Analysis of Algorithms **Comprehensive Examination, Fall 1999**

Sign the exam with your student number - not your name ________________________

Select three (3) of the following questions to answer. Do not answer more than three.

1. ($33\,\frac{1}{3}$ pts) Consider the following code.

1 ⟨*Distribution counting sort* 1⟩≡

```
public Record[] countingSort(Record[] r, int m) {
  int n = r.length;
  int[] count = new int[m];
  Record[] newR = new Record[n];

  for (int i = 0; i < m; i++)  { count[i] = 0; }
  for (int j = 1; j < n-1; j++) { ++count[r[j].key]; }
  for (int i = 1; i < m; i++) { count[i] += count[i-1]; }
  for (int j = n-1; j >= 0; j--) {
     newR[count[r[j].key]] = r[j];
     --count[r[j].key];
  }
  return newR;
}
```

(a) Analyze the time complexity of the algorithm. That is, show how to determine its time complexity.

(b) Analyze the space complexity of the algorithm. That is, explain the memory storage space requirements of the algorithm.

2. (33 $\frac{1}{3}$ pts) Devise a recurrence relation and initial conditions for the following code. Note to use this silly little program you would initially call it as `recur(words, 0, n-1)` where n is the length of the `words` array. (Do not attempt to solve the recurrence!)

2 $\langle$*A silly little program* 2$\rangle\equiv$

```
public void recur(String[] words, int first, int last) {
 if (first < last) {
     int middle =  (first+last)/2;
     recur(words, first, middle);
     recur(words, middle+1, last);
     for (int i = first; i < last; i++) {
       for (int j = last; j > i; j--) {
         swap (word[j], word[i]);
       }
     }
  }
}
```

3. ($33\frac{1}{3}$ pts) Pretend you found a new comparison based sorting algorithm which had time complexity determined by the recurrence relation and initial conditions

$$T(n) = 3T(n/2) + n, \quad T(1) = 1.$$

(a) Solve the above recurrence, and give the big-O time complexity of your new sorting algorithm.

(b) Based on what you know about sorting algorithms, how would you classify yours: too fast, i.e., faster than theory allows, fast, somewhere between fast and slow, slow, or too slow.

4. ($33\frac{1}{3}$ pts) Asymptotic notation is used frequently in the analysis of algorithms.

(a) Provide a precise definition of the statement: $f(n) = O(g(n))$.

(b) Provide a precise definition of the statement: $f(n) = \Omega(g(n))$.

(c) Provide a precise definition of the statement: $f(n) = \Theta(g(n))$.

(d) Explain the situations where one uses O, Ω and Θ notation, e.g., if you say an algorithm has time complexity $O(g(n))$, $\Omega(g(n))$, or $\Theta(g(n))$, what is being expressed about the algorithm?