University of New Mexico
Department of Computer Science

Final Examination

CS 561 Data Structures and Algorithms
Fall, 2007

Name:	
Email:	

- Print your name and email, *neatly* in the space provided above; print your name at the upper right corner of *every* page. Please print legibly.
- This is an *closed book* exam. You are permitted to use *only* two pages of "cheat sheets" that you have brought to the exam and a calculator. *Nothing else is permitted.*
- Do all the problems in this booklet. *Show your work!* You will not get partial credit if we cannot figure out how you arrived at your answer.
- Write your answers in the space provided for the corresponding problem. Let us know if you need more paper.
- Don't spend too much time on any single problem. The questions are weighted equally. If you get stuck, move on to something else and come back later.
- If any question is unclear, ask us for clarification.

Question	Points	Score	Grader
1	20		
2	20		
3	20		
4	20		
5	20		
Total	100		

1. **Short Answer**

For each problem below, give the answer in terms of simplest Θ. Please show your work where appropriate. (2 points each).

(a) Worst case time to find the minimum element in a min-heap *Solution:* $\Theta(1)$

(b) Worst case time to find the maximum element in a min-heap *Solution:* $\Theta(n)$ - *The maximum element could be anywhere on the bottom level*

(c) Worst case cost of n calls to either Insert or Delete on the most efficient implementation of the dynamic table that we discussed in class *Solution:* $\Theta(n)$

(d) $\sum_{i=1}^{n} \log i$ *Solution:* $\Theta(n \log n)$

(e) Time to find a minimum spanning tree in a graph with n nodes and $\Theta(n^2)$ edges, using the fastest algorithm discussed in class
Solution: Prim's algorithm takes time $O(|E| + |V| \log |V|)$ *which is* $O(n^2)$ *in this case*

(f) Time to find if a cycle is reachable from some node v in a directed graph with n nodes and $O(n)$ edges.

Solution: $\Theta(n)$ *- this can be done with BFS starting at v and checking if there are any back edges.*

(g) Solution to the recurrence $T(n) = 2T(n/2) + \sqrt{n}$ *Solution:* $\Theta(n)$

(h) Expected number of empty bins if you randomly throw $2n$ balls into n bins. *Solution: This is* $n(1 - 1/n)^{2n} \leq ne^{-2}$ *which is* $\Theta(n)$

(i) What is the expected time to find the successor of some key (i.e. the node that comes right after the key) in a skiplist containing n items? *Solution:* $\Theta(\log n)$

(j) Solution to the recurrence $T(n) = 2T(n-1) - T(n-2) + n$ (assume all constants in the solution are greater than 0) *Solution:* $\Theta(n^3)$

2. **Short Answer (10 points each) Where appropriate, circle your final answer.**

   (a) Professor Plum postulates that if every edge on an undirected graph has a unique positive weight, then the shortest path tree rooted at v on that graph is always the same as the minimum spanning tree found by Prim's algorithm when seeded initially with the vertex v. Is this correct? If so, prove it. If not, give a counter example.

   Solution: No. Consider the graph over vertices v, w, x where edge (v, w) has weight 1, edge (v, x) has weight 2.5 and edge (w, x) has weight 2. The shortest path tree for v is edges (v, w) and (v, x). The MST found by Prim's when started at v is (v, x) and (w, x).

(b) Consider a data structure over an initially empty list that supports the following two operations. APPEND-NUMBER(x): Adds the number x to the beginning of the list; and MIN-MAX: Traverses the list, computing the minimum and maximum element in the list, and then creates a new list that contains only two numbers, the minimum and maximum of the old list.

- Assume an arbitrary sequence of n operations are performed on this data structure. What is the worst case run time of any particular operation? *Solution:* $\Theta(n)$. *First* $n-1$ *APPEND-NUMBER operations and then a MIN-MAX operation.*

- Show that the amortized cost of an operation is $O(1)$ using the potential method. Make sure to prove your potential function is valid. *Solution: Let* $\phi(D)$ *equal the number of items in the list. This is a valid potential function (Why?). The amortized cost of an APPEND-NUMBER operation is then* $c_i + \phi_i - \phi_{i-1} = 2$. *The amortized cost of a MIN-MAX operation is* $l_i + (2 - l_i) = 2$ *where* l_i *is the length of the list at time* i.

3. **Segmentation**

In the segmentation problem, you want to segment text that is written without spaces into individual words. For example, if you are given the text "meetateight", the best segmentation is "meet at eight", not "me et at eight" or "meet ate ight". Assume you are given as a black box a function q that takes a string of letters x and returns $q(x)$, which gives a measure of how likely x is to be an English word. The quality of x can be positive or negative so that $q("me")$ is positive, and $q("ight")$ is negative. Given a string s, a segmentation of s is a partition of the letters into contiguous blocks. The total quality of a segmentation is the sum of the quality of each of the blocks of letters. So the quality of our segmentation above is $q("meet") + q("at") + q("eight")$. Give an efficient algorithm that takes as input a string s of length n and returns a segmentation of maximum total quality (assume that a single call to the q function takes constant time). Analyze your algorithm.

Solution: We will show only how to find the quality value of the best segmentation. Finding the segmentation itself is straightforward once you know how to compute the optimal quality (you simple need to use another array to keep track of which arguments are used to achieve the maximum values in the following). We will define $m(i)$ to be the maximum quality of a segmentation of $s[1..i]$. Note that $m(1) = q(s[1])$ and that $m(j) = \max_{j'<j} m(j') + q(s[j'+1..j])$. We can easily create a dynamic program based on this recurrence by create an array m of size n, and filling it in from left to right using the above recurrence. The value returned is $m(n)$. This dynamic program will have two loops (an outer loop to range from j values from 2 to n an inner loop to range over j' values from 1 to $j'-1$.) and so will have running time $O(n^2)$.

4. **Goods Trading**

Assume there are n goods $g_1, g_2, ..., g_n$ and there is an n by n table T such that one unit of good g_i buys $T[i, j]$ units of good g_j. Part 1: Give an efficient algorithm to determine whether or not there is a sequence of goods $g_{i1}, g_{i2}, ..., g_{ik}$ such that $T[g_{i1}, g_{i2}] * T[g_{i2}, g_{i3}] * ... * T[g_{ik}, g_{i1}] > 1$. In other words, give an algorithm to determine if there is a sequence of goods that can be traded to actually obtain more of some good. Analyze your algorithm.

Solution: Note that $T[g_{i1}, g_{i2}]*T[g_{i2}, g_{i3}]*...*T[g_{ik}, g_{i1}] > 1$ *is true iff* $1/T[g_{i1}, g_{i2}]*1/T[g_{i2}, g_{i3}]*...*1/T[g_{ik}, g_{i1}] < 1$*. Taking logs of both sides, we can express this inequality as* $-logT[g_{i1}, g_{i2}] - logT[g_{i2}, g_{i3}] - ... - logT[g_{ik}, g_{i1}] < 0$*. Thus we can create a graph over* n *vertices, one for each good. And for every pair of goods* g_i *and* g_j*, we can create an edge with weight* $-logT[g_i, g_j]$*. There is a negative cycle in this graph iff there the desired sequence of goods exists. We can use any of our standard algorithms for finding negative cycles since the graph is strongly connected. If we use Bellman-Ford, our algorithm takes* $O(n^3)$ *time.*

Part 2: Give an efficient algorithm to print out such a sequence of goods if one exists. Analyze your algorithm.

Solution: In the last stage of Bellman-Ford, if there is some edge that is tense,we can just select one such node, (v_1, v_2) to be the first edge in the cycle. We run Bellman-Ford another step to find the next edge, (v_2, v_3) that points out of v_2 that is tense. We continue this process until we find all the edges in a cycle. This will require running Bellman-Ford for at most n additional steps so it still has a run time of $O(n^3)$.

5. **Bad Santa**

A Bad Santa has hidden $n/2$ Nintendo Wii's in n boxes. A child is presented with each box in sequence and must decide to either open that box immediately or to pass on the box and never be able to open it again. The child wants to guarantee she get at least one Wii while opening the smallest number of boxes in expectation. The Bad Santa knows the child's algorithm and places the Wii's in boxes so as to try to maximize the expected number of boxes opened. In this problem, you must design and analyze an algorithm for the child that 1) guarantees that the child finds a Wii; and 2) minimizes the expected boxes opened until that Wii is found.

Hint 1: Birthday paradox; Hint 2: You may again find the following inequality useful $1 - x \leq e^{-x}$

A final note, not necessary for solving the problem: this problem has applications in designing power-efficient sensor networks (the boxes are time steps, opening a box means being awake for a time step, and the presents represent time steps when important data is broadcast)

Solution: Surprisingly, you can do much better than opening $\Theta(n)$ boxes in expectation - You can open $O(\sqrt{n \ln n})$ boxes in expectation. The trick to this problem is the birthday paradox. Recall that when we talked about the b-day paradox, we noticed that if you throw sqrt(n) balls into n bins, that you expect at least two balls to fall into one bin. In this problem, such an event is equivalent to the girl opening a box that has a hidden present in it in the first half, provided that there are at least sqrt(n) presents in the first half. In particular, the $\sqrt(n)$ boxes with prizes in them correspond to special bins. We're throwing $\sqrt(n)$ balls into the $n/2$ total bins in the first half. The probability that a fixed ball falls in a fixed special bin is 1/n. The total number of combinations of balls and special bins is $\sqrt(n) * \sqrt(n) = n$, and so by linearity of expectation, the expected number of special bins with a ball in them is 1. So if there are $\sqrt(n)$ presents in the first half, and we open $\sqrt(n)$ boxes randomly, we would expect to find at least one prize.
The trick is to then realize that if we open up slightly more than $\sqrt(n)$ boxes, i.e. if we open $\sqrt(n) \log n$ boxes then we can turn this expected event into one that will almost certainly occur. Note that anything that is a bit bigger asymptotically than sqrt(n) works fine (e.g. $n^{1/2+\epsilon}$ for any $\epsilon > 0$), and is good for partial credit, it's just that $\sqrt(n) \log n$ is nearly the smallest you can go.
The entire solution is then as follows. Choose $\sqrt{n \ln n}$ boxes from the first $n/2$ boxes uniformly at random with replacement, then choose all of the last $n/2$ boxes. Open each chosen box in turn until a present is found. This algorithm opens no more than $\sqrt{n \ln n}$ boxes in expectation as we will now show by a case based analysis. Case 1: At least $\sqrt{n}$ of the first $n/2$ boxes are full. In this case, the probability that the algorithm does not find a present in the first $n/2$ boxes is no more than $(1 - 1/\sqrt{n})^{\sqrt{n \ln n}} \leq e^{-\sqrt{\ln n}} \leq 1/\sqrt{n}$. Thus the expected number of boxes the algorithm opens is no more than $\sqrt{n \ln n} + (1/\sqrt{n}) * (n/2) = O(\sqrt{n \ln n})$. Case 2: Less than $\sqrt{n}$ of the first $n/2$ boxes are full. In this case, the algorithm opens at most $\sqrt{n \ln n}$ boxes in the first half and at most $\sqrt{n}$ boxes in the second half (since at most $\sqrt{n}$ boxes in the second half can be empty. Thus in this case also the expected number of boxes opened is $O(\sqrt{n \ln n})$