Midterm 1

ECE 608
February 26, 2004, 7-9pm **Name:** SOLUTION

Read all of the following information before starting the exam:

- **NOTE:** Unanswered questions are worth 30% credit, rounded down. Writing any answer loses this "free" credit, which must be earned back by the quality of the answer. If you wish a question to be treated as unanswered, but you have written on it, clearly write "DO NOT GRADE" in the answer area. In a multi-part question, unanswered *parts* are worth 30%.
- Show all work, clearly and in order, if you want to get full credit. I reserve the right to take off points if I cannot see how you arrived at your answer (even if your final answer is correct).
- No calculators, or materials other than pen/pencil and blank paper are allowed except those we distribute during the exam.
- Please keep your written answers brief; be clear and to the point. Points will be deducted for rambling and for incorrect or irrelevant statements.
- This test has 8 problems, each of which is approximately equal in value. Multi-part problems divide the problem score approximately equally among the parts.
- Good luck!

1. Clearly describe a worst-case $\Theta(n \lg n)$ algorithm for determining if an array containing n distinct natural numbers has any entries x and y such that $x = 2y$. Argue that your algorithm is indeed $\Theta(n \lg n)$ worst case.

There are n elements in the array, and because each is a distinct natural number there are no repeated keys in the array. Create a new array of size $2n$ and insert all elements of the original array into it. Now for each element y in the original array, insert key $x = 2y$ in the new array. Thus we have filled all $2n$ elements of the new array. MERGE-SORT the new array in $2n \log 2n$ steps. Now scan the array in $2n$ steps and look for any repeated keys. If a repeated key is found, then it must be the case that one of them must belongs to the original array and one is from doubled elements. Thus we can find if such an x and y from the original array exist for which $x = 2y$.

2. Specify three recurrences and solve them using the Master Theorem. Each recurrence must use a different case of the theorem.

(1)

$$T(n) = 4T(n/2) + n$$

We have $\log_a b = 2$, and $f(n) = n = O(n^2)$. So by case 1 of the master theorem:

$$T(n) = \Theta(n^2)$$

(2)

$$T(n) = 2T(n/2) + n$$

We have $\log_a b = 1$, and $f(n) = n = \Theta(n)$. So by case 2 of the master theorem:

$$T(n) = \Theta(n \log n)$$

(3)

$$T(n) = 2T(n/2) + n^2$$

We have $\log_a b = 1$, and $f(n) = n^2 = \Omega(n)$. Also $af(n/b) \leq cf(n)$ is true for $1/4 \leq c \leq 1$. So by case 3 of the master theorem:

$$T(n) = \Theta(n^2)$$

3.

a. Argue that no correct comparison sort can have a worst-case performance better than $O(n \lg n)$.

See text, Theorem 8.1 on page 167.

b. Give a tight asymptotic bound on the best-case performance a comparison sort can exhibit and explain.

Each element must be compared in order to determine its order relative to other elements. To argue that n comparisons are needed, we can note that n-2 of the elements (those not largest or smallest) must each appear in two different comparisons (one that they win and one that they lose). The remaining 2 elements each appear in at least one comparison. This gives a total of at least n-1 comparisons, which is in fact all that are needed in the best case.

4.

a. Give and prove asymptotic upper and lower bounds for $\sum_{k=1}^{n} k^4$.

$$\sum_{k=1}^{n} k^4 \leq \sum_{k=1}^{n} n^4 = O(n^5)$$

and

$$\sum_{k=1}^{n} k^4 = \sum_{k=1}^{n/2} k^4 + \sum_{k=n/2}^{n} k^4 \geq \sum_{k=1}^{n/2} 0 + \sum_{k=n/2}^{n} (n/2)^4 = \Omega(n^5)$$

Combining the above two observations:

$$\sum_{k=1}^{n} k^4 = \Theta(n^5)$$

b. Exploit the cyclic properties of the sin function to define a function $f(n)$ such that $f(n)$ is not $O(g(n))$ or $\Omega(g(n))$ for any polynomial function $g(n)$.

$$f(n) = 2^n \sin n$$

The exponential part of $f(n)$ cannot be bounded by any polynomial $g(n)$ so $f(n)$ is not $O(g(n))$. The sinusoidal part of $f(n)$ causes the expeonential magnitude to oscillate between positive and negative extremes. At the negative extreme $f(n)$ will be asymptotically smaller than any polynomial $g(n)$ so $f(n)$ is not $\Omega(g(n)))$ either.

This problem continued on the next page.

c. Give and prove asymptotic upper and lower bounds for $T(n) = 2T(n/2) + n/\lg n$.

See solution to homework #10 problem #3 (CLR 4-4 (e)).

5.

a. Describe where care must be taken in MERGESORT to ensure that it is a *stable* sort.

During the merge operation, when the two keys being compared from the two subarrays are equal we choose the key from the subarray that is lower indexed with respect to the original array. This ensures that keys with same values remain in the same relative order as they appeared in the original input array.

b. Clearly describe an *in-place* sorting algorithm that achieves an $O(n)$ worst-case sorting time by exploiting an assumption that keys are all natural numbers and only k distinct keys appear. Here, k is a fixed constant that does not grow with n.

Create a new array of size k. Scan the given array in n steps to find out the k different keys and store each one in the new k-sized array. Sort the k-sized array in $k \log k = \Theta(1)$ time. Say the elements of this sorted array are $k_1, k_2...k_k$. Choose k_1 as pivot and run PARTITION on the given array. The pointers i and j meet at a point where all lower elements in the array are k_1. Frtom this point choose k_2 as a new pivot and run PARTITION only on the remaining upper elements of the array. Once again the pointers i and j meet at a point where the k_1 and k_2 are the lower elements in sorted order. Repeat this process k times, each time forming a new pivot and partitioning only the remaining protion of the array. Each invocation of PARTITION takes n steps at most, and we run PARTITION k times. Thus the running time of the algorithm is $O(nk) = O(n)$ as k is a constant.

6. Suppose k balls are fired into n bins uniformly at random, independently. What is the expected number of bins that have just 1 ball? Explain.

See solution to homework #3 problem #18 (CLR 5.4-6). Use k balls instead of n balls.

7. Consider the code below for BUILDMAXHEAP. Give a loop invariant for the **for** loop and use it to prove the correctness of the algorithm.

```
BUILDMAXHEAP($A$)
1. $heap\text{-}size[A] \leftarrow \text{length}[A]$
2. for i$\leftarrow \lfloor \frac{\text{length}[A]}{2} \rfloor$ downto 1
3.      do MAXHEAPIFY($A, i$)
```

See text, page 133.

8. Consider the code below for BuildMaxHeap. Give and prove a tight asymptotic bound on the worst-case run-time for one call. Assume that length[A] is n.

```
BuildMaxHeap($A$)
1. heap-size[$A$] $\leftarrow$ length[$A$]
2. for i$\leftarrow$ $\lfloor \frac{\text{length}[A]}{2} \rfloor$ downto 1
3.       do MaxHeapify($A, i$)
```

See text, page 133-135.