# CS 373: Combinatorial Algorithms, Spring 2001

**Homework 1 (due Thursday, February 1, 2001 at 11:59:59 p.m.)**

| Name: | | |
|---|---|---|
| Net ID: | Alias: | U 3/4 1 |

| Name: | | |
|---|---|---|
| Net ID: | Alias: | U 3/4 1 |

| Name: | | |
|---|---|---|
| Net ID: | Alias: | U 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, 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 $1 = c[1] < c[2] < \cdots < c[n]$. (In the U.S., for example, $n = 6$ and the values are $1, 5, 10, 25, 50$ and $100$ cents.) Your beloved and belevolent 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?

   $^\star$(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($A[1\,..\,n], r$) :
    $p \leftarrow$ Random($1, n$)
    $k \leftarrow$ 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[1], where *container*$[i]$ is the name of a container that holds $2^i$ ounces of beer.[2]

```
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 *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[3]. Suppose *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 *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.)

[1]Pseudolyrics are to lyrics as pseudocode is to code.

[2]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.

[3]"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.