Analysis of Algorithms **Comprehensive Examination, Spring 2000**

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

Answer all three questions to the best of your ability.

1. (50 pts) Consider the following code that uses the nearest neighbor heuristic to find a cycle in a completely connected weighted graph. Think of the nodes of the graph as a collection of cities separated by roads (edges) at given distances. For notation, let $m =$ `MAXCITIES` be the maximum number of notes in any graph and let $n =$ `numOfCities` be the number of cities in a particular instance.

1 ⟨*Nearest Neighbor Tour* 1⟩≡

```
void NearNeighborTour() {
1    int thisCity, int closestCity;
2    bool visited[MAXCITIES];
3    double closestDistance,
4
5    /* initialize unvisited cities */
6    for (int i = 0; i < numberOfCities; i++) { visited[i] = FALSE; }
7
8    /* choose numberOfCities as starting city */
9    thisCity = numberOfCities - 1;
10   visited[thisCity] = TRUE;
11
12   for (int i = 1; i < numberOfCities; i++) {
13     closestDistance = MAXDIST;
14     for (int j = 0; j < numberOfCities; j++) {
15       if (!visited[j] && distance(thisCity, j) < closestCistance) {
16           closestCistance = distance(thisCity, j);
17           closestCity = j;
18       }
19     }
20     printf("Move from %d to %d\n", thisCity, closestCity);
21     visited[closestCity] = TRUE;
22     thisCity = closestCity;
23   }
24   printf("Move from %d to %d\n", thisCity, numberOfCities - 1);
25 }
```

(a) (25 pts) What is the worst case complexity of the code in lines 12–23 (assuming the `distance()` and `printf()` functions have constant running time)?

(b) (15 pts) What is the (local) space complexity of the the `NearNeighborTour()` function.

(c) (10 pts) A little probability and average execution.

i. For each $i = 1$ to $n - 1$ and for each $j = 0$ to $n - 2$, what is the probability that `!visited[j] == TRUE` in line 15? That is find the probability

$$P(\texttt{!visited[j],i)}) \quad \forall i = 1, \ldots, n-1, j = 0, \ldots, n-1.$$

ii. Pretend that the distances between cities are uniformly distributed with

$$P(\texttt{distance(thisCity, j)} < \texttt{closestCity}) = \frac{1}{n}.$$

also pretend that the events described in line 15 are independent of one another. What is the average (expected) number of times line 16 will execute?

2. (50 pts) Below is a recursive algorithm, `maxSum()` for you to analyze. It solves the problem:

> Given a vector `V` of real numbers, find the maximum sum in any *contiguous* sub-vector, or zero when all elements are negative.

For example, given the vector

31	-41	59	26	-53	58	97	-93	-23	84

the algorithm will return 187 corresponding to the sum of 59, 26, -53, 58, and 97. The algorithm would be originally invoked as

`answer = maxSum(v, 0, n-1)`, where `n` is the length of `v`.

(Note `max()` is a simple constant time macro that returns the largest of the numbers passed to it. Also you needn't understand the algorithm to analyze it.)

3 ⟨*Maximum sum in contiguous elements* 3⟩≡

```
public double maxSum(double v[], int lo, int hi) {
  if (lo > hi) { return 0.0; /* empty vector */ }
  if (lo == hi) { return max (0.0, v[lo]); /* one element vector */ }
  int mid = (lo+hi)/2;
  double sum = 0.0; double maxToLeft = 0.0;
  for (int i = lo; i < mid; i++) {
     sum += v[i];
     maxToLeft = max(maxToLeft, sum);
  }
  sum = 0.0; double maxToRight = 0.0;
  for (int i = mid; i < hi; i++) {
     sum += v[i];
     maxToRight = max(maxToRight, sum);
  }
  double max1 = maxSum(lo, mid);
  double max2 = maxSum(mid+1, hi);
  return max(maxToLeft + maxToRight, max1, max2);
}
```

(a) (30 pts) Find a recurrence equation with initial condition that describes the time complexity of `maxSum()`.

(b) (20 pts) Solve the recurrence you found in question 2a.