING SALESMAN PROBLEM and VACATION TOUR PROBLEM (the longest tour is sought).
  - (d) MIN CUT and MAX CUT (between  $s$  and  $t$ )
  - (e) EDGE COVER and VERTEX COVER
  - (f) TRANSITIVE REDUCTION and MIN EQUIVALENT DIGRAPH

(all of these seem dual or opposites, except the last, which are just two versions of minimal representation of a graph).

Which of these pairs are polytime equivalent and which are not? Why?
- ★8. GRAPH-ISOMORPHISM  
Consider the problem of deciding whether one graph is isomorphic to another.
  - (a) Give a brute force algorithm to decide this.
  - (b) Give a dynamic programming algorithm to decide this.
  - (c) Give an efficient probabilistic algorithm to decide this.
  - (d) Either prove that this problem is NP-complete, give a poly time algorithm for it, or prove that neither case occurs.
9. Prove that PRIMALITY (Given  $n$ , is  $n$  prime?) is in  $\text{NP} \cap \text{co-NP}$ . Hint: co-NP is easy (what's a certificate for showing that a number is composite?). For NP, consider a certificate involving primitive roots and recursively their primitive roots. Show that knowing this tree of primitive roots can be checked to be correct and used to show that  $n$  is prime, and that this check takes poly time.
10. How much wood would a woodchuck chuck if a woodchuck could chuck wood?

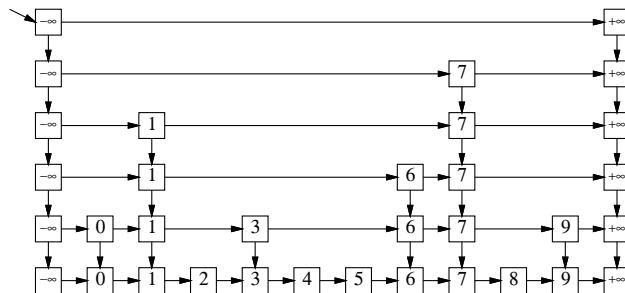
Write your answers in the separate answer booklet.

1. **Multiple Choice:** Each question below has one of the following answers.

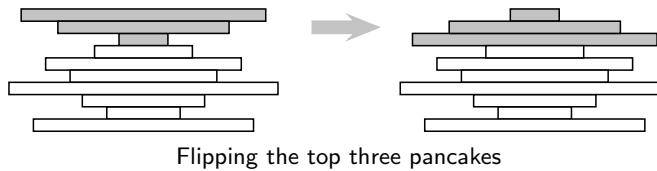
(a)  $\Theta(1)$       (b)  $\Theta(\log n)$       (c)  $\Theta(n)$       (d)  $\Theta(n \log n)$       (e)  $\Theta(n^2)$

For each question, write the letter that corresponds to your answer. You do not need to justify your answers. Each correct answer earns you 1 point, but each incorrect answer costs you  $\frac{1}{2}$  point. You cannot score below zero.

- (a) What is  $\sum_{i=1}^n H_i$ ?
  - (b) What is  $\sum_{i=1}^{\lg n} 2^i$ ?
  - (c) How many digits do you need to write  $n!$  in decimal?
  - (d) What is the solution of the recurrence  $T(n) = 16T(n/4) + n$ ?
  - (e) What is the solution of the recurrence  $T(n) = T(n - 2) + \lg n$ ?
  - (f) What is the solution of the recurrence  $T(n) = 4T\left(\lceil \frac{n+51}{4} \rceil - \sqrt{n}\right) + 17n - 2^{8 \log^*(n^2)} + 6$ ?
  - (g) What is the worst-case running time of randomized quicksort?
  - (h) The expected time for inserting one item into a treap is  $O(\log n)$ . What is the worst-case time for a sequence of  $n$  insertions into an initially empty treap?
  - (i) The amortized time for inserting one item into an  $n$ -node splay tree is  $O(\log n)$ . What is the worst-case time for a sequence of  $n$  insertions into an initially empty splay tree?
  - (j) In the worst case, how long does it take to solve the traveling salesman problem for 10,000,000,000,000,000 cities?
2. What is the *exact* expected number of nodes in a skip list storing  $n$  keys, *not* counting the sentinel nodes at the beginning and end of each level? Justify your answer. A correct  $\Theta()$  bound (with justification) is worth 5 points.



3. Suppose we have a stack of  $n$  pancakes of all different sizes. We want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation we can perform is a *flip* — insert a spatula under the top  $k$  pancakes, for some  $k$  between 1 and  $n$ , turn them all over, and put them back on top of the stack.



Flipping the top three pancakes

- (a) (3 pts) Describe an algorithm to sort an arbitrary stack of  $n$  pancakes using flips.
- (b) (3 pts) Prove that your algorithm is correct.
- (c) (2 pts) Exactly how many flips does your sorting algorithm perform in the worst case? A correct  $\Theta()$  bound is worth one point.
- (d) (2 pts) Suppose one side of each pancake is burned. Exactly how many flips do you need to sort the pancakes, so that the burned side of every pancake is on the bottom? A correct  $\Theta()$  bound is worth one point.

4. Suppose we want to maintain a set of values in a data structure subject to the following operations:

- $\text{INSERT}(x)$ : Add  $x$  to the set (if it isn't already there).
- $\text{DELETERANGE}(a, b)$ : Delete every element  $x$  in the range  $a \leq x \leq b$ . For example, if the set was  $\{1, 5, 3, 4, 8\}$ , then  $\text{DELETERANGE}(4, 6)$  would change the set to  $\{1, 3, 8\}$ .

Describe and analyze a data structure that supports these operations, such that the amortized cost of either operation is  $O(\log n)$ . [Hint: Use a data structure you saw in class. If you use the same  $\text{INSERT}$  algorithm, just say so—you don't need to describe it again in your answer.]

5. [1-unit grad students must answer this question.]

A *shuffle* of two strings  $X$  and  $Y$  is formed by interspersing the characters into a new string, keeping the characters of  $X$  and  $Y$  in the same order. For example, ‘bananaananas’ is a shuffle of ‘banana’ and ‘ananas’ in several different ways.

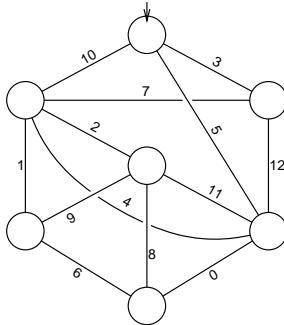
banana ananas      ban ana ana nas      b an an a na na s

The strings ‘prodgyrnammamciing’ and ‘dyprongarmammicing’ are both shuffles of ‘dynamic’ and ‘programming’:

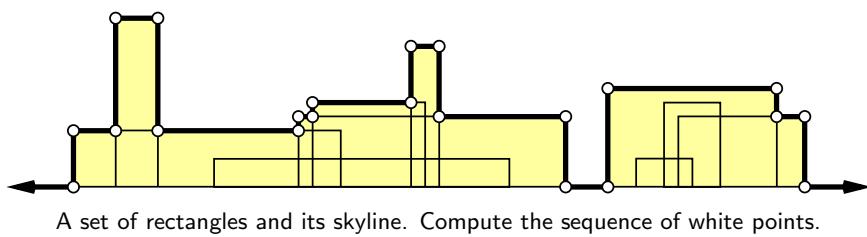
pro<sup>d</sup>g y r nam ammi i n c g      dy pro<sup>n</sup> g a r m amm ic ing

Given three strings  $A[1..m]$ ,  $B[1..n]$ , and  $C[1..m+n]$ , describe and analyze an algorithm to determine whether  $C$  is a shuffle of  $A$  and  $B$ . For full credit, your algorithm should run in  $\Theta(mn)$  time.

1. Using any method you like, compute the following subgraphs for the weighted graph below. Each subproblem is worth 3 points. Each incorrect edge costs you 1 point, but you cannot get a negative score for any subproblem.
- a depth-first search tree, starting at the top vertex;
  - a breadth-first search tree, starting at the top vertex;
  - a shortest path tree, starting at the top vertex;
  - the **maximum** spanning tree.



2. (a) [4 pts] Prove that a connected acyclic undirected graph with  $V$  vertices has exactly  $V - 1$  edges. ("It's a tree!" is not a proof.)
- (b) [4 pts] Describe and analyze an algorithm that determines whether a given undirected graph is a tree, where the graph is represented by an adjacency list.
- (c) [2 pts] What is the running time of your algorithm from part (b) if the graph is represented by an adjacency matrix?
3. Suppose we want to sketch the Manhattan skyline (minus the interesting bits like the Empire State and Chrysler buildings). You are given a set of  $n$  rectangles, each rectangle represented by its left and right  $x$ -coordinates and its height. The bottom of each rectangle is on the  $x$ -axis. Describe and analyze an efficient algorithm to compute the vertices of the skyline.



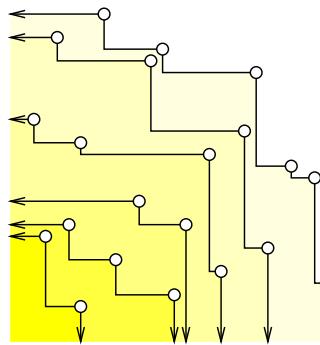
4. Suppose we model a computer network as a weighted undirected graph, where each vertex represents a computer and each edge represents a *direct* network connection between two computers. The weight of each edge represents the *bandwidth* of that connection—the number of bytes that can flow from one computer to the other in one second.<sup>1</sup> We want to implement a point-to-point network protocol that uses a single dedicated path to communicate between any pair of computers. Naturally, when two computers need to communicate, we should use the path with the highest bandwidth. The bandwidth of a *path* is the *minimum* bandwidth of its edges.

Describe an algorithm to compute the maximum bandwidth path between *every* pair of computers in the network. Assume that the graph is represented as an adjacency list.

5. [1-unit grad students must answer this question.]

Let  $P$  be a set of points in the plane. Recall that the *staircase* of  $P$  contains all the points in  $P$  that have no other point in  $P$  both above and to the right. We can define the *staircase layers* of  $P$  recursively as follows. The first staircase layer is just the staircase; for all  $i > 1$ , the  $i$ th staircase layer is the staircase of  $P$  after the first  $i - 1$  staircase layers have been deleted.

Describe and analyze an algorithm to compute the staircase layers of  $P$  in  $O(n^2)$  time.<sup>2</sup> Your algorithm should label each point with an integer describing which staircase layer it belongs to. You can assume that no two points have the same  $x$ - or  $y$ -coordinates.



A set of points and its six staircase layers.

---

<sup>1</sup>Notice the bandwidth is symmetric; there are no cable modems or wireless phones. Don't worry about systems-level stuff like network load and latency. After all, this is a theory class!

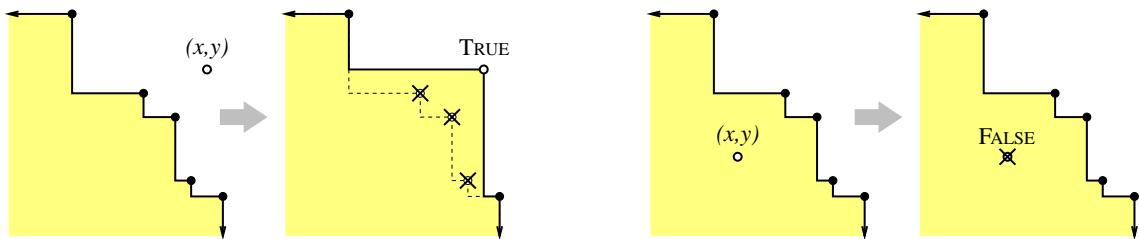
<sup>2</sup>This is *not* the fastest possible running time for this problem.

**You must turn in this question sheet with your answers.****1. Déjà vu**

Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers—if  $F_n$  appears in the sum, then neither  $F_{n+1}$  nor  $F_{n-1}$  will. For example:  $42 = F_9 + F_6$ ,  $25 = F_8 + F_4 + F_2$ , and  $17 = F_7 + F_4 + F_2$ . You must give a complete, self-contained proof, not just a reference to the posted homework solutions.

**2. L'esprit d'escalier**

Recall that the *staircase* of a set of points consists of the points with no other point both above and to the right. Describe a method to maintain the staircase as new points are added to the set. Specifically, describe and analyze a data structure that stores the staircase of a set of points, and an algorithm  $\text{INSERT}(x, y)$  that adds the point  $(x, y)$  to the set and returns TRUE or FALSE to indicate whether the staircase has changed. Your data structure should use  $O(n)$  space, and your  $\text{INSERT}$  algorithm should run in  $O(\log n)$  amortized time.

**3. Engage le jeu que je le gagne**

A palindrome is a text string that is exactly the same as its reversal, such as DEED, RACECAR, or SAIPPUAKAUPPIAS.<sup>1</sup>

- Describe and analyze an algorithm to find the longest *prefix* of a given string that is also a palindrome. For example, the longest palindrome prefix of ILLINOISURBANACHAMPAIGN is ILLI, and the longest palindrome prefix of HYAKUGOJYUUICHI<sup>2</sup> is the single letter S. For full credit, your algorithm should run in  $O(n)$  time.
- Describe and analyze an algorithm to find the length of the longest *subsequence* of a given string that is also a palindrome. For example, the longest palindrome subsequence of ILLINOISURBANACHAMPAIGN is NIAACAAIN (or NIAAHAIN), and the longest palindrome subsequence of HYAKUGOJYUUICHI is HUUUH<sup>3</sup> (or HUGUH or HUYUH or...). You do not need to compute the actual subsequence; just its length. For full credit, your algorithm should run in  $O(n^2)$  time.

<sup>1</sup>Finnish for ‘soap dealer’.

<sup>2</sup>Japanese for ‘one hundred fifty-one’.

<sup>3</sup>English for ‘What the heck are you talking about?’

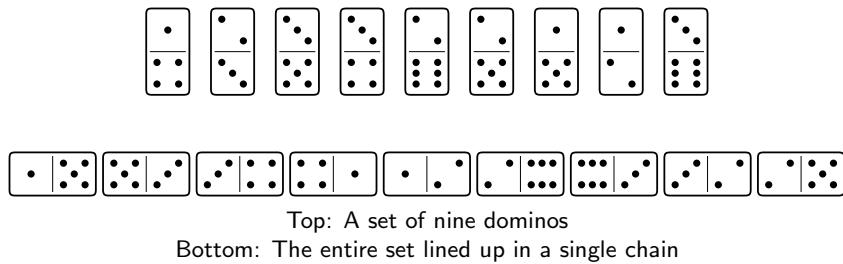
#### 4. Toute votre base sont appartientent à nous

Prove that *exactly*  $2n - 1$  comparisons are required in the worst case to merge two sorted arrays, each with  $n$  distinct elements. Describe and analyze an algorithm to prove the upper bound, and use an adversary argument to prove the lower bound. *You must give a complete, self-contained solution, not just a reference to the posted homework solutions.*<sup>4</sup>

#### 5. Plus ça change, plus ça même chose

A domino is a  $2 \times 1$  rectangle divided into two squares, with a certain number of pips (dots) in each square. In most domino games, the players lay down dominos at either end of a single chain. Adjacent dominos in the chain must have matching numbers. (See the figure below.)

Describe and analyze an efficient algorithm, or prove that it is NP-hard, to determine whether a given set of  $n$  dominos can be lined up in a single chain. For example, for the set of dominos shown below, the correct output is TRUE.



#### 6. Ceci n'est pas une pipe

Consider the following pair of problems:

- **BOXDEPTH:** Given a set of  $n$  axis-aligned rectangles in the plane and an integer  $k$ , decide whether any point in the plane is covered by  $k$  or more rectangles.
  - **MAXCLIQUE:** Given a graph with  $n$  vertices and an integer  $k$ , decide whether the graph contains a clique with  $k$  or more vertices.
- Describe and analyze a reduction of one of these problems to the other.
  - MAXCLIQUE is NP-hard. What does your answer to part (a) imply about the complexity of BOXDEPTH?

#### 7. C'est magique! [1-unit graduate students must answer this question.]

The recursion fairy's cousin, the reduction genie, shows up one day with a magical gift for you—a box that determines in constant time the size of the largest clique in any given graph. (Recall that a clique is a subgraph where every pair of vertices is joined by an edge.) The magic box does not tell you *where* the largest clique is, only its size. Describe and analyze an algorithm to actually find the largest clique in a given graph **in polynomial time**, using this magic box.

---

<sup>4</sup>The posted solution for this Homework 5 practice problem was incorrect. So don't use it!

# CS 373: Combinatorial Algorithms, Fall 2002

## Homework 0, due September 5, 2002 at the beginning of class

Name:		
Net ID:	Alias:	U G

Neatly print your name (first name first, with no comma), your network ID, and an alias of your choice into the boxes above. Circle U if you are an undergraduate, and G if you are a graduate student. **Do not sign your name. Do not write your Social Security number.** Staple this sheet of paper to the top of your homework.

Grades will be listed on the course web site by alias give us, so your alias should not resemble your name or your Net ID. If you don't give yourself an alias, we'll give you one that you won't like.

---

Before you do anything else, please read the Homework Instructions and FAQ on the CS 373 course web page (<http://www-courses.cs.uiuc.edu/~cs373/hwx/faq.html>) and then check the box below. There are 300 students in CS 373 this semester; we are quite serious about giving zeros to homeworks that don't follow the instructions.

I have read the CS 373 Homework Instructions and FAQ.

---

Every CS 373 homework has the same basic structure. There are six required problems, some with several subproblems. Each problem is worth 10 points. Only graduate students are required to answer problem 6; undergraduates can turn in a solution for extra credit. There are several practice problems at the end. Stars indicate problems we think are hard.

---

This homework tests your familiarity with the prerequisite material from CS 173, CS 225, and CS 273, primarily to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Rosen (the 173/273 textbook), CLRS (especially Chapters 1–7, 10, 12, and A–C), and the lecture notes on recurrences should be sufficient review, but you may want to consult other texts as well.

---

## Required Problems

1. Sort the following functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. Please *don't* turn in proofs, but you should do them anyway to make sure you're right (and for practice).

$$\begin{array}{ccccc}
 1 & n & n^2 & \lg n & n \lg n \\
 n^{\lg n} & (\lg n)^n & (\lg n)^{\lg n} & n^{\lg \lg n} & n^{1/\lg n} \\
 \log_{1000} n & \lg^{1000} n & \lg^{(1000)} n & \lg(n^{1000}) & \left(1 + \frac{1}{1000}\right)^n
 \end{array}$$

To simplify notation, write  $f(n) \ll g(n)$  to mean  $f(n) = o(g(n))$  and  $f(n) \equiv g(n)$  to mean  $f(n) = \Theta(g(n))$ . For example, the functions  $n^2$ ,  $n$ ,  $\binom{n}{2}$ ,  $n^3$  could be sorted either as  $n \ll n^2 \equiv \binom{n}{2} \ll n^3$  or as  $n \ll \binom{n}{2} \equiv n^2 \ll n^3$ .

2. Solve these recurrences. State tight asymptotic bounds for each function in the form  $\Theta(f(n))$  for some recognizable function  $f(n)$ . Please *don't* turn in proofs, but you should do them anyway just for practice. Assume reasonable but nontrivial base cases, and state them if they affect your solution. Extra credit will be given for more exact solutions. [Hint: Most of these are very easy.]

$$\begin{array}{ll}
 A(n) = 2A(n/2) + n & F(n) = 9F(\lfloor n/3 \rfloor + 9) + n^2 \\
 B(n) = 3B(n/2) + n & G(n) = 3G(n-1)/5G(n-2) \\
 C(n) = 2C(n/3) + n & H(n) = 2H(\sqrt{n}) + 1 \\
 D(n) = 2D(n-1) + 1 & I(n) = \min_{1 \leq k \leq n/2} (I(k) + I(n-k) + k) \\
 E(n) = \max_{1 \leq k \leq n/2} (E(k) + E(n-k) + n) & *J(n) = \max_{1 \leq k \leq n/2} (J(k) + J(n-k) + k)
 \end{array}$$

3. Recall that a binary tree is *full* if every node has either two children (an internal node) or no children (a leaf). Give at least *four different* proofs of the following fact:

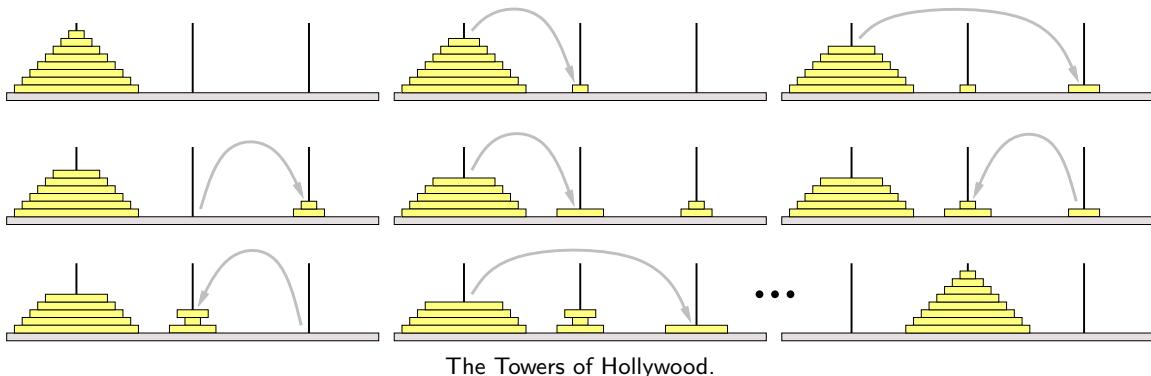
*In any full binary tree, the number of leaves is exactly one more than the number of internal nodes.*

For full credit, each proof must be self-contained, the proof must be substantially different from each other, and at least one proof must *not* use induction. For each  $n$ , your  $n$ th correct proof is worth  $n$  points, so you need four proofs to get full credit. Each correct proof beyond the fourth earns you extra credit. [Hint: I know of at least six different proofs.]

4. Most of you are probably familiar with the story behind the Tower of Hanoi puzzle:<sup>1</sup>

At the great temple of Benares, there is a brass plate on which three vertical diamond shafts are fixed. On the shafts are mounted  $n$  golden disks of decreasing size.<sup>2</sup> At the time of creation, the god Brahma placed all of the disks on one pin, in order of size with the largest at the bottom. The Hindu priests unceasingly transfer the disks from peg to peg, one at a time, never placing a larger disk on a smaller one. When all of the disks have been transferred to the last pin, the universe will end.

Recently the temple at Benares was relocated to southern California, where the monks are considerably more laid back about their job. At the “Towers of Hollywood”, the golden disks were replaced with painted plywood, and the diamond shafts were replaced with Plexiglas. More importantly, the restriction on the order of the disks was relaxed. While the disks are being moved, the *bottom* disk on any pin must be the *largest* disk on that pin, but disks further up in the stack can be in any order. However, after all the disks have been moved, they must be in sorted order again.



Describe an algorithm<sup>3</sup> that moves a stack of  $n$  disks from one pin to the another using the smallest possible number of moves. For full credit, your algorithm should be non-recursive, but a recursive algorithm is worth significant partial credit. *Exactly* how many moves does your algorithm perform? [Hint: The Hollywood monks can bring about the end of the universe quite a bit faster than the original monks at Benares could.]

The problem of computing the minimum number of moves was posed in the most recent issue of the *American Mathematical Monthly* (August/September 2002). No solution has been published yet.

---

<sup>1</sup>The puzzle and the accompanying story were both invented by the French mathematician Eduoard Lucas in 1883. See <http://www.cs.wm.edu/~pkstoc/toh.html>

<sup>2</sup>In the original legend,  $n = 64$ . In the 1883 wooden puzzle,  $n = 8$ .

<sup>3</sup>Since you've read the Homework Instructions, you know exactly what this phrase means.

5. On their long journey from Denmark to England, Rosencrantz and Guildenstern amuse themselves by playing the following game with a fair coin. First Rosencrantz flips the coin over and over until it comes up tails. Then Guildenstern flips the coin over and over until he gets as many heads in a row as Rosencrantz got on his turn. Here are three typical games:

Rosencrantz: H H T

Guildenstern: H T H H

Rosencrantz: T

Guildenstern: (no flips)

Rosencrantz: H H H T

Guildenstern: T H H T H H T H T H H H

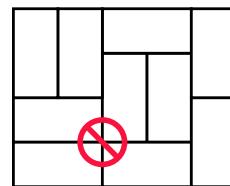
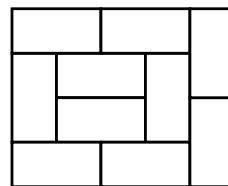
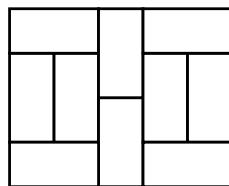
- (a) What is the expected number of flips in one of Rosencrantz's turns?
- (b) Suppose Rosencrantz flips  $k$  heads in a row on his turn. What is the expected number of flips in Guildenstern's next turn?
- (c) What is the expected total number of flips (by both Rosencrantz and Guildenstern) in a single game?

*Prove* your answers are correct. If you have to appeal to “intuition” or “common sense”, your answer is almost certainly wrong! You must give exact answers for full credit, but asymptotic bounds are worth significant partial credit.

6. [This problem is required only for graduate students (including I2CS students); undergrads can submit a solution for extra credit.]

Tatami are rectangular mats used to tile floors in traditional Japanese houses. Exact dimensions of tatami mats vary from one region of Japan to the next, but they are always twice as long in one dimension than in the other. (In Tokyo, the standard size is 180cm×90cm.)

- (a) How many different ways are there to tile a  $2 \times n$  rectangular room with  $1 \times 2$  tatami mats? Set up a recurrence and derive an *exact* closed-form solution. [Hint: The answer involves a familiar recursive sequence.]
- (b) According to tradition, tatami mats are always arranged so that four corners never meet. Thus, the first two arrangements below are traditional, but not the third.



Two traditional tatami arrangements and one non-traditional arrangement.

How many different *traditional* ways are there to tile a  $3 \times n$  rectangular room with  $1 \times 2$  tatami mats? Set up a recurrence and derive an *exact* closed-form solution.

- \*(c) [5 points extra credit] How many different *traditional* ways are there to tile an  $n \times n$  square with  $1 \times 2$  tatami mats? Prove your answer is correct.

## Practice Problems

These problems are only for your benefit; other problems can be found in previous semesters' homeworks on the course web site. You are *strongly* encouraged to do some of these problems as additional practice. Think of them as potential exam questions (hint, hint). Feel free to ask about any of these questions on the course newsgroup, during office hours, or during review sessions.

1. Removing any edge from a binary tree with  $n$  nodes partitions it into two smaller binary trees. If both trees have at least  $\lceil(n - 1)/3\rceil$  nodes, we say that the partition is *balanced*.
  - (a) Prove that every binary tree with more than one vertex has a balanced partition. [Hint: I know of at least two different proofs.]
  - (b) If each smaller tree has more than  $\lfloor n/3 \rfloor$  nodes, we say that the partition is *strictly balanced*. Show that for every  $n$ , there is an  $n$ -node binary tree with *no* strictly balanced partition.
2. Describe an algorithm COUNTTOTENTOTHE( $n$ ) that prints the integers from 1 to  $10^n$ .

Assume you have a subroutine PRINTDIGIT( $d$ ) that prints any integer  $d$  between 0 and 9, and another subroutine PRINTSPACE that prints a space character. Both subroutines run in  $O(1)$  time. You may want to write (and analyze) a separate subroutine PRINTINTEGER to print an arbitrary integer.

Since integer variables cannot store arbitrarily large values in most programming languages, your algorithm must not store any value larger than  $\max\{10, n\}$  in any single integer variable. Thus, the following algorithm is **not correct**:

```
BOGUSCOUNTTOTENTOTHE( $n$ ):
  for  $i \leftarrow 1$  to POWER( $10, n$ )
    PRINTINTEGER( $i$ )
```

(So what exactly *can* you pass to PRINTINTEGER?)

What is the running time of your algorithm (as a function of  $n$ )? How many digits and spaces does it print? How much space does it use?

3. I'm sure you remember the following simple rules for taking derivatives:

- Simple cases:  $\frac{d}{dx}\alpha = 0$  for any constant  $\alpha$ , and  $\frac{d}{dx}x = 1$
- Linearity:  $\frac{d}{dx}(f(x) + g(x)) = f'(x) + g'(x)$
- The product rule:  $\frac{d}{dx}(f(x) \cdot g(x)) = f'(x) \cdot g(x) + f(x) \cdot g'(x)$
- The chain rule:  $\frac{d}{dx}(f(g(x))) = f'(g(x)) \cdot g'(x)$

Using *only* these rules and induction, prove that  $\frac{d}{dx}x^c = cx^{c-1}$  for any integer  $c \neq -1$ . Do not use limits, integrals, or any other concepts from calculus, except for the simple identities listed above. [Hint: Don't forget about negative values of  $c$ !]

4. This problem asks you to calculate the total resistance between two points in a series-parallel resistor network. Don't worry if you haven't taken a circuits class; everything you need to know can be summed up in two sentences and a picture.

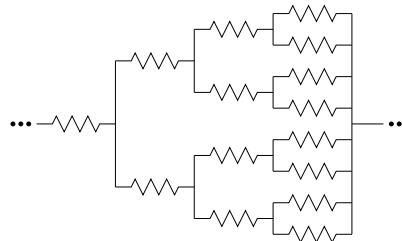
- The total resistance of two resistors in *series* is the sum of their individual resistances.
- The total resistance of two resistors in *parallel* is the reciprocal of the sum of the reciprocals of their individual resistances.

$$\dots - \begin{smallmatrix} \nearrow \\ \searrow \end{smallmatrix} \begin{smallmatrix} x \\ y \end{smallmatrix} - \begin{smallmatrix} \nearrow \\ \searrow \end{smallmatrix} - \dots = \dots - \begin{smallmatrix} \nearrow \\ \searrow \end{smallmatrix} \begin{smallmatrix} x+y \\ \phantom{x+y} \end{smallmatrix} - \dots \quad \dots - \boxed{\begin{smallmatrix} \nearrow \\ \searrow \end{smallmatrix} \begin{smallmatrix} x \\ y \end{smallmatrix}} - \dots = \dots - \begin{smallmatrix} \nearrow \\ \searrow \end{smallmatrix} \frac{I}{I/x + I/y} - \dots$$

Equivalence laws for series-parallel resistor networks.

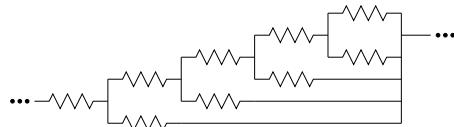
What is the *exact* total resistance<sup>4</sup> of the following resistor networks as a function of  $n$ ? Prove your answers are correct. [Hint: Use induction. Duh.]

- (a) A complete binary tree with depth  $n$ , with a  $1\Omega$  resistor at every node, and a common wire joining all the leaves. Resistance is measured between the root and the leaves.



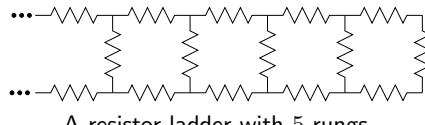
A balanced binary resistor tree with depth 3.

- (b) A totally unbalanced full binary tree with depth  $n$  (every internal node has two children, one of which is a leaf) with a  $1\Omega$  resistor at every node, and a common wire joining all the leaves. Resistance is measured between the root and the leaves.



A totally unbalanced binary resistor tree with depth 4.

- \*(c) A ladder with  $n$  rungs, with a  $1\Omega$  resistor on every edge. Resistance is measured between the bottom of the legs.



A resistor ladder with 5 rungs.

---

<sup>4</sup>The ISO standard unit of resistance is the Ohm, written with the symbol  $\Omega$ . Don't confuse this with the asymptotic notation  $\Omega(f(n))$ !

# CS 373: Combinatorial Algorithms, Fall 2002

## Homework 1, due September 17, 2002 at 23:59:59

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Undergrads	
Grads	

This homework is to be submitted in groups of up to three people. Graduate and undergraduate students are *not* allowed to work in the same group. Please indicate above whether you are undergraduate or graduate students. Only *one* submission per group will be accepted.

## Required Problems

1. The traditional Devonian/Cornish drinking song “The Barley Mow” has the following pseudolyrics, where *container*[ $i$ ] is the name of a container<sup>1</sup> that holds  $2^i$  ounces of beer.

BARLEYMOW( $n$ ):

"Here's a health to the barley-mow, my brave boys,"  
"Here's a health to the barley-mow!"

"We'll drink it out of the jolly brown bowl,"  
"Here's a health to the barley-mow!"

"Here's a health to the barley-mow, my brave boys,"  
"Here's a health to the barley-mow!"

for  $i \leftarrow 1$  to  $n$

    "We'll drink it out of the container[ $i$ ], boys,"  
    "Here's a health to the barley-mow!"

    for  $j \leftarrow i$  downto 1

        "The container[ $j$ ],"  
        "And the jolly brown bowl!"

        "Here's a health to the barley-mow, my brave boys,"  
        "Here's a health to the barley-mow!"

- (a) Suppose each container name  $container[i]$  is a single word, and you can sing four words a second. How long would it take you to sing  $\text{BARLEYMOW}(n)$ ? (Give a tight asymptotic bound.)

<sup>1</sup>One version of the song uses the following containers: nipperkin, gill pot, half-pint, pint, quart, pottle, gallon, half-anker, anker, firkin, half-barrel, barrel, hogshead, pipe, well, river, and ocean. Every container in this list is twice as big as its predecessor, except that a firkin is actually 2.25 ankers, and the last three units are just silly.

- (b) Suppose  $\text{container}[n]$  has  $\Theta(\log n)$  syllables, and you can sing six syllables per second. Now how long would it take you to sing BARLEYMOW( $n$ )? (Give a tight asymptotic bound.)
- (c) Suppose each time you mention the name of a container, you drink the corresponding amount of beer: one ounce for the jolly brown bowl, and  $2^i$  ounces for each  $\text{container}[i]$ . Assuming for purposes of this problem that you are over 21, *exactly* how many ounces of beer would you drink if you sang BARLEYMOW( $n$ )? (Give an *exact* answer, not just an asymptotic bound.)
2. Suppose you have a set  $S$  of  $n$  numbers. Given two elements you *cannot* determine which is larger. However, you are given an oracle that will tell you the median of a set of three elements.
- Give a linear time algorithm to find the pair of the largest and smallest numbers in  $S$ .
  - Give an algorithm to sort  $S$  in  $O(n \lg n)$  time.
3. Given a black and white pixel image  $A[1 \dots m][1 \dots n]$ , our task is to represent  $A$  with a search tree  $T$ . Given a query  $(x, y)$ , a simple search on  $T$  should return the color of pixel  $A[x][y]$ . The algorithm to construct  $T$  will be as follows.

```

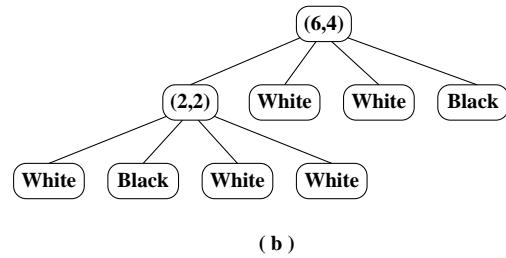
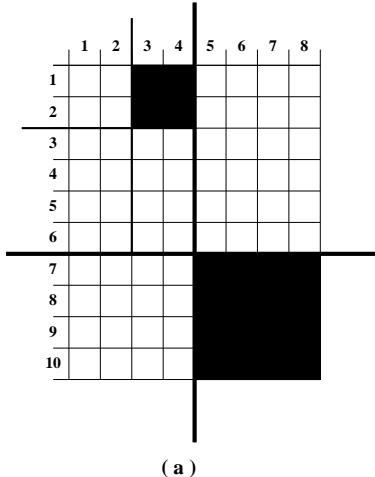
CONSTRUCTSEARCHTREE( $A[1 \dots m][1 \dots n]$ ):
  //Base Case
  if  $A$  contains only one color
    return a leaf node labeled with that color

  //Recurse on Subtrees
   $(i, j) \leftarrow \text{CHOOSECUT}(A[1 \dots m][1 \dots n])$ 
   $T_1 \leftarrow \text{CONSTRUCTSEARCHTREE}(A[1 \dots i][1 \dots j])$ 
   $T_2 \leftarrow \text{CONSTRUCTSEARCHTREE}(A[1 \dots i][j + 1 \dots n])$ 
   $T_3 \leftarrow \text{CONSTRUCTSEARCHTREE}(A[i + 1 \dots m][1 \dots j])$ 
   $T_4 \leftarrow \text{CONSTRUCTSEARCHTREE}(A[i + 1 \dots m][j + 1 \dots n])$ 

  //Construct the Root
   $T.\text{cut} \leftarrow (i, j)$ 
   $T.\text{children} \leftarrow T_1, T_2, T_3, T_4$ 
  return  $T$ 

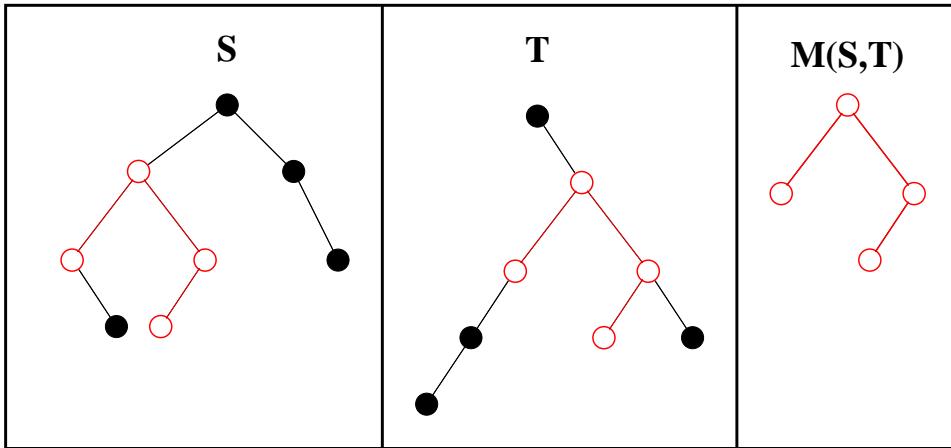
```

That is, this algorithm divides a multicolor image into quadrants and recursively constructs the search tree for each quadrant. Upon a query  $(x, y)$  of  $T$  (assuming  $1 \leq x \leq m$  and  $1 \leq y \leq n$ ), the appropriate subtree is searched. When the correct leaf node is reached, the pixel color is returned. Here's a toy example.

(a) An image  $A$  and the chosen cuts. (b) The corresponding search tree.

Your job in this problem is to give an algorithm for CHOOSECUT. The sequence of chosen cuts must result in an optimal search tree  $T$ . That is, the expected search depth of a uniformly chosen pixel must be minimized. You may use any external data structures (*i.e.* a global table) that you find necessary. You may also preprocess in order to initialize these structures before the initial call to CONSTRUCTSEARCHTREE( $A[1 \dots m][1 \dots n]$ ).

4. Let  $A$  be a set of  $n$  positive integers, all of which are no greater than some constant  $M > 0$ . Give an  $O(n^2M)$  time algorithm to determine whether or not it is possible to split  $A$  into two subsets such that the sum of the numbers in each subset are equal.
5. Let  $S$  and  $T$  be two binary trees. A *matching* of  $S$  and  $T$  is a tree  $M$  which is isomorphic to some subtree in each of  $S$  and  $T$ . Here's an illustration.

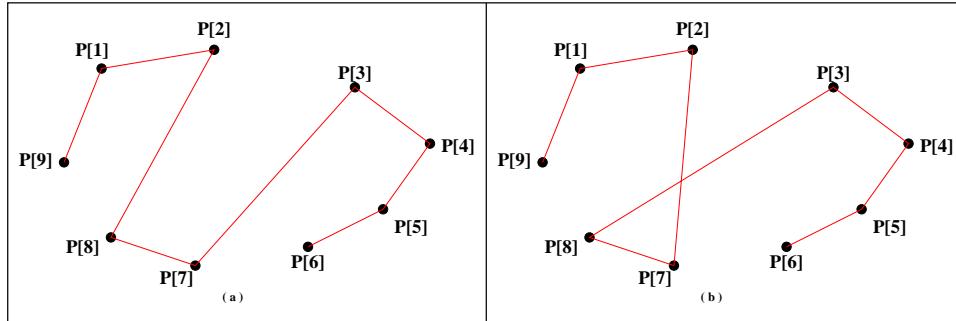
A matching  $M(S, T)$  of binary trees  $S$  and  $T$ .

A *maximal* matching is a matching which contains at least as many vertices as any other matching. Give an algorithm to compute a maximal matching given the roots of two binary trees. Your algorithm should return the size of the match as well as the two roots of the matched subtrees of  $S$  and  $T$ .

6. [This problem is required only for graduate students (including I2CS students); undergrads can submit a solution for extra credit.]

Let  $P[1, \dots, n]$  be a set of  $n$  convex points in the plane. Intuitively, if a rubber band were stretched to surround  $P$  then each point would touch the rubber band. Furthermore, suppose that the points are labeled such that  $P[1], \dots, P[n]$  is a simple path along the convex hull (*i.e.*  $P[i]$  is adjacent to  $P[i + 1]$  along the rubber band).

- (a) Give a simple algorithm to compute a shortest *cyclic* tour of  $P$ .
- (b) A *monotonic* tour of  $P$  is a tour that never crosses itself. Here's an illustration.



(a) A monotonic tour of  $P$ . (b) A non-monotonic tour of  $P$ .

*Prove* that any shortest tour of  $P$  must be monotonic.

- (c) Given an algorithm to compute a shortest tour of  $P$  starting at point  $P[1]$  and finishing on point  $P[\lfloor \frac{n}{2} \rfloor]$ .

## Practice Problems

These remaining practice problems are entirely for your benefit. Don't turn in solutions—we'll just throw them out—but feel free to ask us about these questions during office hours and review sessions. Think of these as potential exam questions (hint, hint).

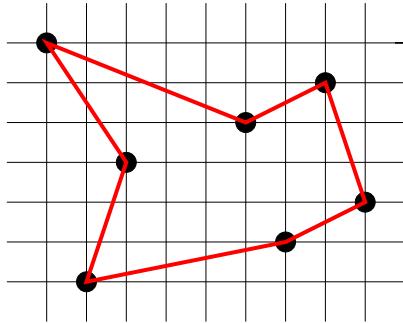
1. Suppose that you are given an  $n \times n$  checkerboard and a checker. You must move the checker from the bottom edge of the board to the top edge of the board according to the following rule. At each step you may move the checker to one of three squares:
  - (a) the square immediately above,
  - (b) the square that is one up and one left (but only if the checker is not already in the leftmost column),
  - (c) the square that is one up and one right (but only if the checker is not already in the rightmost column).

Each time you move from square  $x$  to square  $y$ , you receive  $p(x, y)$  dollars. You are given  $p(x, y)$  for all pairs  $(x, y)$  for which a move from  $x$  to  $y$  is legal. Do not assume that  $p(x, y)$  is positive.

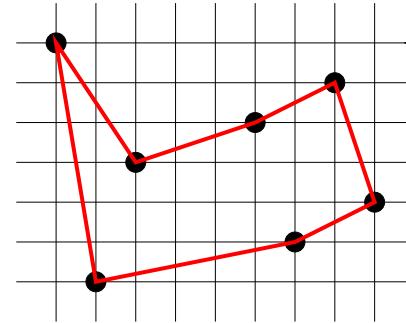
Give an algorithm that figures out the set of moves that will move the checker from somewhere along the bottom edge to somewhere along the top edge while gathering as many dollars as possible. Your algorithm is free to pick any square along the bottom edge as a starting point and any square along the top edge as a destination in order to maximize the number of dollars gathered along the way. What is the running time of your algorithm?

2. (CLRS 15-1) The *euclidean traveling-salesman problem* is the problem of determining the shortest closed tour that connects a given set of  $n$  points in the plane. Figure (a) below shows the solution to a 7-point problem. The general problem is NP-complete, and its solution is therefore believed to require more than polynomial time.

J.L. Bentley has suggested that we simplify the problem by restricting our attention to *bitonic tours* (Figure (b) below). That is, tours that start at the leftmost point, go strictly left to right to the rightmost point, and then go strictly right to left back to the starting point. In this case, a polynomial-time algorithm is possible.



(a)



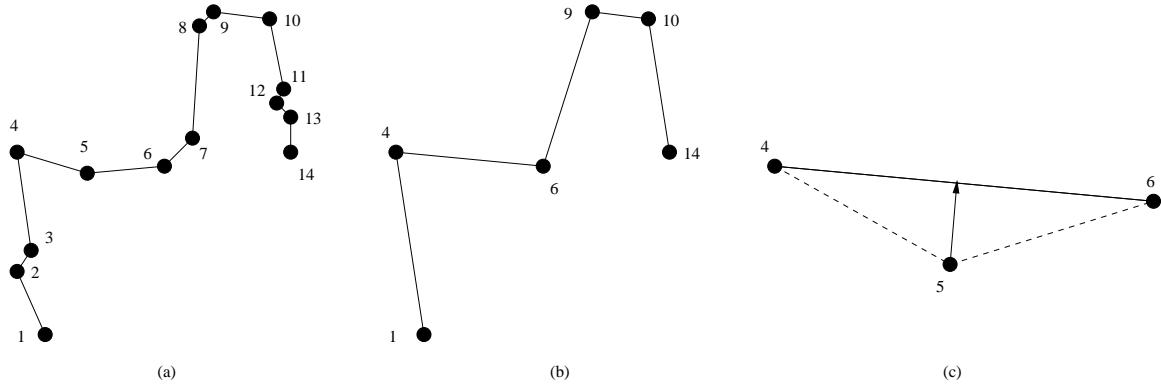
(b)

Seven points in the plane, shown on a unit grid. (a) The shortest closed tour, with length approximately 24.89. This tour is not bitonic. (b) The shortest bitonic tour for the same set of points. Its length is approximately 25.58.

Describe an  $O(n^2)$ -time algorithm for determining an optimal bitonic tour. You may assume that no two points have the same  $x$ -coordinate. [Hint: Scan left to right, maintaining optimal possibilities for the two parts of the tour.]

3. You are given a polygonal line  $\gamma$  made out of  $n$  vertices in the plane. Namely, you are given a list of  $n$  points in the plane  $p_1, \dots, p_n$ , where  $p_i = (x_i, y_i)$ . You need to display this polygonal line on the screen, however, you realize that you might be able to draw a polygonal line with considerably less vertices that looks identical on the screen (because of the limited resolution of the screen). It is crucial for you to minimize the number of vertices of the polygonal line. (Because, for example, your display is a remote Java applet running on the user computer, and for each vertex of the polygon you decide to draw, you need to send the coordinates of the points through the network which takes a *long long long time*. So the fewer vertices you send, the snappier your applet would be.)

So, given such a polygonal line  $\gamma$ , and a parameter  $k$ , you would like to select  $k$  vertices of  $\gamma$  that yield the “best” polygonal line that looks like  $\gamma$ .



Namely, you need to build a new polygonal line  $\gamma'$  and minimize the difference between the two polygonal-lines. The polygonal line  $\gamma'$  is built by selecting  $k$  vertices  $\{p_{i_1}, p_{i_2}, \dots, p_{i_k}\}$  from  $\gamma$ . It is required that  $i_1 = 1$ ,  $i_k = n$ , and  $i_j < i_{j+1}$  for  $j = 1, 2, \dots, k - 1$ .

We define the error between  $\gamma$  and  $\gamma'$  by how far from  $\gamma'$  are the vertices of  $\gamma$ . More formally, The difference between the two polygonal lines is

$$\text{error}(\gamma, \gamma') = \sum_{j=1}^{k-1} \sum_{m=i_j+1}^{i_{j+1}-1} \text{dist}(p_m, p_{i_j}p_{i_{j+1}}).$$

Namely, for every vertex not in the simplification, its associated error, is the distance to the corresponding simplified segment (see (c) in above figure). The overall error is the sum over all vertices.

You can assume that you are provided with a subroutine that can calculate  $\text{dist}(u, vw)$  in constant time, where  $\text{dist}(u, vw)$  is the distance between the point  $u$  and the segment  $vw$ .

Give an  $O(n^3)$  time algorithm to find the  $\gamma'$  that minimizes  $\text{error}(\gamma, \gamma')$ .

# CS 373: Combinatorial Algorithms, Fall 2002

Homework 2 (due Thursday, September 26, 2002 at 11:59:59 p.m.)

Name:		
Net ID:	Alias:	U G

Name:		
Net ID:	Alias:	U G

Name:		
Net ID:	Alias:	U G

---

Homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since graduate students are required to solve problems that are worth extra credit for other students, **Grad students may not be on the same team as undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate or 1-unit grad student by circling U or G, respectively. Staple this sheet to the top of your homework. **NOTE: You must use different sheet(s) of paper for each problem assigned.**

---

## Required Problems

1. For each of the following problems, the input is a set of  $n$  nuts and  $n$  bolts. For each bolt, there is exactly one nut of the same size. Direct comparisons between nuts or between bolts are not allowed, but you can compare a nut and a bolt in constant time.
  - (a) Describe and analyze a deterministic algorithm to find the largest bolt. *Exactly* how many comparisons does your algorithm perform in the worst case? [Hint: This is very easy.]
  - (b) Describe and analyze a randomized algorithm to find the largest bolt. What is the *exact* expected number of comparisons performed by your algorithm?
  - (c) Describe and analyze an algorithm to find the largest and smallest bolts. Your algorithm can be either deterministic or randomized. What is the *exact* worst-case expected number of comparisons performed by your algorithm? [Hint: Running part (a) twice is definitely not the most efficient algorithm.]

In each case, to receive **full** credit, you need to describe the most efficient algorithm possible.

2. Consider the following algorithm:

```

SLOWSHUFFLE( $A[1..n]$ ) :
  for  $i \leftarrow 1$  to  $n$ 
     $B[i] \leftarrow \text{Null}$ 
  for  $i \leftarrow 1$  to  $n$ 
    index  $\leftarrow \text{Random}(1,n)$ 
    while  $B[\text{index}] \neq \text{Null}$ 
      index  $\leftarrow \text{Random}(1,n)$ 
     $B[\text{index}] \leftarrow A[i]$ 
  for  $i \leftarrow 1$  to  $n$ 
     $A[i] \leftarrow B[i]$ 

```

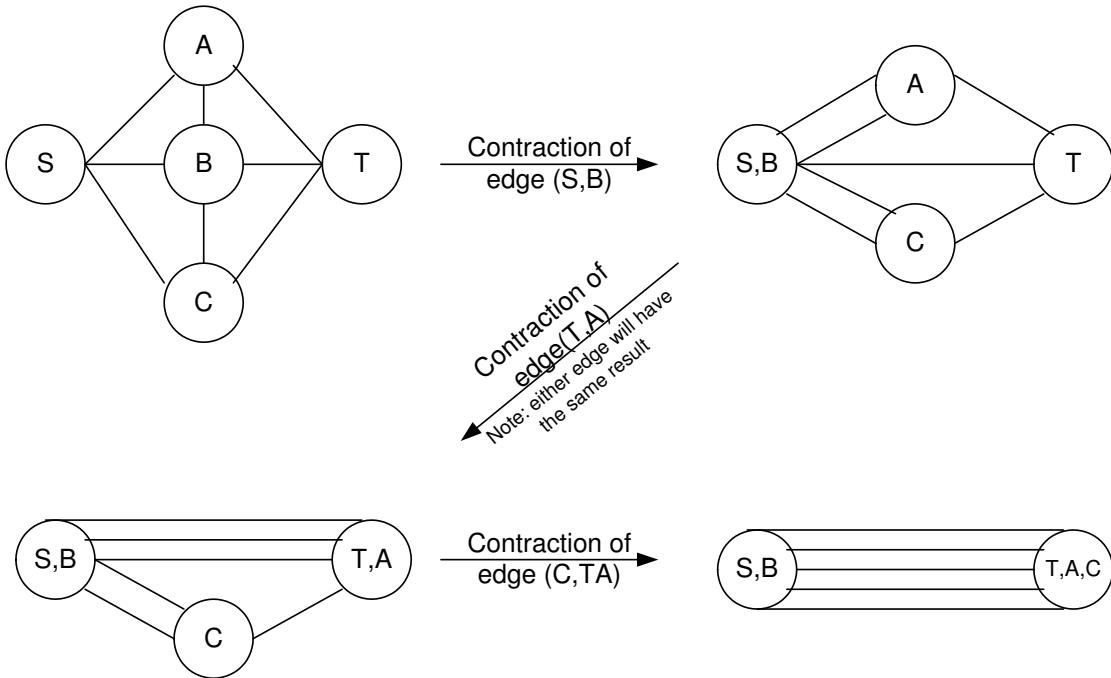
Suppose that  $\text{Random}(i,j)$  will return a random number between  $i$  and  $j$  inclusive in constant time. SLOWSHUFFLE will shuffle the input array into a random order such that every permutation is equally likely.

- (a) What is the expected running time of the above algorithm. Justify your answer and give a tight asymptotic bound.
- (b) Describe an algorithm that randomly shuffles an  $n$ -element array, so that every permutation is *equally* likely, in  $O(n)$  time.

3. Suppose we are given an undirected graph  $G = (V, E)$  together with two distinguished vertices  $s$  and  $t$ . An **s-t min-cut** is a set of edges that once removed from the graph, will disconnect  $s$  from  $t$ . We want to find such a set with the minimum cardinality (The smallest number of edges). In other words, we want to find the smallest set of edges that will separate  $s$  and  $t$

To do this we repeat the following step  $|V| - 2$  times: Uniformly at random, pick an edge from the set  $E$  which contains all edges in the graph excluding those that directly connects vertices  $s$  and  $t$ . Merge the two vertices that are connected by this randomly selected edge. If as a result there are several edges between some pair of vertices, retain them all. Edges that are between the two merged vertices are removed so that there are never any self-loops. We refer to this process of merging the two end-points of an edge into a single vertex as the *contraction* of that edge. Notice with each contraction the number of vertices of  $G$  decreases by one.

As this algorithm proceeds, the vertex  $s$  may get merged with a new vertex as the result of an edge being contracted. We call this vertex the  $s$ -vertex. Similarly, we have a  $t$ -vertex. During the contraction algorithm, we ensure that we never contract an edge between the  $s$ -vertex and the  $t$ -vertex.



- (a) Give an example of a graph in which the probability that this algorithm finds an  $s$ - $t$  min-cut is exponentially small( $O(1/a^n)$ ). Justify your answers.  
*(Hint: Think multigraphs)*
- (b) Give an example of a graph such that there are  $O(2^n)$  number of  $s$ - $t$  min-cuts. Justify your answers.
4. Describe a modification of treaps that supports the following operations, each in  $O(\log n)$  expected time:
- **INSERT( $x$ ):** Insert a new element  $x$  into the data structure.
  - **DELETE( $x$ ):** Delete an element  $x$  from the data structure.
  - **COMPUTERANK( $x$ ):** Return the number of elements in the data structure less than or equal to  $x$ .
  - **FINDBYRANK( $r$ ):** Return the  $k$ th smallest element in the data structure.

Describe and analyze the algorithms that implement each of these operations. *[Hint: Don't reinvent the wheel!]*

5. A *meldable priority queue* stores a set of keys from some totally ordered universe (such as the integers) and supports the following operations:

- **MAKEQUEUE**: Return a new priority queue storing the empty set.
- **FINDMIN( $Q$ )**: Return the smallest element stored in  $Q$  (if any).
- **DELETEMIN( $Q$ )**: Delete the smallest element stored in  $Q$  (if any).
- **INSERT( $Q, x$ )**: Insert element  $x$  into  $Q$ .
- **MELD( $Q_1, Q_2$ )**: Return a new priority queue containing all the elements stored in  $Q_1$  and  $Q_2$ . The component priority queues are destroyed.
- **DECREASEKEY( $Q, x, y$ )**: Replace an element  $x$  of  $Q$  with a smaller key  $y$ . (If  $y > x$ , the operation fails.) The input is a pointer directly to the node in  $Q$  storing  $x$ .
- **DELETE( $Q, x$ )**: Delete an element  $x \in Q$ . The input is a pointer directly to the node in  $Q$  storing  $x$ .

A simple way to implement this data structure is to use a heap-ordered binary tree, where each node stores an element, a pointer to its left child, a pointer to its right child, and a pointer to its parent. **MELD( $Q_1, Q_2$ )** can be implemented with the following randomized algorithm.

- If either one of the queues is empty, return the other one.
  - If the root of  $Q_1$  is smaller than the root of  $Q_2$ , then recursively MELD  $Q_2$  with either **right( $Q_1$ )** or **left( $Q_1$ )**, each with probability  $1/2$ .
  - Similarly, if the root of  $Q_2$  is smaller than the root of  $Q_1$ , then recursively MELD  $Q_1$  with a randomly chosen child of  $Q_2$ .
- (a) Prove that for *any* heap-ordered trees  $Q_1$  and  $Q_2$ , the expected running time of **MELD( $Q_1, Q_2$ )** is  $O(\log n)$ , where  $n = |Q_1| + |Q_2|$ . [Hint: How long is a random path in an  $n$ -node binary tree, if each left/right choice is made with equal probability?] For extra credit, prove that the running time is  $O(\log n)$  with high probability.
- (b) Show that each of the operations **DELETEMIN**, **INSERT**, **DECREASEKEY**, and **DELETE** can be implemented with one call to **MELD** and  $O(1)$  additional time. (This implies that every operation takes  $O(\log n)$  with high probability.)
6. [This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]

The following randomized algorithm selects the  $r$ th smallest element in an unsorted array  $A[1,..,n]$ . For example, to find the smallest element, you would call **RANDOMSELECT( $A, 1$ )**; to find the median element, you would call **RANDOMSELECT( $A, \lfloor n/2 \rfloor$ )**. Recall from lecture that **PARTITION** splits the array into three parts by comparing the pivot element  $A[p]$  to every other element of the array, using  $n - 1$  comparisons altogether, and returns the new index of the pivot element.

```
RANDOMSELECT( $A[1..n], r$ ) :  
     $p \leftarrow \text{RANDOM}(1, n)$   
     $k \leftarrow \text{PARTITION}(A[1..n], p)$   
    if  $r < k$   
        return RANDOMSELECT( $A[1..k - 1], r$ )  
    else if  $r > k$   
        return RANDOMSELECT( $A[k + 1..n], r - k$ )  
    else  
        return  $A[k]$ 
```

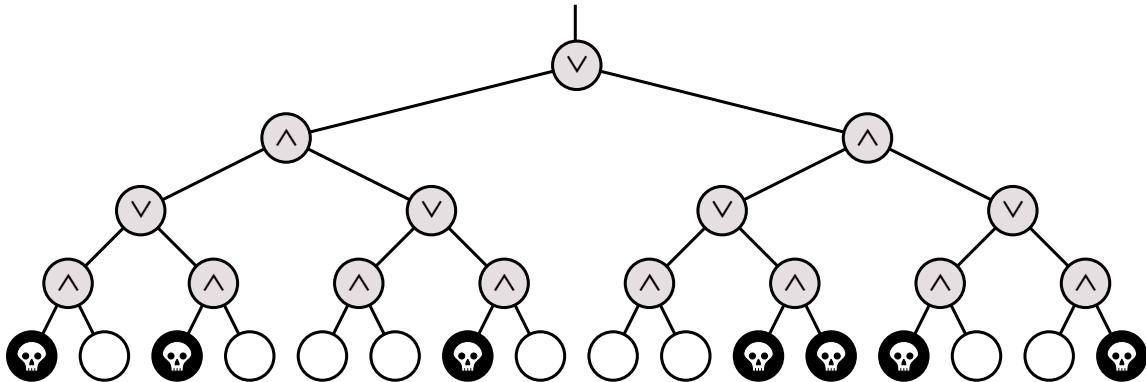
- (a) State a recurrence for the expected running time of RANDOMSELECT, as a function of both  $n$  and  $r$ .
- (b) What is the *exact* probability that RANDOMSELECT compares the  $i$ th smallest and  $j$ th smallest elements in the input array? The correct answer is a simple function of  $i$ ,  $j$ , and  $r$ . [Hint: Check your answer by trying a few small examples.]
- (c) Show that for any  $n$  and  $r$ , the expected running time of RANDOMSELECT is  $\Theta(n)$ . You can use either the recurrence from part (a) or the probabilities from part (b). For extra credit, find the *exact* expected number of comparisons, as a function of  $n$  and  $r$ .
- (d) What is the expected number of times that RANDOMSELECT calls itself recursively?

## Practice Problems

1. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with  $4^n$  leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after  $2n$  moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.

You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are AND gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

- (a) Describe and analyze a deterministic algorithm to determine whether or not you can win. [Hint: This is easy!]
- (b) Unfortunately, Death won't let you even look at every node in the tree. Describe a randomized algorithm that determines whether you can win in  $\Theta(3^n)$  expected time. [Hint: Consider the case  $n = 1$ .]



2. What is the *exact* number of nodes in a skip list storing  $n$  keys, *not* counting the sentinel nodes at the beginning and end of each level? Justify your answer.
3. Suppose we are given two sorted arrays  $A[1..n]$  and  $B[1..n]$  and an integer  $k$ . Describe an algorithm to find the  $k$ th smallest element in the union of  $A$  and  $B$ . (For example, if  $k = 1$ , your algorithm should return the smallest element of  $A \cup B$ ; if  $k = n$ , our algorithm should return the median of  $A \cup B$ .) You can assume that the arrays contain no duplicates. Your algorithm should be able to run in  $\Theta(\log n)$  time. [Hint: First try to solve the special case  $k = n$ .]

**CS 373: Combinatorial Algorithms, Fall 2002**  
**Homework 3, due October 17, 2002 at 23:59:59**

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Undergrads	
Grads	

This homework is to be submitted in groups of up to three people. Graduate and undergraduate students are *not* allowed to work in the same group. Please indicate above whether you are undergraduate or graduate students. Only *one* submission per group will be accepted.

## Required Problems

1.
    - (a) Prove that only one subtree gets rebalanced in a scapegoat tree insertion.
    - (b) Prove that  $I(v) = 0$  in every node of a perfectly balanced tree. (Recall that  $I(v) = \max\{0, |T| - |s| - 1\}$ , where  $T$  is the child of greater height and  $s$  the child of lesser height, and  $|v|$  is the number of nodes in subtree  $v$ . A perfectly balanced tree has two perfectly balanced subtrees, each with as close to half the nodes as possible.)
  - \*(c) Show that you can rebuild a fully balanced binary tree from an unbalanced tree in  $O(n)$  time using only  $O(\log n)$  additional memory.
  2. Suppose we can insert or delete an element into a hash table in constant time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:
    - After an insertion, if the table is more than  $3/4$  full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
    - After a deletion, if the table is less than  $1/4$  full, we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do *not* use the potential method (it makes it much more difficult).

3. A stack is a FILO/LIFO data structure that represents a stack of objects; access is only allowed at the top of the stack. In particular, a stack implements two operations:

  - $\text{PUSH}(x)$ : adds  $x$  to the top of the stack.

- `POP`: removes the top element and returns it.

A queue is a FIFO/LIFO data structure that represents a row of objects; elements are added to the front and removed from the back. In particular, a queue implements two operations:

- `ENQUEUE( $x$ )`: adds  $x$  to the front of the queue.
- `DEQUEUE`: removes the element at the back of the queue and returns it.

Using two stacks and no more than  $O(1)$  additional space, show how to simulate a queue for which the operations `ENQUEUE` and `DEQUEUE` run in constant amortized time. You should treat each stack as a black box (i.e., you may call `PUSH` and `POP`, but you do not have access to the underlying stack implementation). Note that each `PUSH` and `POP` performed by a stack takes  $O(1)$  time.

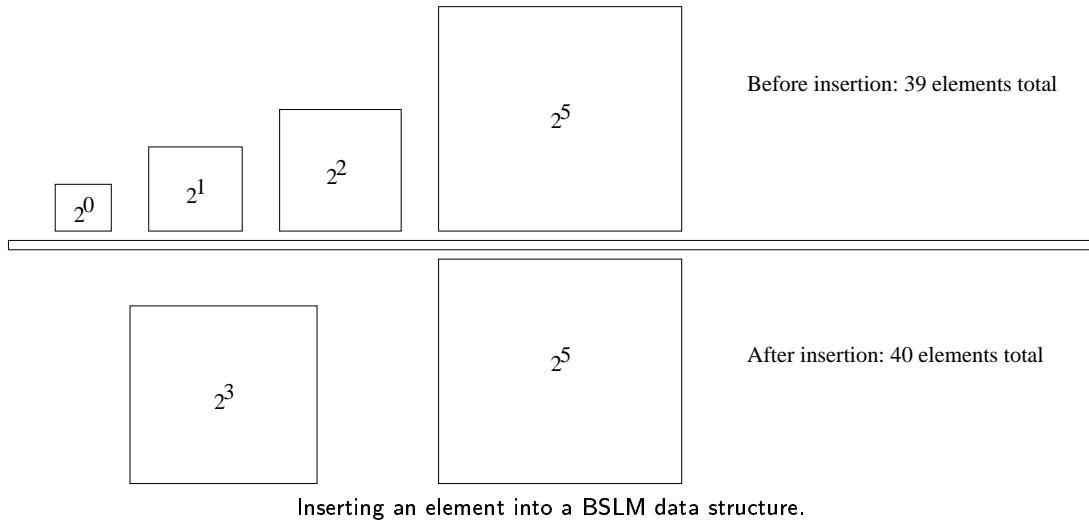
4. A data structure is *insertion-disabled* if there is no way to add elements to it. For the purposes of this problem, further assume that an insertion-disabled data structure implements the following operations with the given running times:

- `INITIALIZE( $S$ )`: Return an insertion-disabled data structure that contains the elements of  $S$ . Running time:  $O(n \log n)$ .
- `SEARCH( $D, x$ )`: Return TRUE if  $x$  is in  $D$ ; return FALSE if not. Running time:  $O(\log n)$ .
- `RETURNALL( $D, x$ )`: Return an unordered set of all elements in  $D$ . Running time:  $O(n)$ .
- `DELETE( $D, x$ )`: Remove  $x$  from  $D$  if  $x$  is in  $D$ . Running time:  $O(\log n)$ .

Using an approach known as the Bentley-Saxe Logarithmic Method (BSLM), it is possible to represent a dynamic (i.e., supports insertions) data structure with a collection of insertion-disabled data structures, where each insertion-disabled data structure stores a number of elements that is a distinct power of two. For example, to store  $39 = 2^0 + 2^1 + 2^2 + 2^5$  elements in a BSLM data structure, we use four insertion-disabled data structures with  $2^0$ ,  $2^1$ ,  $2^2$ , and  $2^5$  elements.

To find an element in a BSLM data structure, we search the collection of insertion-disabled data structures until we find (or don't find) the element.

To insert an element into a BSLM data structure, we think about adding a  $2^0$ -size insertion-disabled data structure. However, an insertion-disabled data structure with  $2^0$  elements may already exist. In this case, we can combine two  $2^0$ -size structures into a single  $2^1$ -size structure. However, there may already be a  $2^1$ -size structure, so we will need to repeat this process. In general, we do the following: Find the smallest  $i$  such that for all nonnegative  $k < i$ , there is a  $2^k$ -sized structure in our collection. Create a  $2^i$ -sized structure that contains the element to be inserted and all elements from  $2^k$ -sized data structures for all  $k < i$ . Destroy all  $2^k$ -sized data structures for  $k < i$ .



We delete elements from the BSLM data structure lazily. To delete an element, we first search the collection of insertion-disabled data structures for it. Then we call `DELETE` to remove the element from its insertion-disabled data structure. This means that a  $2^i$ -sized insertion-disabled data structure might store less than  $2^i$  elements. That's okay; we just say that it stores  $2^i$  elements and say that  $2^i$  is its *pretend size*. We keep track of a single variable, called *Waste*, which is initially 0 and is incremented by 1 on each deletion. If *Waste* exceeds three-quarters of the total pretend size of all insertion-disabled data structures in our collection (i.e., the total number of elements stored), we rebuild our collection of insertion-disabled data structures. In particular, we create a  $2^m$ -sized insertion-disabled data structure, where  $2^m$  is the smallest power that is greater than or equal to the total number of elements stored. All elements are stored in this  $2^m$ -sized insertion-disabled data structure, and all other insertion-disabled data structures in our collection are destroyed. *Waste* is reset to  $2^m - n$ , where  $n$  is the total number of elements stored in the BSLM data structure.

Your job is to prove the running times of the following three BSLM operations:

- `SEARCHBSLM( $D, x$ )`: Search for  $x$  in the collection of insertion-disabled data structures that represent the BSLM data structure  $D$ . Running time:  $O(\log^2 n)$  worst-case.
- `INSERTBSLM( $D, x$ )`: Insert  $x$  into the collection of insertion-disabled data structures that represent the BSLM data structure  $D$ , modifying the collection as necessary. Running time:  $O(\log^2 n)$  amortized.
- `DELETEBSLM( $D, x$ )`: Delete  $x$  from the collection of insertion-disabled data structures that represent the BSLM data structure  $D$ , rebuilding when there is a lot of wasted space. Running time:  $O(\log^2 n)$  amortized.

5. Except as noted, the following sub-problems refer to a Union-Find data structure that uses both path compression and union by rank.
  - (a) Prove that in a set of  $n$  elements, a sequence of  $n$  consecutive `FIND` operations takes  $O(n)$  total time.
  - (b) Show that any sequence of  $m$  `MAKESET`, `FIND`, and `UNION` operations takes only  $O(m)$  time if all of the `UNION` operations occur before any of the `FIND` operations.

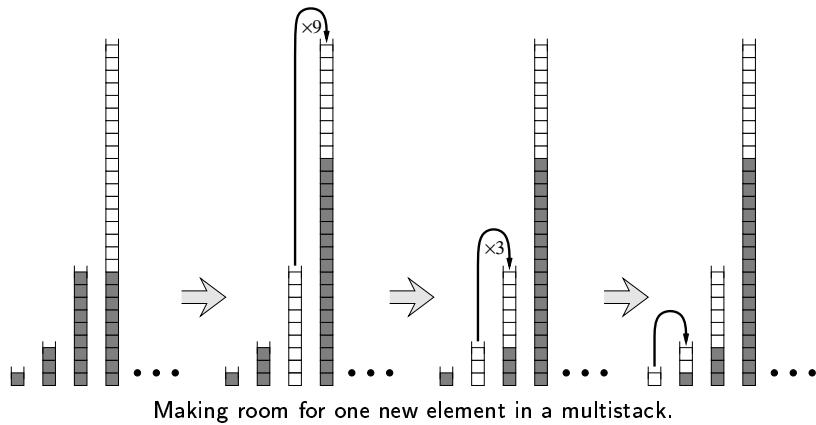
- (c) Now consider part b with a Union-Find data structure that uses path compression but does *not* use union by rank. Is  $O(m)$  time still correct? Prove your answer.
6. [This problem is required only for graduate students (including I2CS students); undergrads can submit a solution for extra credit.]

Suppose instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of ‘fits’, where the  $i$ th least significant fit indicates whether the sum includes the  $i$ th Fibonacci number  $F_i$ . For example, the fit string 101110 represents the number  $F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14$ . Describe algorithms to increment and decrement a fit string in constant amortized time. [Hint: Most numbers can be represented by more than one fit string.]

## Practice Problems

These remaining practice problems are entirely for your benefit. Don't turn in solutions—we'll just throw them out—but feel free to ask us about these questions during office hours and review sessions. Think of these as potential exam questions (hint, hint).

1. A *multistack* consists of an infinite series of stacks  $S_0, S_1, S_2, \dots$ , where the  $i$ th stack  $S_i$  can hold up to  $3^i$  elements. Whenever a user attempts to push an element onto any full stack  $S_i$ , we first move all the elements in  $S_i$  to stack  $S_{i+1}$  to make room. But if  $S_{i+1}$  is already full, we first move all its members to  $S_{i+2}$ , and so on. Moving a single element from one stack to the next takes  $O(1)$  time.



- (a) In the worst case, how long does it take to push one more element onto a multistack containing  $n$  elements?  
(b) Prove that the amortized cost of a push operation is  $O(\log n)$ , where  $n$  is the maximum number of elements in the multistack. You can use any method you like.
2. A hash table of size  $m$  is used to store  $n$  items with  $n \leq m/2$ . Open addressing is used for collision resolution.
  - Assuming uniform hashing, show that for  $i = 1, 2, \dots, n$ , the probability that the  $i^{th}$  insertion requires strictly more than  $k$  probes is at most  $2^{-k}$ .
  - Show that for  $i = 1, 2, \dots, n$ , the probability that the  $i^{th}$  insertion requires more than  $2 \lg n$  probes is at most  $1/n^2$ .

Let the random variable  $X_i$  denote the number of probes required by the  $i^{th}$  insertion. You have shown in part (b) that  $\Pr\{X_i > 2 \lg n\} \leq 1/n^2$ . Let the random variable  $X = \max_{1 \leq i \leq n} X_i$  denote the maximum number of probes required by any of the  $n$  insertions.

- Show that  $\Pr\{X > 2 \lg n\} \leq 1/n$ .
- Show that the expected length of the longest probe sequence is  $E[X] = O(\lg n)$ .

3. A sequence of  $n$  operations is performed on a data structure. The  $i$ th operation costs  $i$  if  $i$  is an exact power of 2, and 1 otherwise. That is operation  $i$  costs  $f(i)$ , where:

$$f(i) = \begin{cases} i, & i = 2^k, \\ 1, & \text{otherwise} \end{cases}$$

Determine the amortized cost per operation using the following methods of analysis:

- (a) Aggregate method
- (b) Accounting method
- \*(c) Potential method

# CS 373: Combinatorial Algorithms, Fall 2002

## Homework 4, due Thursday, October 31, 2002 at 23:59.99

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Undergrads	Grads

This homework is to be submitted in groups of up to three people. Graduate and undergraduate students are *not* allowed to work in the same group. Please indicate above whether you are undergraduate or graduate students. Only *one* submission per group will be accepted.

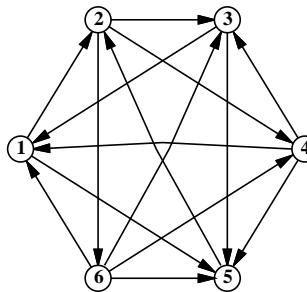
---

## Required Problems

### 1. Tournament:

A *tournament* is a directed graph with exactly one edge between every pair of vertices. (Think of the nodes as players in a round-robin tournament, where each edge points from the winner to the loser.) A *Hamiltonian path* is a sequence of directed edges, joined end to end, that visits every vertex exactly once.

Prove that every tournament contains at least one Hamiltonian path.



A six-vertex tournament containing the Hamiltonian path  $6 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1$ .

### 2. Acrophobia:

Consider a graph  $G = (V, E)$  whose nodes are cities, and whose edges are roads connecting the cities. For each edge, the weight is assigned by  $h_e$ , the maximum altitude encountered when traversing the specified road. Between two cities  $s$  and  $t$ , we are interested in those paths whose maximum altitude is as low as possible. We will call a subgraph,  $G'$ , of  $G$  an *acrophobic friendly subgraph*, if for any two nodes  $s$  and  $t$  the path of minimum altitude is always

included in the subgraph. For simplicity, assume that the maximum altitude encountered on each road is unique.

- (a) Prove that every graph of  $n$  nodes has an acrophobic friendly subgraph that has only  $n - 1$  edges.
- (b) Construct an algorithm to find an acrophobic friendly subgraph given a graph  $G = (V, E)$ .
3. Refer to the lecture notes on single-source shortest paths. The **GENERICSSSP** algorithm described in class can be implemented using a stack for the 'bag'. Prove that the resulting algorithm, given a graph with  $n$  nodes as input, could perform  $\Omega(2^n)$  relaxation steps before stopping. You need to describe, for any positive integer  $n$ , a specific weighted directed  $n$ -vertex graph that forces this exponential behavior. The easiest way to describe such a family of graphs is using an *algorithm*!

4. Neighbors:

Two spanning trees  $T$  and  $T'$  are defined as *neighbors* if  $T'$  can be obtained from  $T$  by swapping a single edge. More formally, there are two edges  $e$  and  $f$  such that  $T'$  is obtained from  $T$  by adding edge  $e$  and deleting edge  $f$ .

- (a) Let  $T$  denote the minimum cost spanning tree and suppose that we want to find the second cheapest tree  $T'$  among all trees. Assuming unique costs for all edges, prove that  $T$  and  $T'$  are neighbors.
- (b) Given a graph  $G = (V, E)$ , construct an algorithm to find the second cheapest tree,  $T'$ .
- (c) Consider a graph,  $H$ , whose vertices are the spanning trees of the graph  $G$ . Two vertices are connected by an edge if and only if they are neighbors as previously defined. Prove that for any graph  $G$  this new graph  $H$  is connected.

5. Network Throughput:

Suppose you are given a graph of a (tremendously simplified) computer network  $G = (V, E)$  such that a weight,  $b_e$ , is assigned to each edge representing the communication bandwidth of the specified channel in Kb/s and each node is assigned a value,  $l_v$ , representing the server latency measured in seconds/packet. Given a fixed packet size, and assuming all edge bandwidth values are a multiple of the packet size, your job is to build a system to decide which paths to route traffic between specified servers.

More formally, a person wants to route traffic from server  $s$  to server  $t$  along the path of maximum throughput. Give an algorithm that will allow a network design engineer to choose an optimal path by which to route data traffic.

## 6. All-Pairs-Shortest-Path:

*[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

Given an undirected, unweighted, connected graph  $G = (V, E)$ , we wish to solve the distance version of the all-pairs-shortest-path problem. The algorithm APD takes the  $n \times n$  0-1 adjacency matrix  $A$  and returns an  $n \times n$  matrix  $D$  such that  $d_{ij}$  represents the shortest path between vertices  $i$  and  $j$ .

```

APD( $A$ )
 $Z \leftarrow A \cdot A$ 
let  $B$  be an  $n \times n$  matrix, where  $b_{ij} = 1$  iff  $i \neq j$  and ( $a_{ij} = 1$  or  $z_{ij} > 0$ )
if  $b_{ij} = 1$  for all  $i \neq j$ 
    return  $D \leftarrow 2B - A$ 
 $T \leftarrow APD(B)$ 
 $X \leftarrow T \cdot A$ 
foreach  $x_{ij}$ 
    if  $x_{ij} \geq t_{ij} \cdot \text{degree}(j)$ 
         $d_{ij} \leftarrow 2t_{ij}$ 
    else
         $d_{ij} \leftarrow 2t_{ij} - 1$ 
return  $D$ 

```

- (a) In the APD algorithm above, what do the matrices  $Z$ ,  $B$ ,  $T$ , and  $X$  represent? Justify your answers.
- (b) Prove that the APD algorithm correctly computes the matrix of shortest path distances. In other words, prove that in the output matrix  $D$ , each entry  $d_{ij}$  represents the shortest path distance between node  $i$  and node  $j$ .
- (c) Suppose we can multiply two  $n \times n$  matrices in  $M(n)$  time, where  $M(n) = \Omega(n^2)$ .<sup>1</sup> Prove that APD runs in  $O(M(n) \log n)$  time.

---

<sup>1</sup>The matrix multiplication algorithm you already know runs in  $O(n^3)$  time, but this is not the fastest known. The current record is  $M(n) = O(n^{2.376})$ , due to Don Coppersmith and Shmuel Winograd. Determining the smallest possible value of  $M(n)$  is a long-standing open problem.

## Practice Problems

### 1. Makefiles:

In order to facilitate recompiling programs from multiple source files when only a small number of files have been updated, there is a UNIX utility called ‘make’ that only recompiles those files that were changed after the most recent compilation, *and* any intermediate files in the compilation that depend on those that were changed. A Makefile is typically composed of a list of source files that must be compiled. Each of these source files is dependent on some of the other files which are listed. Thus a source file must be recompiled if a file on which it depends is changed.

Assuming you have a list of which files have been recently changed, as well as a list for each source file of the files on which it depends, design an algorithm to recompile only those necessary. DO NOT worry about the details of parsing a Makefile.

### 2. The incidence matrix of an undirected graph $G = (V, E)$ is a $|V| \times |E|$ matrix $B = (b_{ij})$ such that

$$b_{ij} = \begin{cases} 1 & \text{if vertex } v_i \text{ is an endpoint of edge } e_j \\ 0 & \text{otherwise} \end{cases}$$

- (a) Describe what all the entries of the matrix product  $BB^T$  represent ( $B^T$  is the matrix transpose). Justify.
- (b) Describe what all the entries of the matrix product  $B^TB$  represent. Justify.
- ★(c) Let  $C = BB^T - 2A$ . Let  $C'$  be  $C$  with the first row and column removed. Show that  $\det C'$  is the number of spanning trees. ( $A$  is the adjacency matrix of  $G$ , with zeroes on the diagonal).

### 3. Reliable Network:

Suppose you are given a graph of a computer network  $G = (V, E)$  and a function  $r(u, v)$  that gives a reliability value for every edge  $(u, v) \in E$  such that  $0 \leq r(u, v) \leq 1$ . The reliability value gives the probability that the network connection corresponding to that edge will *not* fail. Describe and analyze an algorithm to find the most reliable path from a given source vertex  $s$  to a given target vertex  $t$ .

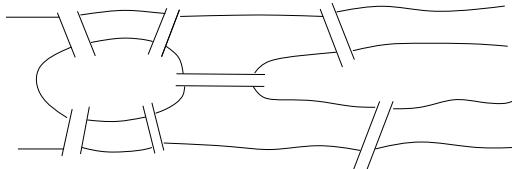
### 4. Aerophobia:

After graduating you find a job with Aerophobes-R'-Us, the leading traveling agency for aerophobic people. Your job is to build a system to help customers plan airplane trips from one city to another. All of your customers are afraid of flying so the trip should be as short as possible.

In other words, a person wants to fly from city  $A$  to city  $B$  in the shortest possible time. S/he turns to the traveling agent who knows all the departure and arrival times of all the flights on the planet. Give an algorithm that will allow the agent to choose an optimal route to minimize the total time in transit. Hint: rather than modify Dijkstra’s algorithm, modify the data. The total transit time is from departure to arrival at the destination, so it will include layover time (time waiting for a connecting flight).

## 5. The Seven Bridges of Königsberg:

During the eighteenth century the city of Königsberg in East Prussia was divided into four sections by the Pregel river. Seven bridges connected these regions, as shown below. It was said that residents spent their Sunday walks trying to find a way to walk about the city so as to cross each bridge exactly once and then return to their starting point.



- (a) Show how the residents of the city could accomplish such a walk or prove no such walk exists.
- (b) Given any undirected graph  $G = (V, E)$ , give an algorithm that finds a cycle in the graph that visits every edge exactly once, or says that it can't be done.
- 6. Given an undirected graph  $G = (V, E)$  with costs  $c_e \geq 0$  on the edges  $e \in E$  give an  $O(|E|)$  time algorithm that tests if there is a minimum cost spanning tree that contains the edge  $e$ .

## 7. Combining Boruvka and Prim:

Give an algorithm that finds the MST of a graph  $G$  in  $O(m \log \log n)$  time by combining Boruvka's and Prim's algorithm.

## 8. Minimum Spanning Tree changes:

Suppose you have a graph  $G$  and an MST of that graph (i.e. the MST has already been constructed).

- (a) Give an algorithm to update the MST when an edge is added to  $G$ .
- (b) Give an algorithm to update the MST when an edge is deleted from  $G$ .
- (c) Give an algorithm to update the MST when a vertex (and possibly edges to it) is added to  $G$ .

## 9. Nesting Envelopes

You are given an unlimited number of each of  $n$  different types of envelopes. The dimensions of envelope type  $i$  are  $x_i \times y_i$ . In nesting envelopes inside one another, you can place envelope  $A$  inside envelope  $B$  if and only if the dimensions  $A$  are *strictly smaller* than the dimensions of  $B$ . Design and analyze an algorithm to determine the largest number of envelopes that can be nested inside one another.

10.  $o(V^2)$  Adjacency Matrix Algorithms

- (a) Give an  $O(V)$  algorithm to decide whether a directed graph contains a *sink* in an adjacency matrix representation. A sink is a vertex with in-degree  $V - 1$ .

- (b) An undirected graph is a scorpion if it has a vertex of degree 1 (the sting) connected to a vertex of degree two (the tail) connected to a vertex of degree  $V - 2$  (the body) connected to the other  $V - 3$  vertices (the feet). Some of the feet may be connected to other feet.

Design an algorithm that decides whether a given adjacency matrix represents a scorpion by examining only  $O(V)$  of the entries.

- (c) Show that it is impossible to decide whether  $G$  has at least one edge in  $O(V)$  time.

11. Shortest Cycle:

Given an **undirected** graph  $G = (V, E)$ , and a weight function  $f : E \rightarrow \mathbf{R}$  on the **edges**, give an algorithm that finds (in time polynomial in  $V$  and  $E$ ) a cycle of smallest weight in  $G$ .

12. Longest Simple Path:

Let graph  $G = (V, E)$ ,  $|V| = n$ . A *simple path* of  $G$ , is a path that does not contain the same vertex twice. Use dynamic programming to design an algorithm (not polynomial time) to find a simple path of maximum length in  $G$ . Hint: It can be done in  $O(n^c 2^n)$  time, for some constant  $c$ .

13. Minimum Spanning Tree:

Suppose all edge weights in a graph  $G$  are equal. Give an algorithm to compute an MST.

14. Transitive reduction:

Give an algorithm to construct a *transitive reduction* of a directed graph  $G$ , i.e. a graph  $G^{TR}$  with the fewest edges (but with the same vertices) such that there is a path from  $a$  to  $b$  in  $G$  iff there is also such a path in  $G^{TR}$ .

15. (a) What is  $5^{2^{29}5^0 + 23^41 + 17^32 + 11^23 + 5^14} \pmod{6}$ ?

- (b) What is the capital of Nebraska? Hint: It is not Omaha. It is named after a famous president of the United States that was not George Washington. The distance from the Earth to the Moon averages roughly 384,000 km.

# CS 373: Combinatorial Algorithms, Fall 2002

<http://www-courses.cs.uiuc.edu/~cs373>

## Homework 5 (due Thur. Nov. 21, 2002 at 11:59 pm)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

---

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate,  $\frac{3}{4}$ -unit grad student, or 1-unit grad student by circling U,  $\frac{3}{4}$ , or 1, respectively. Staple this sheet to the top of your homework.

---

## Required Problems

1. (10 points) Given two arrays,  $A[1..n]$  and  $B[1..m]$  we want to determine whether there is an  $i \geq 0$  such that  $B[1] = A[i + 1], B[2] = A[i + 2], \dots, B[m] = A[i + m]$ . In other words, we want to determine if  $B$  is a substring of  $A$ . Show how to solve this problem in  $O(n \log n)$  time with high probability.
2. (5 points) Let  $a, b, c \in \mathbb{Z}^+$ .
  - (a) Prove that  $\gcd(a, b) \cdot \text{lcm}(a, b) = ab$ .
  - (b) Prove  $\text{lcm}(a, b, c) = \text{lcm}(\text{lcm}(a, b), c)$ .
  - (c) Prove  $\gcd(a, b, c)\text{lcm}(ab, ac, bc) = abc$ .
3. (5 points) Describe an efficient algorithm to compute multiplicative inverses modulo a prime  $p$ . Does your algorithm work if the modulus is composite?
4. (10 points) Describe an efficient algorithm to compute  $F_n \bmod m$ , given integers  $n$  and  $m$  as input.

5. (10 points) Let  $n$  have the prime factorization  $p_1^{k_1} p_2^{k_2} \cdots p_t^{k_t}$ , where the primes  $p_i$  are distinct and have exponents  $k_i > 0$ . Prove that

$$\phi(n) = \prod_{i=1}^t p_i^{k_i-1} (p_i - 1).$$

Conclude that  $\phi(n)$  can be computed in polynomial time given the prime factorization of  $n$ .

6. (10 points) Suppose we want to compute the Fast Fourier Transform of an integer vector  $P[0..n-1]$ . We could choose an integer  $m$  larger than any coefficient  $P[i]$ , and then perform all arithmetic modulo  $m$  (or more formally, in the ring  $\mathbb{Z}_m$ ). In order to make the FFT algorithm work, we need to find an integer that functions as a "primitive  $n$ th root of unity modulo  $m$ ".

For this problem, let's assume that  $m = 2^{n/2} + 1$ , where as usual  $n$  is a power of two.

- (a) Prove that  $2^n \equiv 1 \pmod{m}$ .
  - (b) Prove that  $\sum_{k=0}^{n-1} 2^k \equiv 0 \pmod{m}$ . These two conditions imply that 2 is a primitive  $n$ th root of unity in  $\mathbb{Z}_m$ .
  - (c) Given (a), (b), and (c), *briefly* argue that the "FFT modulo  $m$ " of  $P$  is well-defined and can be computed in  $O(n \log n)$  arithmetic operations.
  - (d) Prove that  $n$  has a multiplicative inverse in  $\mathbb{Z}_m$ . [Hint:  $n$  is a power of 2, and  $m$  is odd.] We need this property to implement the inverse FFT modulo  $m$ .
  - (e) What is the FFT of the sequence  $[3, 1, 3, 3, 7, 3, 7, 3]$  modulo 17?
7. (10 points) [*This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.*]
- (a) Prove that for any integer  $n > 1$ , if the  $n$ -th Fibonacci number  $F_n$  is prime then either  $n$  is prime or  $n = 4$ .
  - (b) Prove that if  $a$  divides  $b$ , then  $F_a$  divides  $F_b$ .
  - (c) Prove that  $\gcd(F_a, F_b) = F_{\gcd(a,b)}$ . This immediately implies parts (a) and (b), so if you solve this part, you don't have to solve the other two.

## Practice Problems

1. Let  $a, b, n \in \mathbb{Z} \setminus \{0\}$ . Assume  $\gcd(a, b) | n$ . Prove the entire set of solutions to the equation

$$n = ax + by$$

is given by:

$$\Gamma = \left\{ x_0 + \frac{tb}{\gcd(a, b)}, y_0 - \frac{ta}{\gcd(a, b)} : t \in \mathbb{Z} \right\}.$$

2. Show that in the RSA cryptosystem the decryption exponent  $d$  can be chosen such that  $de \equiv 1 \pmod{\text{lcm}(p-1, q-1)}$ .

3. Let  $(n, e)$  be a public RSA key. For a plaintext  $m \in \{0, 1, \dots, n-1\}$ , let  $c = m^e \pmod{n}$  be the corresponding ciphertext. Prove that there is a positive integer  $k$  such that

$$m^{e^k} \equiv m \pmod{n}.$$

For such an integer  $k$ , prove that

$$c^{e^{k-1}} \equiv m \pmod{n}.$$

Is this dangerous for RSA?

4. Prove that if Alice's RSA public exponent  $e$  is 3 and an adversary obtains Alice's secret exponent  $d$ , then the adversary can factor Alice's modulus  $n$  in time polynomial in the number of bits in  $n$ .

# CS 373: Combinatorial Algorithms, Fall 2002

<http://www-courses.cs.uiuc.edu/~cs373>

## Homework 6 (Do not hand in!)

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

Name:		
Net ID:	Alias:	U $\frac{3}{4}$ 1

---

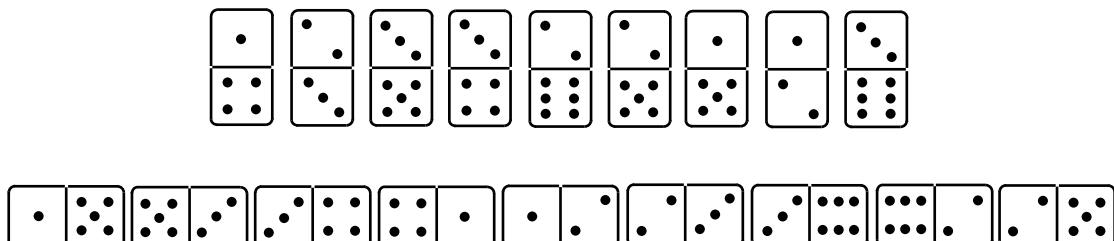
Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U,  $\frac{3}{4}$ , or 1, respectively. Staple this sheet to the top of your homework.

---

## Required Problems

- (10 points) Prove that SAT is still a NP-complete problem even under the following constraints: each variable must show up once as a positive literal and once or twice as a negative literal in the whole expression. For instance,  $(A \vee \bar{B}) \wedge (\bar{A} \vee C \vee D) \wedge (\bar{A} \vee B \vee \bar{C} \vee \bar{D})$  satisfies the constraints, while  $(A \vee \bar{B}) \wedge (\bar{A} \vee C \vee D) \wedge (A \vee B \vee \bar{C} \vee \bar{D})$  does not, because positive literal A appears twice.
- (10 points) A domino is  $2 \times 1$  rectangle divided into two squares, with a certain number of pips(dots) in each square. In most domino games, the players lay down dominos at either end of a single chain. Adjacent dominos in the chain must have matching numbers. (See the figure below.)

Describe and analyze an efficient algorithm, or prove that it is NP-complete, to determine whether a given set of n dominos can be lined up in a single chain. For example, for the sets of dominos shown below, the correct output is TRUE.



Top: A set of nine dominos  
Bottom: The entire set lined up in a single chain

3. (10 points) Prove that the following 2 problems are NP-complete. Given an undirected Graph  $G = (V, E)$ , a subset of vertices  $V' \subseteq V$ , and a positive integer  $k$ :

- (a) determine whether there is a spanning tree  $T$  of  $G$  whose leaves are the same as  $V'$ .
- (b) determine whether there is a spanning tree  $T$  of  $G$  whose degree of vertices are all less than  $k$ .

4. (10 points) An optimized version of Knapsack problem is defined as follows. Given a finite set of elements  $U$  where each element of the set  $u \in U$  has its own size  $s(u) > 0$  and the value  $v(u) > 0$ , maximize  $A(U') = \sum_{u \in U'} v(u)$  under the condition  $\sum_{u \in U'} s(u) \leq B$  and  $U' \subseteq U$ . This problem is NP-hard. Consider the following polynomial time approximation algorithm. Determine the worst case approximation ratio  $R(U) = \max_U Opt(U)/Approx(U)$  and prove it.

```
APPROXIMATION ALGORITHM:
A1 ← Greedy()
A2 ← SingleElement()
return max(A1, A2)
```

**GREEDY:**

Put all the elements  $u \in U$  into an array  $A[i]$   
Sort  $A[i]$  by  $v(u)/s(u)$  in a decreasing order  
 $S \leftarrow 0$   
 $V \leftarrow 0$   
for  $i \leftarrow 0$  to NumOfElements  
if  $(S + s(u[i]) > B)$   
break  
 $S \leftarrow S + s(u[i])$   
 $V \leftarrow V + v(u[i])$   
return  $V$

**SINGLE ELEMENT:**

Put all the elements  $u \in U$  into an array  $A[i]$   
 $V \leftarrow 0$   
for  $i \leftarrow 0$  to NumOfElements  
if  $(s(u[i]) \leq B \& V < v(u[i]))$   
 $V \leftarrow v(u[i])$   
return  $V$

5. (10 points) The recursion fairy's distant cousin, the reduction genie, shows up one day with a magical gift for you: a box that determines in constant time whether or not a graph is 3-colorable.(A graph is 3-colorable if you can color each of the vertices red, green, or blue, so that every edge has do different colors.) The magic box does not tell you how to color the graph, just wheter or not it can be done. Devise and analyze an algorithm to 3-color any graph in **polynomial time** using the magic box.

6. (10 points) The following is an NP-hard version of PARTITION problem.

**PARTITION(NP-HARD):**

Given a set of  $n$  positive integers  $S = \{a_i | i = 0 \dots n - 1\}$ ,

$$\text{minimize } \max \left( \sum_{a_i \in T} a_i, \sum_{a_i \in S-T} a_i \right)$$

where  $T$  is a subset of  $S$ .

A polynomial time approximation algorithm is given in what follows. Determine the worst case approximation ratio  $\min_S Approx(S)/Opt(S)$  and prove it.

**APPROXIMATION ALGORITHM:**

```

Sort S in an increasing order
s1 ← 0
s2 ← 0
for i ← 0 to n
    if s1 ≤ s2
        s1 ← s1 + ai
    else
        s2 ← s2 + ai
result ← max(s1, s2)

```

**Practice Problems**

1. Construct a linear time algorithm for 2 SAT problem.
  
  
  
  
  
2. Assume that  $P \neq NP$ . Prove that there is no polynomial time approximation algorithm for an optimized version of Knapsack problem, which outputs  $A(I)$  s.t.  $|Opt(I) - A(I)| \leq K$  for any instance  $I$ , where  $K$  is a constant.
  
  
  
  
  
3. Your friend Toidi is planning to hold a party for the coming Christmas. He wants to take a picture of all the participants including himself, but he is quite **shy** and thus cannot take a picture of a person whom he does not know very well. Since he has only **shy** friends, every participant coming to the party is also **shy**. After a long struggle of thought he came up with a seemingly good idea:
  - At the beginning, he has a camera.
  - A person, holding a camera, is able to take a picture of another participant whom the person knows very well, and pass a camera to that participant.
  - Since he does not want to waste films, everyone has to be taken a picture exactly once.

Although there can be some people whom he does not know very well, he knows completely who knows whom well. Therefore, in theory, given a list of all the participants, he can determine if it is possible to take all the pictures using this idea. Since it takes only linear time to take all the pictures if he is brave enough (say “Say cheese!”  $N$  times, where  $N$  is the number of people), as a student taking CS373, you are highly expected to give him an advice:

- show him an efficient algorithm to determine if it is possible to take pictures of all the participants using his idea, given a list of people coming to the party.
- or prove that his idea is essentially facing a NP-complete problem, make him give up his idea, and give him an efficient algorithm to practice saying “Say cheese!”:

e.g., for  $i \leftarrow 0$  to  $N$   
    Make him say “Say cheese!”  $2^i$  times oops, it takes exponential time...

4. Show, given a set of numbers, that you can decide whether it has a subset of size 3 that adds to zero in polynomial time.

- 
5. Given a CNF-normalized form that has at most one negative literal in each clause, construct an efficient algorithm to solve the satisfiability problem for these clauses. For instance,

$$\begin{aligned}(A \vee B \vee \bar{C}) \wedge (B \vee \bar{A}), \\ (A \vee \bar{B} \vee C) \wedge (B \vee \bar{A} \vee D) \wedge (A \vee D), \\ (\bar{A} \vee B) \wedge (B \vee \bar{A} \vee C) \wedge (C \vee D)\end{aligned}$$

satisfy the condition, while

$$\begin{aligned}(\bar{A} \vee B \vee \bar{C}) \wedge (B \vee \bar{A}), \\ (A \vee \bar{B} \vee C) \wedge (B \vee \bar{A} \vee \bar{D}) \wedge (A \vee D), \\ (\bar{A} \vee B) \wedge (B \vee \bar{A} \vee C) \wedge (\bar{C} \vee \bar{D})\end{aligned}$$

do not.

6. The ExactCoverByThrees problem is defined as follows: given a finite set  $X$  and a collection  $C$  of 3-element subsets of  $X$ , does  $C$  contain an exact cover for  $X$ , that is, a sub-collection  $C' \subseteq C$  where every element of  $X$  occurs in exactly one member of  $C'$ ? Given that ExactCoverByThrees is NP-complete, show that the similar problem ExactCoverByFours is also NP-complete.

7. The *LongestSimpleCycle* problem is the problem of finding a simple cycle of maximum length in a graph. Convert this to a formal definition of a decision problem and show that it is NP-complete.

Write your answers in the separate answer booklet.

1. **Multiple Choice:** Each question below has one of the following answers.

A:  $\Theta(1)$     B:  $\Theta(\log n)$     C:  $\Theta(n)$     D:  $\Theta(n \log n)$     E:  $\Theta(n^2)$     X: I don't know.

For each question, write the letter that corresponds to your answer. You do not need to justify your answers. Each correct answer earns you 1 point. Each X earns you  $\frac{1}{4}$  point. **Each incorrect answer costs you  $\frac{1}{2}$  point.** Your total score will be rounded **down** to an integer. Negative scores will be rounded up to zero.

- (a) What is  $\sum_{i=1}^n \frac{i}{n}$ ?
- (b) What is  $\sum_{i=1}^n \frac{n}{i}$ ?
- (c) How many bits do you need to write  $10^n$  in binary?
- (d) What is the solution of the recurrence  $T(n) = 9T(n/3) + n$ ?
- (e) What is the solution of the recurrence  $T(n) = T(n - 2) + \frac{3}{n}$ ?
- (f) What is the solution of the recurrence  $T(n) = 5T\left(\lceil \frac{n-17}{25} \rceil - \lg \lg n\right) + \pi n + 2\sqrt{\log^* n} - 6$ ?
- (g) What is the worst-case running time of randomized quicksort?
- (h) The expected time for inserting one item into a randomized treap is  $O(\log n)$ . What is the worst-case time for a sequence of  $n$  insertions into an initially empty treap?
- (i) Suppose STUPIDALGORITHM produces the correct answer to some problem with probability  $1/n$ . How many times do we have to run STUPIDALGORITHM to get the correct answer with high probability?
- (j) Suppose you correctly identify three of the possible answers to this question as obviously wrong. If you choose one of the three remaining answers at random, each with equal probability, what is your expected score for this question?

2. Consider the following algorithm for finding the smallest element in an unsorted array:

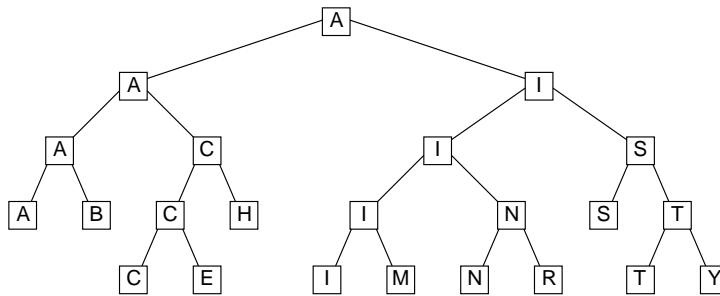
```
RANDOMMIN( $A[1..n]$ ):
     $min \leftarrow \infty$ 
    for  $i \leftarrow 1$  to  $n$  in random order
        if  $A[i] < min$ 
             $min \leftarrow A[i]$  (*)
```

return  $min$

- (a) [1 point] In the worst case, how many times does RANDOMMIN execute line (\*)?
- (b) [3 points] What is the probability that line (\*) is executed during the  $n$ th iteration of the for loop?
- (c) [6 points] What is the exact expected number of executions of line (\*)? (A correct  $\Theta()$  bound is worth 4 points.)

3. Algorithms and data structures were developed millions of years ago by the Martians, but not quite in the same way as the recent development here on Earth. Intelligent life evolved independently on Mars' two moons, Phobos and Deimos.<sup>1</sup> When the two races finally met on the surface of Mars, after thousands of Phobos-orbits<sup>2</sup> of separate philosophical, cultural, religious, and scientific development, their disagreements over the proper structure of binary search trees led to a bloody (or more accurately, ichorous) war, ultimately leading to the destruction of all Martian life.

A *Phobian* binary search tree is a full binary tree that stores a set  $X$  of search keys. The root of the tree stores the *smallest* element in  $X$ . If  $X$  has more than one element, then the left subtree stores all the elements less than some pivot value  $p$ , and the right subtree stores everything else. Both subtrees are *nonempty* Phobian binary search trees. The actual pivot value  $p$  is *never* stored in the tree.



A Phobian binary search tree for the set  $\{M, A, R, T, I, N, B, Y, S, C, H, E\}$ .

- (a) [2 points] Describe and analyze an algorithm  $\text{FIND}(x, T)$  that returns TRUE if  $x$  is stored in the Phobian binary search tree  $T$ , and FALSE otherwise.
  - (b) [2 points] Show how to perform a rotation in a Phobian binary search tree in  $O(1)$  time.
  - (c) [6 points] A *Deimoid* binary search tree is almost exactly the same as its Phobian counterpart, except that the *largest* element is stored at the root, and both subtrees are Deimoid binary search trees. Describe and analyze an algorithm to transform an  $n$ -node Phobian binary search tree into a Deimoid binary search tree in  $O(n)$  time, using as little additional space as possible.
4. Suppose we are given an array  $A[1..n]$  with the special property that  $A[1] \geq A[2]$  and  $A[n-1] \leq A[n]$ . We say that an element  $A[x]$  is a *local minimum* if it is less than or equal to both its neighbors, or more formally, if  $A[x-1] \geq A[x]$  and  $A[x] \leq A[x+1]$ . For example, there are five local minima in the following array:

9	7	7	2	<b>1</b>	3	7	5	4	7	<b>3</b>	<b>3</b>	4	8	<b>6</b>	9
---	---	---	---	----------	---	---	---	---	---	----------	----------	---	---	----------	---

We can obviously find a local minimum in  $O(n)$  time by scanning through the array. Describe and analyze an algorithm that finds a local minimum in  $O(\log n)$  time. [Hint: With the given boundary conditions, the array **must** have at least one local minimum. Why?]

<sup>1</sup>Greek for “fear” and “panic”, respectively. Doesn’t that make you feel better?

<sup>2</sup>1000 Phobos orbits  $\approx 1$  Earth year

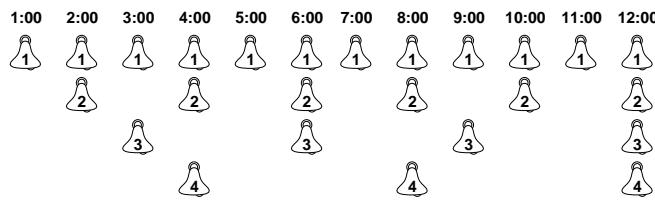
5. [Graduate students must answer this question.]

A *common supersequence* of two strings  $A$  and  $B$  is a string of minimum total length that includes both the characters of  $A$  in order and the characters of  $B$  in order. Design and analyze an algorithm to compute the length of the *shortest* common supersequence of two strings  $A[1..m]$  and  $B[1..n]$ . For example, if the input strings are ANTHROHOPOBIOLOGICAL and PRETERDIPLOMATICALLY, your algorithm should output 31, since a shortest common supersequence of those two strings is PREANTHEROHODPOBIOPLOMATGICALY. You do not need to compute an actual supersequence, just its length. For full credit, your algorithm must run in  $\Theta(nm)$  time.

**Write your answers in the separate answer booklet.**  
**This is a 90-minute exam. The clock started when you got the questions.**

1. Professor Quasimodo has built a device that automatically rings the bells in the tower of the Cathédrale de Notre Dame de Paris so he can finally visit his true love Esmerelda. Every hour exactly on the hour (when the minute hand is pointing at the 12), the device rings at least one of the  $n$  bells in the tower. Specifically, the  $i$ th bell is rung once every  $i$  hours.

For example, suppose  $n = 4$ . If Quasimodo starts his device just after midnight, then his device rings the bells according to the following twelve-hour schedule:



What is the *amortized* number of bells rung per hour, as a function of  $n$ ? For full credit, give an exact closed-form solution; a correct  $\Theta()$  bound is worth 5 points.

2. Let  $G$  be a directed graph, where every edge  $u \rightarrow v$  has a weight  $w(u \rightarrow v)$ . To compute the shortest paths from a start vertex  $s$  to every other node in the graph, the generic single-source shortest path algorithm calls INITSSSP once and then repeatedly calls RELAX until there are no more tense edges.

**INITSSSP( $s$ ):**

```

dist( $s$ )  $\leftarrow 0$ 
pred( $s$ )  $\leftarrow \text{NULL}$ 
for all vertices  $v \neq s$ 
    dist( $v$ )  $\leftarrow \infty$ 
    pred( $v$ )  $\leftarrow \text{NULL}$ 

```

**RELAX( $u \rightarrow v$ ):**

```

if dist( $v$ )  $>$  dist( $u$ ) +  $w(u \rightarrow v)$ 
    dist( $v$ )  $\leftarrow$  dist( $u$ ) +  $w(u \rightarrow v)$ 
    pred( $v$ )  $\leftarrow u$ 

```

Suppose the input graph has no negative cycles. Let  $v$  be an arbitrary vertex in the input graph. **Prove** that after every call to RELAX, if  $\text{dist}(v) \neq \infty$ , then  $\text{dist}(v)$  is the total weight of some path from  $s$  to  $v$ .

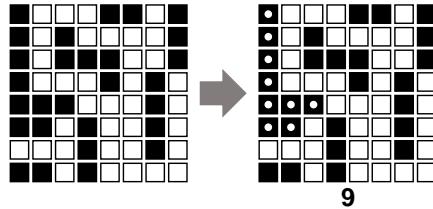
3. Suppose we want to maintain a dynamic set of values, subject to the following operations:

- **INSERT( $x$ ):** Add  $x$  to the set (if it isn't already there).
- **PRINT&DELETERANGE( $a, b$ ):** Print and delete every element  $x$  in the range  $a \leq x \leq b$ .  
For example, if the current set is  $\{1, 5, 3, 4, 8\}$ , then PRINT&DELETERANGE(4, 6) prints the numbers 4 and 5 and changes the set to  $\{1, 3, 8\}$ .

Describe and analyze a data structure that supports these operations, each with amortized cost  $O(\log n)$ .

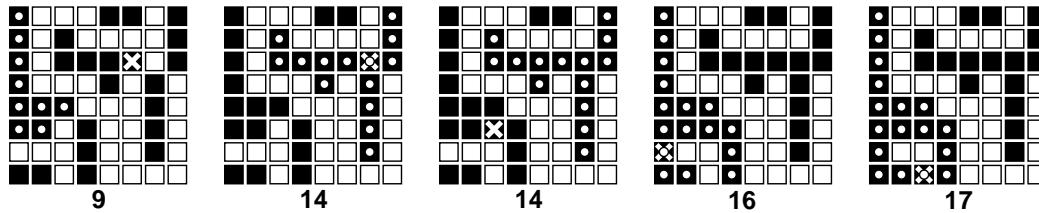
4. (a) [4 pts] Describe and analyze an algorithm to compute the size of the largest connected component of black pixels in an  $n \times n$  bitmap  $B[1..n, 1..n]$ .

For example, given the bitmap below as input, your algorithm should return the number 9, because the largest connected black component (marked with white dots on the right) contains nine pixels.



- (b) [4 pts] Design and analyze an algorithm  $\text{BLACKEN}(i, j)$  that colors the pixel  $B[i, j]$  black and returns the size of the largest black component in the bitmap. For full credit, the *amortized* running time of your algorithm (starting with an all-white bitmap) must be as small as possible.

For example, at each step in the sequence below, we blacken the pixel marked with an X. The largest black component is marked with white dots; the number underneath shows the correct output of the  $\text{BLACKEN}$  algorithm.



- (c) [2 pts] What is the *worst-case* running time of your  $\text{BLACKEN}$  algorithm?

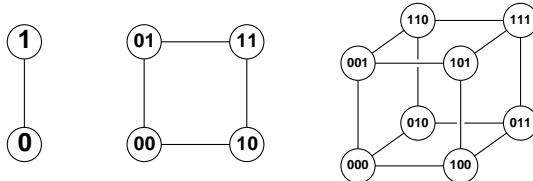
5. [Graduate students must answer this question.]

After a grueling 373 midterm, you decide to take the bus home. Since you planned ahead, you have a schedule that lists the times and locations of every stop of every bus in Champaign-Urbana. Unfortunately, there isn't a single bus that visits both your exam building and your home; you must transfer between bus lines at least once.

Describe and analyze an algorithm to determine the sequence of bus rides that will get you home as early as possible, assuming there are  $b$  different bus lines, and each bus stops  $n$  times per day. Your goal is to minimize your *arrival time*, not the time you actually spend travelling. Assume that the buses run exactly on schedule, that you have an accurate watch, and that you are too tired to walk between bus stops.

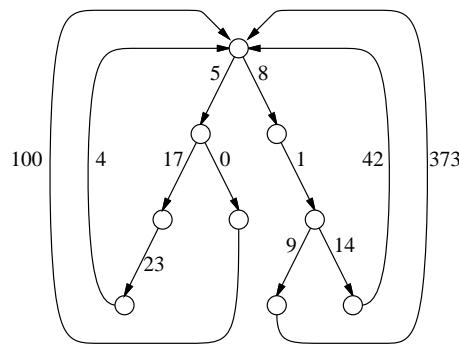
**Write your answers in the separate answer booklet.**  
**This is a 180-minute exam. The clock started when you got the questions.**

1. The  $d$ -dimensional hypercube is the graph defined as follows. There are  $2^d$  vertices, each labeled with a different string of  $d$  bits. Two vertices are joined by an edge if their labels differ in exactly one bit.



The 1-dimensional, 2-dimensional, and 3-dimensional hypercubes.

- (a) [8 pts] Recall that a Hamiltonian cycle passes through every vertex in a graph exactly once. **Prove** that for all  $d \geq 2$ , the  $d$ -dimensional hypercube has a Hamiltonian cycle.
- (b) [2 pts] Which hypercubes have an Eulerian circuit (a closed walk that visits every edge exactly once)? [Hint: This is very easy.]
2. A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has a non-negative weight. The number of nodes in the graph is  $n$ .



- (a) How long would it take Dijkstra's algorithm to compute the shortest path between two vertices  $u$  and  $v$  in a looped tree?
- (b) Describe and analyze a faster algorithm.
3. Prove that  $(x + y)^p \equiv x^p + y^p \pmod{p}$  for any prime number  $p$ .

4. A *palindrome* is a string that reads the same forwards and backwards, like X, 373, noon, redivider, or amanaplanacatahamayakayamahatacanalpanama. Any string can be written as a sequence of palindromes. For example, the string **bubbasesabananana** ('Bubba sees a banana.') can be decomposed in several ways; for example:

$$\begin{aligned} &\text{bub} + \text{baseesab} + \text{anana} \\ &\text{b} + \text{u} + \text{bb} + \text{a} + \text{sees} + \text{aba} + \text{nan} + \text{a} \\ &\text{b} + \text{u} + \text{bb} + \text{a} + \text{sees} + \text{a} + \text{b} + \text{anana} \\ &\text{b} + \text{u} + \text{b} + \text{b} + \text{a} + \text{s} + \text{e} + \text{e} + \text{s} + \text{a} + \text{b} + \text{a} + \text{n} + \text{a} + \text{n} + \text{a} \end{aligned}$$

Describe an efficient algorithm to find the *minimum* number of palindromes that make up a given input string. For example, given the input string **bubbasesabananana**, your algorithm would return the number 3.

5. Your boss wants you to find a *perfect* hash function for mapping a known set of  $n$  items into a table of size  $m$ . A hash function is perfect if there are *no* collisions; each of the  $n$  items is mapped to a different slot in the hash table. Of course, this requires that  $m \geq n$ .

After cursing your 373 instructor for not teaching you about perfect hashing, you decide to try something simple: repeatedly pick *random* hash functions until you find one that happens to be perfect. A random hash function  $h$  satisfies two properties:

- $\Pr[h(x) = h(y)] = \frac{1}{m}$  for any pair of items  $x \neq y$ .
- $\Pr[h(x) = i] = \frac{1}{m}$  for any item  $x$  and any integer  $1 \leq i \leq m$ .

- (a) [2 pts] Suppose you pick a random hash function  $h$ . What is the *exact* expected number of collisions, as a function of  $n$  (the number of items) and  $m$  (the size of the table)? Don't worry about how to *resolve* collisions; just count them.
- (b) [2 pts] What is the *exact* probability that a random hash function is perfect?
- (c) [2 pts] What is the *exact* expected number of different random hash functions you have to test before you find a perfect hash function?
- (d) [2 pts] What is the *exact* probability that none of the first  $N$  random hash functions you try is perfect?
- (e) [2 pts] How many random hash functions do you have to test to find a perfect hash function *with high probability*?

To get full credit for parts (a)–(d), give *exact* closed-form solutions; correct  $\Theta(\cdot)$  bounds are worth significant partial credit. Part (e) requires only a  $\Theta(\cdot)$  bound; an exact answer is worth extra credit.

6. Your friend Toidi is planning to hold a Christmas party. He wants to take a picture of all the participants, including himself, but he is quite shy and thus cannot take a picture of a person whom he does not know very well. Since he has only shy friends<sup>1</sup>, everyone at the party is also shy. After thinking hard for a long time, he came up with a seemingly good idea:

- Toidi brings a disposable camera to the party.
- Anyone holding the camera can take a picture of someone they know very well, and then pass the camera to that person.
- In order not to waste any film, every person must have their picture taken exactly once.

Although there can be some people Toidi does not know very well, he knows completely who knows whom well. Thus, *in principle*, given a list of all the participants, he can determine whether it is possible to take all the pictures using this idea. But how quickly?

Either describe an efficient algorithm to solve Toidi's problem, or show that the problem is NP-complete.

7. The recursion fairy's cousin, the reduction genie, shows up one day with a magical gift for you: a box that can solve the NP-complete PARTITION problem in constant time! Given a set of positive integers as input, the magic box can tell you in constant time it can be split into two subsets whose total weights are equal.

For example, given the set  $\{1, 4, 5, 7, 9\}$  as input, the magic box cheerily yells "YES!", because that set can be split into  $\{1, 5, 7\}$  and  $\{4, 9\}$ , which both add up to 13. Given the set  $\{1, 4, 5, 7, 8\}$ , however, the magic box mutters a sad "Sorry, no."

The magic box does not tell you *how* to partition the set, only whether or not it can be done. Describe an algorithm to actually split a set of numbers into two subsets whose sums are equal, **in polynomial time**, using this magic box.<sup>2</sup>

---

<sup>1</sup>Except you, of course. Unfortunately, you can't go to the party because you're taking a final exam. Sorry!

<sup>2</sup>Your solution to problem 4 in homework 1 does *not* solve this problem in polynomial time.