Ground Rules

1. This exam is due at 2:00 am on October 28.

2. You may not discuss this exam with anyone except Prof. Drysdale until after you have turned it in.

3. The only written materials you may consult are the text *Introduction to Algorithms*, your lecture notes, your homeworks, and my sample solutions. In particular you may not look up things on the internet.

4. In all questions that ask you to give an algorithm, you must argue that your algorithm is correct. For dynamic programming and greedy algorithms include all of the parts that were specified in the Announcements on Canvas.

5. Any changes or clarifications will be posted as Announcements, so check regularly.
Problem 1: (20 points)

You work for an online ski shop. People order skis, and your job is to package them up and ship them. To protect the skis each is shipped in a ski bag. The bags have various lengths and costs. For each pair of skis you need to find the least expensive bag that is long enough to fit the ski. Suppose your bag inventory consisted of the following (length, cost) pairs:

\[(100, 20), (130, 50), (150, 30), (175, 40), (200, 35)\]

For a ski of length 100 you should pick the \((100, 20)\) bag. For a ski of length 120 you should pick the \((150, 30)\) bag. For a ski of length 160 you should pick the \((200, 35)\) bag. For a ski of length 210 you should return \(null\) to indicate that no such bag exists.

Design a data structure that will support each of the following operations in \(O(\log n)\) time, where \(n\) is the number of bags currently in your bag inventory:

- \(\text{insert}(D, \text{length}, \text{cost})\) - add a bag of the given length and cost to data structure \(D\).
- \(\text{delete}(D, \text{length}, \text{cost})\) - delete a bag of the given length and cost from the data structure \(D\).
- \(\text{getBag}(D, \text{length})\) - return a reference to a node in data structure \(D\) that contains the least expensive bag whose length is greater than or equal to \(\text{length}\). Return \(null\) if no such bag is in \(D\).

You should give pseudocode for the \(\text{getBag}\) procedure and explain why the \(\text{insert}\) and \(\text{delete}\) operations run in \(O(\log n)\) time, where \(n\) is the number of bags currently in your data structure \(D\).

Problem 2: (16 points)

You have a box containing \(n\) pieces of string. You pick two ends at random and tie them together. This creates either a longer piece of string or a loop. You repeat this until you have nothing but loops.

a) (3 points) How many steps will it take until you have only loops? (Hint - What happens to the number of free ends at each step?)

b) (13 points) What is the expected number of loops when only loops remain? (Hint - Use indicator random variables.) Express your answer as a summation. For full credit express your summation as a formula involving harmonic numbers. (This last step is worth 1 point.)

Problem 3: (20 points)

Suppose that you are the NH state campaign manager for a presidential candidate. The candidate has an all-day meeting with campaign officials in a room in your campaign office. The candidate would like to thank all of the volunteers who work in the campaign office personally, but wants to interrupt the meeting as infrequently as possible.

Each volunteer is assigned one shift on that day to do phone-banking, assemble mailings, organize canvassers, etc. The shifts are arranged around the volunteers’ jobs, classes, childcare, etc., so are quite irregular. You have \(n\) volunteers, and volunteer \(v_i\)’s shift begins at \(v_i.\text{start}\) and ends at \(v_i.\text{finish}\). The start and finish times are integers representing the number of minutes since midnight. The shift begins at the start of minute \(v_i.\text{start}\) and finishes at the end of minute \(v_i.\text{finish}\).
Describe a greedy algorithm that will pick a minimum number of times that the candidate can come out and spend a minute thanking the volunteers present at that time. The candidate must thank every volunteer. Thus if the volunteer shifts are expressed by the following \((start, finish)\) pairs:

\[
(400, 800), (500, 560), (600, 900), (700, 800)
\]

then the candidate could come out at times 525 and 750, and every volunteer would be thanked. (The first would be thanked twice, but that is OK.) There is no single time when all of the volunteers are present, so this selection is optimal.

Prove that the set of times that your algorithm selects covers every volunteer and is the smallest set of times that does this. Explain how to implement your algorithm as efficiently as you can and analyze its run time.

Problem 4: (24 points)

Again, suppose that you are the NH state campaign manager for a presidential candidate. Your candidate plans to fly into Manchester Airport early in the morning, spend the day going to campaign events, and then fly out of the Manchester airport that night. Your job is to make a schedule of campaign events for your candidate to attend.

Local organizers have proposed \(n\) possible events. An event \(e\) has a location \(e.location\), a start time \(e.start\), a finish time \(e.finish\), and an estimate of the number of people who would attend that event \(e.num\). The start and finish times are integers representing the number of minutes since midnight. Assume that you also have two additional events \(arrival\) and \(departure\). Both of the extra events have “Manchester Airport” as their location name and 0 as the expected number of people attending. \(arrival.finish\) is the time that the candidate leaves the airport and \(departure.start\) is the time that the candidate needs to be back at the airport.

Unfortunately you probably cannot schedule all of these events. If two events overlap you clearly can choose at most one of them. But what complicates things further is that your candidate will need time to travel between events. You are given a function \(travelTime(a, b, t)\) which returns the travel time from location \(a\) to location \(b\) for a trip starting at time \(t\). (It probably uses Google Maps.) An event \(e_1\) can be followed by an event \(e_2\) only if

\[
e_2.start - e_1.finish \geq travelTime(e_1.name, e_2.name, e_1.finish)
\]

This is the only restriction on scheduling events.

a) (20 points) Describe a dynamic programming algorithm to compute a valid schedule of events that starts with the \(arrival\) event, ends with the \(departure\) event, and maximizes the total number of people expected to attend those events. Give a recursive definition of a function that will allow you to solve this problem, describe a table that will let you efficiently compute this function, and explain the order to fill the table. Also explain how you can construct a schedule that achieves the maximum total expected attendance. Justify your algorithm and give its run time.

b) (4 points) Suppose that there could be multiple proposed events at a given location (at different times), but you are allowed to choose at most one proposal for any location. How could this violate optimal substructure?
Problem 5: (20 points)
This problem considers finding a local maximum in one and two dimensional arrays. We assume that all entries in the array are different. A local maximum is a value which is larger than all of its neighbors (two neighbors in a one-dimensional array, four neighbors in a two-dimensional array). It is easy to solve this problem in time proportional to the number of array elements in either case, but it is possible to do much better using divide and conquer.

a) (5 points) Describe a divide and conquer algorithm that will find a local maximum in a one-dimensional array $A[1..n]$ with $n$ elements in $O(\log n)$ time. Explain why your algorithm works and runs in the given time. To avoid boundary cases assume that $A[0] = A[n + 1] = -\infty$. (Hint - if the middle element is not a local maximum, why isn’t it? In this case can you reduce the problem to one of about half the size? Note that the original interval contains a value larger than the values at its boundaries $A[0]$ and $A[n + 1]$. Can you maintain this property in recursive calls?)

b) (15 points) Describe a divide and conquer algorithm that will find a local maximum in an $n \times n$ array $A[1..n, 1..n]$ in $O(n)$ time. The four neighbors of $A[i, j]$ are $A[i - 1, j]$, $A[i + 1, j]$, $A[i, j - 1]$, and $A[i, j + 1]$. To avoid boundary cases assume that any array entry with a subscript out of bounds has value $-\infty$, so that every array element has 4 neighbors.

Problem 6: (Extra Credit) (35 points)
You are hired as a summer intern at Intel, and your job is to test a new chip design to find at what voltage it burns out. The chips will work correctly for all voltages less than or equal to some $v$ but will fail for all higher voltages. The voltages that you are to test are $1, 2, \ldots, n$. Your job is to find a voltage $v$ such that the chips work correctly at voltage $v$ but burn out at voltage $v + 1$ (or to determine that the chips work correctly at the maximum voltage $n$). The process to test if the chip burns out at a given voltage is rather lengthy, so you want to perform as few tests as possible.

You have learned the best solution to this problem: binary search. There is one minor difficulty. Very few of the chips have been fabricated. If a chip passes the test at some voltage $v$ it can be tested again at a different voltage. However, if it fails to pass a test it has burned out and must be thrown away. A new chip must be used for subsequent tests. If you use binary search you may burn out all of the available chips before you determine the maximum voltage $v$ at which the chips function correctly.

a) (5 points) If you have only one chip, what is the only possible testing strategy? Why can any other strategy fail? How many tests does your strategy perform in the worst case?

b) (15 points) Describe a testing strategy for two chips that minimizes the number of tests performed. How many tests does your strategy perform in the worst case? To get credit for this part you must come up with a strategy that performs $o(n)$ tests.

c) (15 points) Describe and analyze a strategy for $k$ chips that minimizes the number of tests performed.