CS 170 Efficient Algorithms and Intractable Problems
Spring 2016 Alessandro Chiesa and Umesh Vazirani **Final**

Name:

SID:

GSI and section time:

Write down the names of the students on your left and right as they appear on their SID.

Name of student on your left:

Name of student on your right:

Name of student behind you:

Name of student in front of you:

Instructions: You may consult two handwritten, double-sided sheet of notes. You may not consult other notes, textbooks, etc. Cell phones and other electronic devices are not permitted.

Blank policy: Blanks will receive 0 points. Whether you have a solution or just some progress on a problem, make sure to cross out any extraneous work: we will reward succinct/clear solutions and penalize extra text. (Don't write down a bunch of formulas / attempts in the hopes one is correct; this will receive no partial credit.)

If you finish in the last 15 minutes, please remain seated to avoid distracting your fellow classmates.

There are 7 questions. The last page is page 12. *Please write your name and SID on the top of each sheet, and do not detach the last sheet.*

On questions asking for an algorithm, make sure to respond in the format requested. Write in the space provided. Good luck!

Do not turn this page until an instructor tells you to do so.

1. (40 pts.) Short Answer

(a) **(4 pts.) Primality** Use Fermat's Little Theorem with a base of $a = 3$ to show that 8 is not prime. *Explain your answer in a single sentence.*

Solution: $3^7 \equiv 3 * (3^2)^3 \equiv 3 * 1^3 \equiv 3 \not\equiv 1 \bmod 8$
FLT states that for all primes p , and bases $0 < a < p$, $a^{p-1} \equiv 1 \pmod{p}$; since in this case, $a^{p-1} \not\equiv 1 \pmod{p}$, we know that p is not actually prime.
Alternate approach: It would also have been correct to use the form of FLT which says $a^p = a \bmod p$ for prime p; this might have been slightly easier to calculate.

(b) **(8 pts.) Hashing**

Given a prime p and $a, b \in \{0, \ldots, p-1\}$, define the function $h_{a,b}(x) = ax + b \bmod p$ where $x \in \{0, \ldots, p-1\}$. Show that $H = \{h_{a,b}\}_{a,b \in \{0,\ldots,p-1\}}$ is a pairwise independent hash function family, i.e., show that for every $x \neq y$ and $c, d \in \{0, \ldots, p-1\}$ it holds that

$$\Pr_{h_{a,b} \leftarrow H}\left[h_{a,b}(x) = c \wedge h_{a,b}(y) = d\right] = \frac{1}{p^2} \ .$$

The notation $h_{a,b} \leftarrow H$ means that $h_{a,b}$ is chosen uniformly at random from H, meaning a and b are chosen independently uniformly at random from $\{0, \ldots, p-1\}$.

Solution: For this entire solution, all equations are $\bmod p$.
$h(x) = c$ and $h(y) = d$ if and only if $ax + b = c$ and $ay + b = d$. This is true if and only if a, b solve the system of equations

$$ax + b = c$$
$$ay + b = d$$

Solving this, we have

$$a = (c - d)(x - y)^{-1}$$
$$b = (cy - dx)(y - x)^{-1}$$

We are guaranteed that the multiplicative inverses exist because p is prime, and we know $x \neq y$. Thus, there is only one value of a and one value of b that satisfy these equations, which are chosen independently at random: the probability of this occurring is $1/p^2$.

(c) **(8 pts.) Zero-sum games** Alice and Bob play a game: each player chooses either X or Y. If both players pick X, Bob pays Alice \$1. If both players pick Y, Bob pays Alice \$2. Otherwise, Alice pays Bob \$1.

(i) Formulate this as a zero-sum game by filling in the table below. Alice is the row player and Bob is the column player. Fill in the values so that Alice is the maximizer and Bob is the minimizer.

		Bob: X	Bob: Y
Alice	X		
	Y		

Solution:

		Bob: X	Bob: Y
Alice	X	1	-1
	Y	-1	2

(ii) What are the optimal strategies and for Alice and Bob respectively? *No justification required.*

Alice's optimal strategy:

probability of playing X: _____

probability of playing Y: _____

Bob's optimal strategy:

probability of playing X: _____

probability of playing Y: _____

Solution: Both are $(3/5, 2/5)$

(iii) What is the value of the game? *No justification required.*

Value: ☐

Solution: $1/5$

(d) **(6 pts.) Complexity** For the following questions, circle the (unique) condition that would make the statement true. *No justification required.*

(i) If B is **NP**-complete, then for any problem $A \in$ **NP**, there exists a polynomial-time reduction from A to B.

always True True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ always False

Solution: Always true: this is the definition of **NP**-hard, and all **NP**-complete problems are **NP**-hard.

(ii) If B is in **NP**, then for any problem $A \in$ **P**, there exists a polynomial-time reduction from A to B.

always True True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ always False

Solution: Always true: since we have polynomial time for our reduction, we have enough problem to simply solve any instance of A during the reduction.

(iii) Horn SAT is **NP**-complete.

always True True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ always False

Solution: True iff $\mathbf{P} = \mathbf{NP}$: Horn SAT is in **P**.

(e) **(6 pts.) Duality**

For the following linear program, prove that the value of the optimal solution is at most $7/3$ by finding the values of the dual variables y_1 and y_2.

$$\begin{aligned} &\max\ 2x_1 + x_2 \\ &\text{such that } 5x_1 + 2x_2 \leq 5 \\ &\quad x_1 + x_2 \leq 2 \end{aligned}$$

$y_1 =$ $\square$ $y_2 =$ $\square$

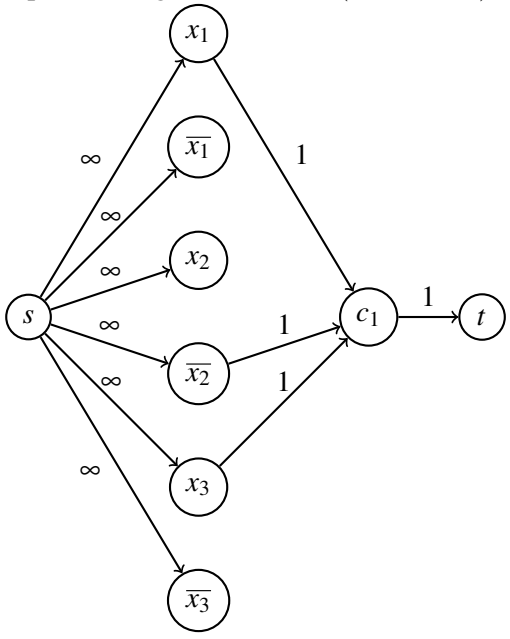
Solution: $1/3, 1/3$

(f) **(8 pts.) Bad Reduction** What's wrong with the following reduction from 3SAT to Max Flow? Give a counterexample to show why this reduction does not hold.

Given a 3SAT instance $(c_1, \ldots, c_m)$, we create a Max Flow instance (G, s, t, F) as follows:
For each literal x_i, we create two vertices, x_i and $\overline{x_i}$. For each clause c_j, we create a vertex c_j. We create two more vertices, s and t. Add an edge of infinite capacity from s to each x_i and each $\overline{x_i}$.

Add an edge of capacity 1 from each c_j to t. For each clause c_j, add an edge of capacity 1 from the literals in that clause to the vertex c_j. For example, if $c_5 = (x_2 \vee \overline{x_4} \vee x_9)$, we create edges of capacity 1 from x_2 to c_5, $\overline{x_4}$ to c_5, and x_9 to c_5. Then, G has an $s-t$ flow of F if and only if the 3SAT formula is satisfiable.

For example, the single clause $c_1 = (x_1 \vee \overline{x_2} \vee x_3)$ would yield the following graph:



Solution: There is no constraint preventing both x_i and $\overline{x_i}$ from being set to true. One counterexample is $(x_1) \wedge (\overline{x_1})$.

2. (20 pts.) Cyclic Sorting?

Tanya the TA has a stack of final exams, sorted by score. She wanted to show Prof Vazirani the top score, 105%! Sadly, Charles the clueless TA has split the stack as follows: he picked up the top i exams in the stack, and moved them to the bottom. Help Tanya find the exam with the maximum score again!

Formally, an original sorted array of integers $A[1..n]$ has been cycled: $C = A[i+1..n] \cdot A[1..i]$ for some i such that $0 \leq i < n$, and in this case $\cdot$ is the concatenation operator. You have access to C, but not A or i. Give an efficient algorithm to find the maximum value in the array.

For example, if $A = [-5, -3, 0, 42, 100, 2016]$ and $C = [100, 2016, -5, -3, 0, 42]$, and the maximum value is 2016.

You may assume that all the values in A are distinct.

(a) Give clear pseudocode to solve this problem.

Solution:

Algorithm 1 UNCYCLE$(C[i..j])$

```
if j − i ≤ 3 then
    return max(C[i..j])
m = ⌊j + i⌋/2
if C[i] < C[m] then
    return UNCYCLE(C[m..j])
else
    return UNCYCLE(C[i..m])
```

Algorithm 2 UNCYCLE V2$(C[i..j])$

```
if j − i ≤ 3 then
    return max(C[i..j])
if C[j] > C[i] then
    return C[j]
m = ⌊j + i⌋/2
if C[j] < C[m] then
    return UNCYCLE(C[m..j])
else
    return UNCYCLE(C[i..m])
```

Note: we only awarded you the 4 runtime points if your pseudocode was correct/had a minor error.

Note: dividing the array into more than two chunks was OK if your code worked.

Common Mistakes:

- Code failing on a sorted list (sometimes the only issue was if the very last element was the largest).
- Having two calls to your recursive function, one for each half of the array. This runs in linear time.
- Submitting a linear implementation that loops through the list and returns the max. This should have raised red flags as being inefficient because the solution was so simple.
- Making comparisons only between the middle element and elements immediately before and after it – $C[m], C[m-1]$, and $C[m+1]$. This is not enough information to tell you which half of the array to look at.

(b) What is the runtime of your algorithm? *No justification required.*

Solution: $O(\log n)$

3. (20 pts.) Summer Subletting

You're going to be traveling this summer, and you've listed your apartment on Airbnb. Due to your prime location on Telegraph and Blake and lovely first-story view of the charming streetscape, n guests have expressed interest in renting your apartment. Airbnb is testing out a model where potential guests submit their offer (the total price they're willing to pay for the duration of their stay). Formally, find an efficient algorithm for the following task:

Input: For $1 \leq i \leq n$, (s_i, t_i, p_i) is an offer for a guest who arrives on day s_i, leaves on day t_i, and pays $p_i \geq 0$.

Output: The maximum total value $P^* = \sum_{i \in S^*} p_i$, where $S^* \subseteq \{1, \ldots, n\}$ is the optimal non-overlapping subset of guests. It is OK for a guest to arrive on the same day that the previous guest leaves.

Note: Your runtime should be polynomial in n, the number of guests. If it exceeds our expectations, we will award extra credit.

(a) We'll solve this problem with dynamic programming. First, *clearly* define your subproblems. This consists of defining a function $V(\cdot)$ with some number of arguments, and explaining the meaning of each argument and the value of the function.

Solution: $V(i) =$ the maximum value possible using some subset of guests $1..i$, where the guests are sorted by t_i.

Alternate solution: As above, except requiring that the chosen subset included guest i.

Alternate solution 2: There are at most $2n$ unique dates (the s_i's and t_i's), and we only care about the dates in their relative order, and not their absolute values. Thus, we can utilize the method of coordinate compression, and remap these dates into the range $1...2n$ (perhaps less, if there are duplicates). Now, we can define our subproblem to use these dates instead: $V(d)$ is the maximum value possible renting only during dates $1..d$.

Common Mistakes:

- As the problem states, the definition of a subproblem is a function, its arguments, the meaning of the arguments, and the meaning of the value of the function. The definition of a subproblem is *not* how to solve each subproblem, or the recurrence.
- Almost every answer that includes time or "days" as an argument was incorrect: if the number of subproblems is proportional to the number of days, then this will lead to an exponential algorithm. The only way to get a polynomial-runtime algorithm with the subproblem involving dates is via the second alternate solution above.
- Sorting by s_i works for the $\Theta(n^2)$ solution below, but not the $\Theta(n \log n)$ solution. This is necessary, in order to ensure that we don't rule out guests too early. (For example, if we don't sort and guest i leaves before guest j arrives, then we are unable to consider the possibility of accepting the offers of both, without an exponential-runtime solution.)
- Including problem parameters (e.g. s_i, t_i) as subarguments isn't wrong, strictly speaking, but is unnecessary and suspicious of not understanding how DP works.

(b) Write pseudocode to solve this problem. It should fill in the entries of $V(\cdot)$, and return the maximum total value P^*. Make sure to address the base case(s).

Solution:

Algorithm 3 SUBLET(s,t,p)

Sort the guests in increasing order of t
$V(0) = 0$
for $i = 1,\ldots,n$ **do**
    $V(i) = \max(V(i-1), p_i + \max_{j:t_j \le s_i} V(j))$ ▷ This can be done with binary search
return $V(n)$

Alternate solution 1, using the subproblem definition including the ith guest:

Algorithm 4 SUBLET(s,t,p)

Sort the guests in increasing order of t
$V(0) = 0$
for $i = 1,\ldots,n$ **do**
    $V(i) = \max(0, p_i + \max_{j:t_j \le s_i} V(j))$ ▷ This can be done with binary search
return $\max_i V(i)$

Alternate solution 2, using coordinate compression:

Algorithm 5 SUBLET(s,t,p)

Sort the s_i's and t_i's together, and relabel them into the range $1..x$ (where x is the number of distinct dates.)
Let $A[d]$ be a map from (relabeled) dates to a list of guests whose desired tenancy ends on that day.
Iterate through the guests, to populate A.
$V(0) = 0$
for $d = 1,\ldots,x$ **do**
    $V(i) = \max(V(i-1), p_i + \max_{j:j \in A[i]} V(s_j))$
return $V(x)$

Common Mistakes:

- Guests have specific dates that they want to start and end their sublet period. The usage of the expression $t_i - s_i$ is indeed the number of days they want to sublet, but unlikely to be part of a correct answer.
- If the subproblem definition was the first above (i.e. $V(i)$ does not necessarily include guest i), then it was important to include $V(i) = max(V(i-1),\ldots)$.
- We provided the name of the function with values to solve for. Please do not ignore instructions and rename this function or go off solving for a different subproblem.
- For explanations of why looking at number of "days" was problematic or issues with not sorting, see the common mistakes section of part a.

(c) What is the running time of your algorithm? *No justification required.*

Solution: $O(n^2)$ was the desired runtime, that received full credit.
$O(n \log n)$ was also possible (if the solution used binary search or coordinate compression approach).

4. (20 pts.) The Bold Bluff

You are investing in stocks in the Delphi Exchange. The Delphi Exchange has *only two stocks*, and on each day you either stay with your current stock, or sell all of your holdings and invest all your money in the other stock.

On each day, the price of one stock goes down by 75% (multiplies by 0.25, and this may be a different stock each day), and the other stock doubles. There are 64 oracles in Delphi, who give, on each day, their forecast for the change in stock prices for the next day. You are guaranteed that at least one oracle is always correct.

You start with \$1, and you have 64 oracles. Describe a strategy such that after 28 days, you are guaranteed to have at least \$1000. Justify your answer.

Hint: Don't use the formula from the Multiplicative Weights / Experts lecture.

Main idea:

Solution: On each day, choose the stock selected by the majority of the oracles. After seeing the outcome of that day, eliminate the oracles that forecasted incorrectly.

Analysis:

Solution: You can be wrong at most 6 times, so you lose a factor of $1/4^6 = 1/2^{12}$. It takes you 12 days to recover from this, and another 10 days to get from 1 to 1024. 6 days losing + 12 days to get back + 10 days to get to 1024 is 28 days.

5. (20 pts.) Rad Reduction You are the captain of a ship, tasked with taking animals across the river. However, not all animals get along – some animals will attack each other. If the ith animal gets attacked, you have to pay the owner a fee of f_i. If the ith animal makes it across the river without getting attacked, you receive a reward of r_i.

In other words, for each animal i, your payoff is as follows:

- 0 if you don't take the animal
- r_i if you take the animal and it doesn't get attacked
- $-f_i$ if you take the animal and it gets attacked

You want to know if there's a subset of animals you can take such that your net profit (the sum of the rewards you get minus the sum of the fees you pay) is at least k.

Input: $[f_1, \ldots, f_n]$; $[r_1, \ldots, r_n]$; k; $E = \{(i, j) : \text{animal } i \text{ attacks animal } j\}$.

Prove that this problem is NP-complete. You may start from the fact that any of the following problems are NP-complete: 3-SAT, circuit-SAT, clique, independent set, TSP, Rudrata path.

Main idea:

Solution:

- Reduce from Independent Set; this shows problem is NP-hard.
- In NP: enumerate over animals on ship; for animal i subtract f_i if it is attacked by another animal on ship, and add r_i otherwise.

Reduction:

Solution: Given an independent sets instance (G, k), construct an animal crossing instance with $|V|$ animals as follows:

$$
\begin{aligned}
f_i &= \infty \\
r_i &= 1 \\
k' &= k \\
E' &= E \qquad \text{(turn undirected edge } (U, v) \text{ into 2 directed edges } (u \to v) \text{ and } (v \to u))
\end{aligned}
$$

Remark: instead of $f_i = \infty$ one could set f_i to any non-negative value.

Proof:

Solution:

- If there's an independent sets solution S of size $\geq k$, take those same animals. You have $\geq k$ of them, so your reward is at least k. None attack each other because there are no edges between them.
- If there's an animal crossing solution S, it can't contain any animals that attack each other because otherwise the total profit would be $-\infty$. At least k animals must cross because all the rewards are 1. Thus, we have $\geq k$ animals that don't attack each other, so those same $\geq k$ vertices have no edges between them.

6. (20 pts.) Max 2SAT We saw in class that there is an efficient algorithm to solve 2SAT. Here we consider the Max 2SAT problem: given a set of clauses C such that each clause has at most 2 literals, and integer k, is there a truth assignment that satisfies at least k clauses? We will prove that Max 2SAT is NP-complete by reducing from 3SAT.

(a) Show that the 2SAT formula $(x_1) \wedge (x_2) \wedge (x_3) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_3} \vee \overline{x_1})$ has 3 clauses satisfied if x_1, x_2, and x_3 are either all true or all false, and 4 clauses satisfied otherwise.

Solution: Draw the truth table.

(b) Show that the 2SAT formula $(\overline{z}) \wedge (x_1 \vee z) \wedge (x_2 \vee z) \wedge (x_3 \vee z)$ has 4 clauses satisfied if x_1, x_2, and x_3 are all true, and 3 clauses satisfied otherwise.

Solution: If x_1, x_2, x_3 are all true and z is false, all 4 clauses will be satisfied. Otherwise, if more than one x_i is false, then setting z to true will satisfy 3 clauses.

(c) Show that Max 2SAT is NP-complete.

Solution: For each clause, construct the 10 clauses described above, adding a new extra variable z each time. Each clause is satisfied iff exactly 7 of the 10 new clauses is satisfied. Thus, the 3SAT instance is satisfiable iff the Max 2SAT instance can have $7m$ clauses satisfied.

7. (30 pts.) Fruit Fanatic

(a) You are managing Fred's Fantastic Fruits delivery service. Your job is to choose which fruits to take to the market in your truck. You have n crates of fruit, each of which contains a different type of fruit. The ith crate holds c_i fruits, each of which sells for p_i at the market. Every fruit weighs 1 pound. Your truck can hold T pounds of fruit in total. You are allowed to open all the crates and take any combination of fruits you like, as long as the total weight of those fruits is no more than T. For the remainder of this problem, you may assume that T is an integer and that $c_i \leq T$ for all i.

(i) Describe a greedy strategy to determine the optimal combination of fruits to take (no pseudocode necessary).
Solution: Take the most expensive fruits until you run out of room.

(ii) Prove your algorithm is correct using an exchange argument.
Solution: Sort the fruits in any solution, replace the first one that differs from the greedy solution with the corresponding fruit in the greedy solution, which must be worth at least as much.
Common mistakes:

- Start with greedy, can't get better by swapping – wrong because this shows that greedy is better than any solution that differs by 1, but not that it's better than every solution. The whole point of the exchange argument is to say that by changing any other solution to look more like the greedy solution, things can only get better, and by induction on the number of differences between greedy and the other solution, no solution can be better than greedy
- Change the order of the items, the value doesn't change – this doesn't say anything

(b) You return home from the market dejected, as the fruits didn't survive the long drive to the market in the back of your truck. You realize that the only way to transport them safely is to keep them in their crates. This means that you can either take the ith crate or not – you don't get to take individual fruit out of the crate. However, you realize that this is exactly the knapsack (without replacement) problem!

(i) Reduce this problem to the knapsack problem by defining v_i (the value of the ith knapsack item), w_i (the weight of the ith knapsack item), and W (the total weight of the knapsack) in terms of the variables c_i, p_i, and T from the previous problem.

$$v_i =$$
$$w_i =$$
$$W =$$

Solution:

$$v_i = p_i c_i$$
$$w_i = c_i$$
$$W = T$$

(ii) Since you can't solve knapsack exactly, describe an approximation scheme for knapsack that yields at least half the optimal value. Prove that your algorithm gives the desired approximation. You may assume that W and w_i are all integers.
Hint: Use your greedy algorithm from part (a) as inspiration.
Solution: Sort by v_i/w_i, let k be the smallest number such that you can't take the k crates with the highest v_i/w_i ratio. Then, your solution is to either take the first $k-1$ crates or take the kth

crate, whichever has more value. Both are feasible. If you took all k crates, this would have a larger value than the one produced by the greedy strategy. Either the first $k-1$ or the kth must have a value at least of half of the value of the first k, so one of those must have a value at least half the greedy value, which is larger than the optimal knapsack value.

Common mistakes:

- Sort by v_i
- Sort by v_i/w_i and take the first k that fit. Counterexample: $(v_1, w_1) = (2, 1)$, $(v_2, w_2) = (10, 10)$, $W = 10$. This version of the greedy strategy takes item 1 for a total value of 2, whereas the optimal solution is to take item 2 for a value of 10.

Extra page

Use this page for scratch work, or for more space to continue your solution to a problem. If you want this page to be graded, please refer the graders here from the original problem page.