CS 477/677 Analysis of Algorithms
Fall 2006 - Dr. George Bebis
Midterm Exam
Duration: 2:30 - 3:45

Student Name:

1. **[20 pts]** For each of the following statements, indicate whether it is true or false. To get credit, you must give brief reasons for your answer.

T F InsertionSort's running time is $\Theta(n^2)$

T F Both MergeSort and QuickSort have $\Theta(nlgn)$ running time.

T F The Master method can be used to solve the recurrence $T(n) = \sqrt{n}T(n/2) + n$, but not the recurrence $T(n) = 2T(n/\sqrt{n}) + n$.

T F Suppose that you write a program that frequently sorts arrays whose size varies between 5 and 10. You should do this using MergeSort instead of InsertionSort, since the running time of MergeSort is $\Theta(nlgn)$, while that of InsertionSort is $O(n^2)$

T F QuickSort's running time depends on whether the partitioning is balanced or unbalanced. If the partitioning is balanced, running time is $\Theta(nlgn)$, however, when the partitioning is unbalanced, the running time is $\Theta(n^2)$.

2. **[20 pts]** Prove the following:

(a) $2^{n-3} = \Omega(2^{n+1})$

(b) $n \lg n = O(n^{3/2})$

3. **[20 pts]** Solve the following recurrences:

(a) $T(n) = 7T(n/3) + n^2$

(b) $T(n) = 2T(n-2) + 2$

4. **[20 pts]** Consider sorting *n* numbers in array *A* by first finding the smallest element of *A* and putting it first. Then find the second smallest of *A* and put it second. Continue in this manner for the *n* elements of *A*. This algorithm is known as *SELECTION-SORT*; the pseudocode is shown below.

```
Alg.: SELECTION-SORT(A)
  n <-- length[A]
  for j <-- 1 to n - 1
      do smallest <-- j
          for i <-- j + 1 to n
                do if A[i] < A[smallest]
                     then smallest <-- i
       exchange A[j] <-- A[smallest]
```

(a) **[7.5 pts]** How many key comparisons does SELECTION-SORT do ? Justify your answer.

(b) **[7.5 pts]** What arrangement of keys is a worst case for SELECTION-SORT? What arrangement of keys is a best case ?

5. (**Undergraduate Students** only) **[20 pts]** Consider the following algorithm. Given an array L of n values, it places in $L[i]$ the sum of the elements from 1 to i.

```
PartSum(L,n)
  if (n==1) return;
  for (i=1; i<n; i++)
    L[n]=L[n]+L[i];
  PartSum(L,n-1);
```

(a) **[10 pts]** Find the recurrence that describes the running time of the above algorithm.

(b) **[10 pts]** Solve the recurrence describing the running time of the above algorithm.

5. (**Graduate Students** only) **[20 pts]** The following algorithm finds the maximum value in an array $A[1..n]$.

Maximum(A,p,r)
if r-p ≤ 1 then return (max(A[p],A[r]))
else
max1=Maximum(A,p,$\frac{(p+r)}{2}$)
max2=Maximum(A,$\frac{(p+r)}{2}$+1,r)
return(max(max1,max2))

(a) **[10 pts]** Find the recurrence that describes the running time of the above algorithm.

(b) **[10 pts]** Solve the recurrence describing the running time of the above algorithm.