University of New Mexico
Department of Computer Science

Midterm 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 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. **Asymptotic Analysis and Recurrence Relations**

Is $n/2 \in o(n)$? Prove your answer using definitions of asymptotic notation given in the book and in class and solve for the values required to show the definitions hold or do not hold. *Solution: $n/2$ is not $o(n)$. To show that $n/2$ is $o(n)$, we would need to show that for any positive constant $c > 0$, there exists a constant n_0 such that $0 \leq n/2 \leq cn$ for all $n \geq n_0$. This is equivalent to saying that*

$$\begin{aligned} n/2 &\leq cn \\ 1/2 &\leq c \end{aligned}$$

But if $c > 1/2$, then the statement is not true for all $c > 0$.

Recurrence Relations: Consider the Recurrence $T(n) = 4T(n-1) - 4T(n-2) + 3^n$. Write down the general form of the solution for this recurrence (i.e. don't solve for the constants). *Solution: The homogeneous part is annihilated by $\boldsymbol{L}^2 - 4\boldsymbol{L} + 4$ which factors into $(\boldsymbol{L}-2)(\boldsymbol{L}-2)$. The nonhomogeneous part is annihilated by $(\boldsymbol{L}-3)$. Looking this up in the lookup table gives us that the general solution is of the form $T(n) = c_1 3^n + (c_2 + c_3 n)2^n$.*

2. **Heaps and Sorting**

Prove that you cannot build a Priority Queue in the comparison model (i.e. only comparison operations $\leq$, $\geq$, $=$, etc. are allowed on the keys) with both of the following properties:

- Extract-Min runs in $\Theta(1)$ time
- Build-Heap runs in $\Theta(n)$ time

Solution: If such a priority queue existed, we could use if to sort in $\Theta(n)$ *time as follows. First we would build a heap out of the elements to be sorted. Next we would run Extract-Min* n *times to return the elements in sorted order. This violates the lower-bound of* $\Theta(n \log n)$ *we proved for comparison-based sorting, so such a Priority Queue must not exist.*

Prove *succinctly* that the following algorithm correctly sorts a list of n elements by induction on n. Don't forget to include the base case, inductive hypothesis and inductive step.

```
GoofySort(A,i,j){
  if i+1 > j
       then return;
  Let s be the index of the minimum element in A[i..j];
  Exchange A[1] and A[s];
  Let b be the index of the maximum element in A[i..j];
  Exchange A[j] and A[b];
  GoofySort(A,i+1,j-1);
}
```

Solution: We will do induction on $n = j - i + 1$ *to show that SillySort sorts the array A[i..j]. B.C. if* $n = 1$*, there is only one element in the list and thus the list is already sorted. Hence SillySort is correct in this case by simply returning without doing anything. I.H. SillySort correctly sorts lists of size less than* n*. I.S. When faced with a list of size n, SillySort first moves the minimum element to the front of the list. It then moves the maximum element to the end. It then calls it self recursively on the remainder of the list. We can assume by the I.H., that the recursive call correctly sorts the remaining elements. Thus the entire array A[i..j] is in sorted order when the algorithm exits.*

3. **Search Trees**

   Consider a tree with the following properties:

   - Each internal node has exactly three children
   - The heights of the subtrees rooted at each child differ by at most 1.

   What is the maximum height of such a tree containing n nodes?

   Hint: Write a recurrence relation for the maximum number of nodes as a function of the height and then solve for the height. Show your work!

   Solution: Let $T(h)$ *be the maximum number of nodes in a tree of height* h*. Then* $T(h) = T(h-1) + 2T(h-2) + 1$*.* $(\boldsymbol{L}^2 - \boldsymbol{L} - 2) = (\boldsymbol{L} - 2)(\boldsymbol{L} + 1)$ *annihilates the homogeneous part and* $\boldsymbol{L} - 1$ *annihilates the non-homogeneous part. Thus, the solution is of the form* $T(h) = c_1 2^h + c_2 + c_3(-1)^h$*. Let* n *be the number of nodes in the tree. We know that* $n \geq T(h)$ *and so* $n \geq c_1 2^h + c_2 + c_3(-1)^h$*. If we let* $c = c_2 - c_3$*, we can say that,* $n \geq c_1 2^h + c$*. Taking logs of both sides, we have that* $\log n \geq h \log(2c_1) + \log c$*. This implies that* $h \leq 1/(log(2c_1))(\log n - \log c)$*. The right hand side is* $O(\log n)$*. Thus,* h *is* $O(\log n)$*.*

4. **Hash Tables and Probability**

   Assume we hash n items into a hash table with n bins using a good hash function i.e. each item is hashed to a bin chosen independently and uniformly at random. Give a good upper bound on the number of empty bins. Solve for the constants in your upper bound i.e. do not use asymptotic notation.

   Hint: Use the fact that $1 - x \leq e^{-x}$ for all x.

   Solution: Let X_i be an indicator random variable which is 1 *if the i-th bin is empty and is* 0 *otherwise. Then note that $E(X_i)$ equals the probability that the i-th bin is empty. This is the probability that the n items do not fall in bin i. The probability that a single item does not fall in bin i is exactly $1-1/n$. Since the items are all hashed independently, the probability that* no *item hashes into bin i is exactly $(1-1/n)^n$. Using the hint, note that $(1-1/n)^n \leq (e^{-1/n})^n = e^{-1}$. Let X be a random variable giving the number of empty bins and note that $X = \sum_{i=1}^{n} X_i$. Using linearity of expectation, we see that $E(X) = E(\sum_{i=1}^{n} X_i) = \sum_{i=1}^{n} E(X_i) \leq n/e$. Thus the expected expected number of empty bins is at most n/e. This is actually a pretty tight upper bound as n gets large.*

5. **Divide and Conquer**

   Imagine that after graduating from UNM, you start your new job at the exciting investment banking firm SELLOUT, Inc. The firm if faced with the following problem: they have an array of the predicted prices of a stock over n days and they want to determine, using this array, exactly one day to buy the stock and one day to sell the stock in order to maximize their profit.

   The problem can be formally stated as follows. You are given an array A of n numbers. You want to choose indices $1 \leq i < j \leq n$ such that $A[j] - A[i]$ is maximized over all such indices. Give an $o(n^2)$ algorithm to solve this problem.

   Solution: Use Recursion! Recursively find the pair of indices i_l and j_l on the left half of the array such that $i_l < j_j$ and $A[j] - A[i]$ is maximized over all such pairs. Find a similar pair i_r and j_r on the right half of the array. Now find x, the index of the element with smallest value on the left half and y, the index of the largest element on the right half. Finally, return max($A[j_l] - A[i_l]$, $A[j_r] - A[i_r]$, $A[y] - A[x]$). The run time of this algorithm is given by the same recurrence as for merge sort $T(n) = 2T(n/2) + n$, whose solution is $n \log n$. Note that we can get an even faster algorithm than this using dynamic programming. Also there is another $O(n \log n)$ solution that makes use of a heap or BST.