University of New Mexico
Department of Computer Science

Final Examination

CS 561 Data Structures and Algorithms
Fall, 2009

Name:
Email:

- "Nothing is true. All is permitted" - Friedrich Nietzsche. Well, not exactly. *You are not permitted to discuss this exam with any other person.* If you do so, you will surely be smitten. You may consult any *other* sources including books, papers, web pages, computational devices, animal entrails, seraphim, cherubim, etc. in your quest for truth and solutions. Please acknowledge your sources.
- *Show your work!* You will not get full credit if we cannot figure out how you arrived at your answer. A numerical solution obtained via a computer program is unlikely to get much credit, if any, without a correct mathematical derivation.
- Write your solution in the space provided for the corresponding problem.
- If any question is unclear, ask for clarification.

Question	Points	Score	Grader
1	10		
2	20		
3	20		
4	20		
5	20		
Total	100		

1. **Dominos**

   Imagine we change the domino problem we discussed in class as follows. As before, we have a 2 by n rectangle and we want to count the number of ways we can tile it with dominos that are of dimension 2 by 1. However, now there are two colors of dominos: red and black. A red domino must 1) be placed vertically; and 2) be adjacent to a vertically placed black domino. There are no constraints on the black dominos. How many ways can we now tile the rectangle? Show your work.[1] Give an exact solution.

   Solution: Let $f(n)$ be the number of ways to tile a 2 by n rectangle with dominos. The new recurrence is: $f(1) = 1$, $f(2) = 3$ and $f(n) = f(n-1) + 2f(n-2)$. $\boldsymbol{L}^2 - \boldsymbol{L} - 2 = (\boldsymbol{L}+1)(\boldsymbol{L}-2)$ annihilates this recurrence. The general form of the solution is thus $c_1 2^n + c_2(-1)^n$. Solving for the constants based on the initial conditions, we get that the specific solution is: $f(n) = (2/3) * 2^n + (1/3)(-1)^n$.

[1] Two tilings are identical if both the colors and orientation of dominos is identical in both tilings.

2. **Quicksort**

You are working on the problem of developing a new randomized PARTITION [2] algorithm for quicksort (Section 7.1 in the text). In this problem, you will consider how improving the effectiveness of PARTITION might change the runtime of quicksort. Recall that the PARTITION function splits an array into two subarrays, A_1 and A_2, such that any element in A_1 is less than or equal to any element in A_2.

- (5 points) Imagine that you create a randomized partition function with the following properties: 1) its expected run time is $\theta(\sqrt{n})$; and 2) it always partitions into two subarrays that differ in size by at most 1 (i.e. each subarray is of size no more than $n/2$). If you use this new partition function in quicksort, what is the new expected run time?

  Solution: Expected runtime is given by the recurrence $T(n) = 2T(n/2) + \sqrt{n}$. By the Master's method, the solution to this is $T(n) = \theta(n)$.

- (20 points) Now imagine that your randomized partition function has the following properties: 1) its expected run time is $\theta(1)$; and 2) for i selected uniformly at random between 1 and $\sqrt{n}$, one subarray is of size no more than $n/2-i$ and the other subarray is of size no more than $n/2+i$. Show by induction that the expected runtime of quicksort is now linear. Hints: (1) This is a variation of Problem 7-2; (2) In your inductive proof, you may find it easiest to show that the expected run time is no more than $k_1 n - k_2$) for constants k_1 and k_2 (see the "Subtleties" subsection in Chapter 4.1 for details).

  Solution: This is a variation on Problem 7-2. Let $f(n)$ be the expected run time of quicksort now on a list of size n. Then via linearity of expectation, we have

$$\begin{aligned} f(n) &\leq \sum_{q=-\sqrt{n}}^{\sqrt{n}} (1/2\sqrt{n})(f(n/2+q) + f(n/2-q) + c) \\ f(n) &\leq (1/\sqrt{n}) \sum_{q=-\sqrt{n}}^{\sqrt{n}} (f(n/2+q) + c) \end{aligned}$$

  Let's guess that $f(n) \leq k_1 n - k_2$ for some constants k_1,k_2 and then prove this by induction.
  Base Case: $f(1) = \theta(1) \leq k_1 n - k_2$ for k_1 sufficiently large compared to k_2
  I.H.: For all $j < n$, $f(j) \leq kj$
  I.S.:

$$\begin{aligned} f(n) &\leq (1/\sqrt{n}) \sum_{q=-\sqrt{n}}^{\sqrt{n}} (f(n/2+q) + c) \\ &\leq (1/\sqrt{n}) \sum_{q=-\sqrt{n}}^{\sqrt{n}} (k_1(n/2+q) - k_2 + c) \end{aligned}$$

[2] Not necessarily comparison based

$$
\begin{aligned}
&\leq (1/\sqrt{n}) \sum_{q=-\sqrt{n}}^{\sqrt{n}} (k_1(n/2+q) \\
&\leq (2\sqrt{n}+1)(1+k\sqrt{n}/2) + (k/\sqrt{n}) \sum_{q=0}^{\sqrt{n}} q \\
&\leq kn
\end{aligned}
$$

Where the second step follows by the inductive hypothesis; the third step follows for k_2 sufficiently large; and the last step holds for k sufficiently large.

to have created a (non-comparison based) algorithm that can

do the following when given a list. It returns an element selected uniformly at random from the set of all elements in the list that

3. **Poker**

Recall that a standard deck of 52 cards contains one card from each of 13 different ranks: 2 through 10, Jack, Queen, King and Ace; and 4 different suits: hearts, diamonds, clubs and spades. In the game of poker, a *royal flush* consists of 5 cards: 10, Jack, Queen, King, and Ace that are all of the same suit. Thus, there are 4 possible royal flushes that can be formed from a deck of 52 cards. The probability that a randomly dealt hand of 5 cards is a royal flush is thus $4/\binom{52}{5}$.

(a) (8 points) If a dealer turns over n cards, for some $5 \le n \le 52$, from a well-shuffled deck of 52 cards, what is the expected number of royal flushes that can be formed from those n cards? Hints: 1) Use linearity of expectation; 2) Check that your answer makes sense for boundary values of n.

Solution: For a fixed subset S of 5 cards, let X_S be a random variable that is 1 if S forms a royal flush and 0 otherwise. Then $E(X_S) = \frac{4}{\binom{52}{5}}$. Let X be the total number of royal flushes that can be formed out of the n cards. Note that X is the sum over all $\binom{n}{5}$ sets S of X_S, thus $E(X) = \binom{n}{5}4/\binom{52}{5}$. This makes sense since when $n = 52$, the number is 4 and when $n = 5$, the number is just the probability of a hand of 5 cards being a royal flush.

(b) (2 points) How many cards need to be turned over before you would expect to be able to form at least 1 royal flush? Hint: Google has a built in calculator that does some combinatorics - try it out by googling "x choose y" for integers x and y.

Solution: Plugging in values of n *from the answer above, we see that* $n = 40$ *is necessary to make the expectation greater than* 1.

4. **Minimum Spanning Trees**

What if you're given a graph where the weight of each edge is distributed uniformly at random between ℓ and u for some real numbers ℓ and u? Describe an efficient *deterministic* algorithm to solve this problem. Please analyze your algorithm - note that you can analyze the expected run time of your algorithm since the input is randomized.

Solution: Use bucket sort and Kruskall's algorithm to get $O(m \log^* m)$

5. **Amortized Analysis**

(a) Your boss asks you to create a data structure that supports the following operations: Insert(x) and Delete-Top-Half(). The first function inserts a single number x, and the second functions deletes (and returns) all elements in the data structure that have value greater than or equal to the median. Assume your data structure must be comparison based i.e. numbers are compared via $\leq$, $<$, $>$, $\geq$ only). What is the best amortized complexity you can get for these operations? Give both upper ($O()$) and lowerbounds ($\Omega()$) and justify your answers. *Solution: Possible to get* $O(1)$ *- can show this via the accounting method.*

(b) Now Imagine your data structure must support the following operations: Insert(x) and Delete-Max(). The Insert function is as before, but the Delete-Max function deletes and returns the largest element in your data structure. What is the best amortized complexity you can now get for these operations? As before give both upper and lower bounds and justify your answer. *Solution: Can get* $O(\log n)$ *using a heap. Can't do better than* $\Omega(\log n)$ *since this would allow comparison based sorting in better than* $\theta(n \log n)$

(c) Now Imagine your data structure must support the following operations: Insert(x) and Delete-Deciles(d). The Insert function is as before, but the Delete-Deciles(d) function deletes and returns all elements greater than or equal to the $d/10 * n$-th largest element, where n is the number of items in the data structure and d is an integer between 1 and 9. What is the best amortized complexity you can now get for these operations? As before give both upper and lower bounds and justify your answer.

Solution: Can do this in $O(1)$ *amortized time. Can use the accounting method to show this. Charge \$11 for an insertion and store the remaining money on the element. We know that each call to Delete-Deciles deletes at least* $n/10$ *elements. The money stored on these elements is thus enough to pay for the call.*

6. **Sudoku**

   Network flow type problem for Sudoku.

7. **Let's make a Linear Program**

You are back on the game show circuit, this time on the 70's British game show *Mod*. You are now playing the following game against an opponent. At the beginning of each round, you and your opponent secretly choose an integer between 0 and 2. At the end of the round, you both reveal your numbers simultaneously and you win $x + y(\text{mod c})$ dollars where x is your number and y is your opponents number and c will be either 2 or 4 depending on the game variant. You goal is to determine a probability distribution over your 3 choices that will maximize your winnings, *given that your opponent knows your strategy and plays optimally for that strategy*! In other words, you want to choose probabilities p_i for each number i such that your winnings are maximized even if your opponent knows your p_i values.

(a) (10 points) Assume $c = 4$. Write down a linear program to find your optimal strategy. Hint: Use a trick similar to the one that we used in designing the linear program for the shortest paths problem.

Solution: *Maximize w*
Subject to:

$$w \leq 1p_1 + 2p_2$$

$$w \leq 1p_0 + 2p_1 + 3p_2$$

$$w \leq 2p_0 + 3p_1$$

$$\textit{For all } 0 \leq i \leq 2, 0 \leq p_i \leq 1$$

$$\sum_{i=0}^{2} p_i = 1$$

(b) (10 points)

Winnings for each outcome. Choose probabilities that maximize expected winnings.

8. **Monopoly**

   Find path of length L or less that gives the largest amount of profit for trades.

9. **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:* $\Theta(n^3)$

10. **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.

11. **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 $q(\text{"}me\text{"})$ is positive, and $q(\text{"}ight\text{"})$ 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 $q(\text{"}meet\text{"}) + q(\text{"}at\text{"}) + q(\text{"}eight\text{"})$. 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 $O(n^2)$.

12. **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* $O(n^3)$.

13. **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* $O(\sqrt{n \ln n})$