Department of Computer Science

Final Examination

CS 361 Data Structures and Algorithms
Spring, 2004

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 a *closed book* exam. You are permitted to use *only* a calculator and two pages of "cheat sheets" that you have brought to the exam. *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. If you get stuck, move on to something else and come back later.
- If any question is unclear, ask us for clarification.
- Good Luck!!!

Question	Points	Score	Grader
1	30		
2	20		
3	20		
4	20		
5	20		
6	30		
Total	140		

1. **Multiple Choice (30 points)**

Multiple Choice:

The following choices will be used in this multiple choice problem.

(a) $\Theta(1)$
(b) $\Theta(\log n)$
(c) $\Theta(\sqrt{n})$
(d) $\Theta(n)$
(e) $\Theta(n \log n)$
(f) $\Theta(n^2)$
(g) $\Theta(n^3)$
(h) $\Theta(2^n)$

For each of the questions below, choose one of the above possible answers. Please write the letter of your chosen answer to the left of the question.

(a) $\sum_{i=1}^{n} \log i$ *Solution:* $\Theta(n \log n)$
(b) $\sum_{i=1}^{n} i^2$ *Solution:* $\Theta(n^3)$
(c) $\sum_{i=1}^{n} 1/i$ *Solution:* $\Theta(\log n)$
(d) Time to print out the n elements in a binary search tree in sorted order *Solution:* $\Theta(n)$. *Just do an in-order traversal of the tree*
(e) Time to print out the n elements in a heap in sorted order *Solution:* $\Theta(n \log n)$. *Need to successively remove the minimum element.*
(f) Time to find the maximum element in a red-black tree with n nodes *Solution:* $\Theta(\log n)$
(g) Time to find the maximum element in a min heap with n nodes *Solution:* $\Theta(n)$. *The maximum element can be any of the* $\Theta(n)$ *leaf nodes*
(h) Expected number of nodes at the $(1/2) * \log n$ level of a skip list with n nodes *Solution:* $\Theta(\sqrt{n})$
(i) Worst Case runtime of randomized quicksort *Solution:* $\Theta(n^2)$
(j) Best Case runtime of bucket sort *Solution:* $\Theta(n)$
(k) Amount of extra space required by bucket sort (not counting the space to store the input array) *Solution:* $\Theta(n)$
(l) Expected lookup time for a hash table with n keys stored in n buckets with collision resolution by chaining *Solution:* $\Theta(1)$
(m) Worst case lookup time for a hash table with n keys stored in n buckets with collision resolution by linear probing *Solution:* $\Theta(n)$
(n) Imagine that we augment a skip list so that there is always a pointer to the leftmost element in the bottom list. What is the worst case time to find the minimum element in this augmented skip list? *Solution:* $\Theta(1)$. *The minimum is just the leftmost elem in the bottom list.*
(o) Amount of time to solve the string alignment problem for two strings each of size $\Theta(n)$ *Solution:* $\Theta(n^2)$

2. **Substitution Method (20 points)**

Consider the recurrence $T(n) = \frac{1}{n}(T(\lfloor n/2 \rfloor) * T(\lfloor n/2 \rfloor)) + (3/4)n$ where $T(1) = 1$.

Show that $T(n) \leq n$ by induction. Include the following in your proof: 1)the base case(s) 2)the inductive hypothesis and 3)the inductive step.

Solution: Base Case: $T(1) = 1$ *which is in fact no more than* 1.
Inductive Hypothesis: For all $1 \leq j < n$, $T(j) \leq n$
Inductive Step: We must show that $T(n) \leq n$, *assuming the inductive hypothesis.*

$$
\begin{aligned}
T(n) &= \frac{1}{n}(T(\lfloor n/2 \rfloor) * T(\lfloor n/2 \rfloor)) + (3/4)n && (1)\\
&\leq \frac{1}{n}((n/2) * (n/2)) + (3/4)n && (2)\\
&\leq (n/4) + (3/4)n && (3)\\
&= n && (4)
\end{aligned}
$$

where the inductive hypothesis allows us to make the replacements in the second step.

3. **Annihilators (20 points)**

Consider the recurrence $T(n) = 2T(n-1) - T(n-2) + 4$, $T(0) = 0$, $T(1) = 0$, $T(2) = 4$. First, find the general form of the solution for this recurrence using annihilators. Then solve for the constants in the general form using the initial conditions in order to get an exact solution.

Solution: Consider the homogeneous part first. Let $T_n = 2T(n-1) - T(n-2)$, *and* $T = \langle T_n \rangle$. *Then*

$$\begin{aligned} T &= \langle T_n \rangle & (5)\\ \boldsymbol{L}T &= \langle T_{n+1} \rangle & (6)\\ \boldsymbol{L}^2 T &= \langle T_{n+2} \rangle & (7) \end{aligned}$$

Since $\langle T_{n+2} \rangle = \langle 2T_{n+1} - T_n \rangle$, *we know that* $\boldsymbol{L}^2 T - 2\boldsymbol{L}T + T = \langle 0 \rangle$, *and thus* $\boldsymbol{L}^2 - 2\boldsymbol{L} + 1 = (\boldsymbol{L}-1)(\boldsymbol{L}-1)$ *annihilates* T. *Further we know that* $(\boldsymbol{L}-1)$ *annihilates the non-homogeneous part. Thus the annihilator of the whole sequence is* $(\boldsymbol{L}-1)^3$. *Thus* $T(n)$ *is of the form:*

$$T(n) = c_1 n^2 + c_2 n + c_3$$

We know:

$$\begin{aligned} T(0) = 0 &= c_3 & (8)\\ T(1) = 0 &= c_1 + c_2 & (9)\\ T(2) = 4 &= 4c_1 + 2c_2 & (10) \end{aligned}$$

so $c_1 = 2$, $c_2 = -2$, $c_3 = 0$ *and thus*

$$T(n) = 2n^2 - 2n$$

Check: $T(3) = 2 * 4 - 0 + 4 = 12$ *and* $2 * 9 - 6 = 12$.

4. **Sorting (20 points)**

Consider the following generalization of mergesort. This algorithm divides l into k equal size lists, recursively sorts each of these lists and then merges the k sorted lists. Note that k is just some arbitrary constant like 2 or 5 (If $k = 2$, this algorithm is just like regular mergesort.)

```
k-MergeSort(l){
  if(l.size()<=1){
    return l;
  } else{
    Divide l into k lists each of size n/k;
    Recursively k-Mergesort each of these k lists;
    Merge the k sorted lists;
  }
}
```

(a) Recall that we can merge k sorted lists (each of size n/k) together in time $n \log k$. What data structure do we use to do this?

Solution: A heap

(b) Let $T(n)$ be the run time of k-Mergesort on a list of size n. Write down a recurrence relation for $T(n)$.

Solution: $T(n) = kT(n/k) + n \log k$

(c) Now solve this recurrence relation. Hint: Use a recursion tree; recall that $\log_k n = \frac{\log n}{\log k}$. Based on your solution, which values of k make the algorithm fastest?

Solution: Every level of the recursion tree has weight $n \log k$. *There are* $\log_k n$ *levels of the recursion tree, so the total weight of the tree is* $(n \log k) * \log_k n = (n \log k) * \frac{\log n}{\log k} = n \log n$. *Hence all values of* k *are equally good.*

5. **Probability (20 points)**

Imagine that you are inserting the numbers 1 through n into a hash table with m buckets. Assume uniform hashing with chaining. For any i between 1 and $n-1$, we call the pair of numbers i, $i+1$ a *consecutive pair* (e.g. 3,4 is a consecutive pair but 3,5 is not). In this problem, you will compute the expected number of consecutive pairs that are hashed to the same bucket.

(a) Let X_i be an indicator random variable that is 1 if the consecutive pair i,$i+1$ are hashed to the same bucket and 0 otherwise. What is $P(X_i = 1)$? What is $E(X_i)$?

Solution: $P(X_i = 1) = E(X_i) = 1/m$

(b) Now let X be a random variable which is the total number of consecutive pairs that are hashed to the same bucket (i.e. the sum of all the X_i). Compute the expected value of X (hint: use linearity of expectation). *Solution:* $X = \sum_{i=1}^{n-1} X_i$. *So* $E(X) = E(\sum_{i=1}^{n-1} X_i) = \sum_{i=1}^{n-1} E(X_i) = \sum_{i=1}^{n-1} 1/m = (n-1)/m$.

(c) How large must n be before we would expect at least one consecutive pair to be hashed to the same bucket? *Solution: n would have to be $m+1$*

6. **Binary Search Trees (30 points)**

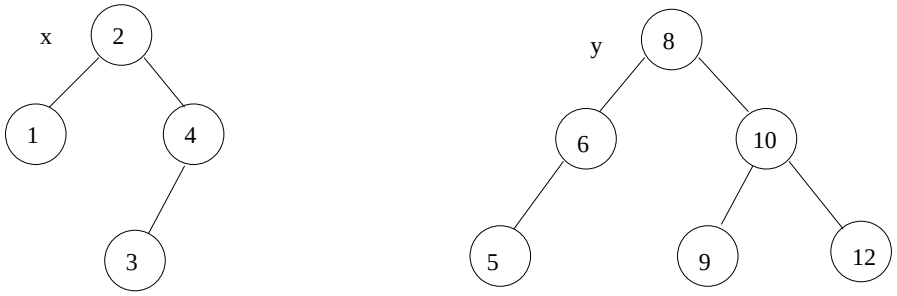
Recall the following code for finding the minimum element in a Binary Search Tree(BST) rooted at the node r.

```
Find-Min(r){
  while (r.left != NULL){
    r = r.left;
  }
  return r;
}
```

(a) State the *loop invariant* that you would use to prove that Find-Min does in fact return the minimum element in the BST.

Solution: The minimum key is in the subtree rooted at r

The following problems deal with merging two binary search trees. Imagine that you are given two BSTs, one rooted at a node x and another rooted at a node y. Assume that every key in the first BST is less than every key in the second BST (see the example below). Your want to merge these two input BSTs into a single BST which contains the union of the keys in the two input trees. *The merged tree must have height no more than one plus the maximum of the heights of the two input trees.* Example figure:



(b) Give an algorithm to merge the two BSTs that runs in time linear in the height of the trees. Hint: Look again at part a. *Solution: Let m be the minimum key in the BST rooted at y. Delete m from the BST rooted at y. Create a new tree with m as the root node and x as the left child and y as the right child. Return this tree.*

(c) Argue that your algorithm is correct *Solution: We will argue that the new tree has the BST property. Note that it's easy to see that all nodes underneath x and all nodes underneath y have the BST property. Consider the node m. We know that its key must be at least as large as all nodes under x by assumption. We further know that its key is no larger than all nodes under y since it is the minimum node in y.*