ICS 161 — Algorithms — Winter 1998 — First Midterm

Name:

ID:

1:

2:

3:

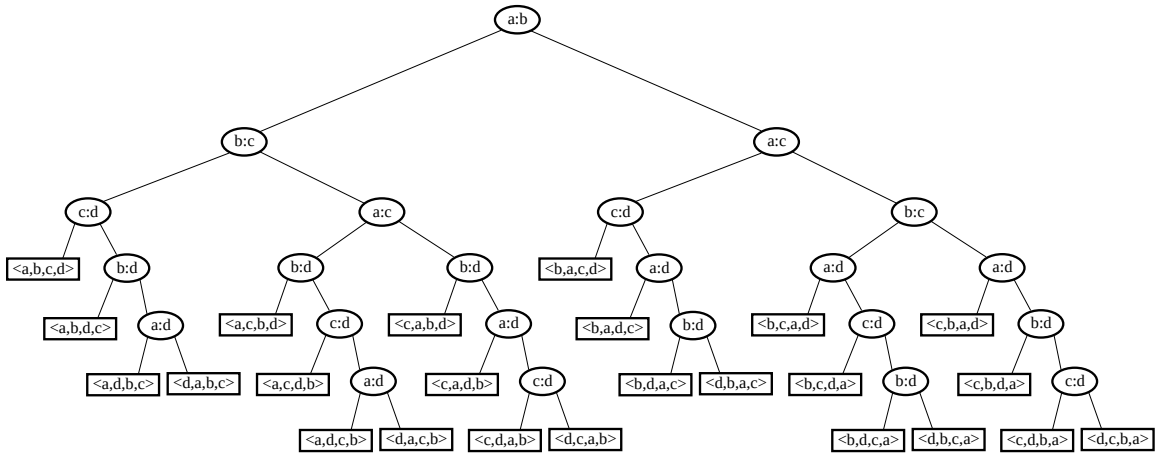
4:

5:

6:

total:

1. (15 points). The following comparison tree sorts the four values a, b, c, and d.



(a) What is the worst-case number of comparisons performed by the comparison tree?

(b) What is the best-case number of comparisons performed by the comparison tree?

(c) What is the average-case number of comparisons performed by the comparison tree, assuming that any permutation of the five inputs is equally likely?

2. (15 points). Recall that merge sort takes time $\Theta(n \log n)$ to sort n values, while if the values are positive integers, the largest of which is k, then radix sort takes time $\Theta(n \log k / \log n)$.

(a) If $k = \Theta(2^n)$, what is the time for radix sort (expressed as a function of n alone)? Which of these two algorithms would be faster?

(b) If $k = \Theta(2^{\log^3 n})$, what is the time for radix sort (expressed as a function of n alone)? Which of these two algorithms would be faster?

(c) If $k = \Theta(n^5)$, what is the time for radix sort (expressed as a function of n alone)? Which of these two algorithms would be faster?

3. (15 points). Write a recursive algorithm (C code or pseudocode) that computes the value of $T(n)$, where $T(n)$ is defined by the following recurrence. You do not need to analyze your algorithm.

$$T(n) = 1, \qquad \text{if } n \leq 2;$$

$$T(n) = T(n-1)T(n-2) + T(n-3) + 1, \qquad \text{otherwise.}$$

4. (15 points). Write a recurrence describing the number of times the following algorithm compares two members of array *A*, measured as a function of the array length *n*. (Do not solve this recurrence.)

```
fibsort(long A[], int n)
{
    if (n <= 1) return;
    if (A[0] > A[n-1]) swap A[0] and A[n-1];
    fibsort(A+1, n-2);
    if (A[0] > A[1]) swap A[0] and A[1];
    fibsort(A+1, n-1);
}
```

5. (20 points). State whether each of the following sorting algorithms is stable or not stable.

(a) Merge sort.

(b) Heapsort.

(c) Quicksort.

(d) Radix sort.

6. (20 points). Suppose we are given an array of numbers $A[0]$, $A[2]$, ..., $A[n-1]$. You may assume that n is a multiple of 6. We wish to find a number x that simultaneously satisfies two properties: First, x should be in the middle two-thirds of the array (in other words, $x = A[i]$ for some $n/6 \leq i < 5n/6$). Second, after sorting the array, x should still be in the middle two-thirds.

(a) Prove that such an x exists. (Hint: how many values are not in the middle two-thirds of the original array? How many are not in the middle two-thirds of the sorted array?)

(b) Describe a linear time algorithm for finding such an x. (Note, if your solution calls one of the algorithms described in the lectures, you should just refer to it by name, e.g. "quicksort", rather than explaining how it works.)