CMPT 307
Fall 2020
T. Shermer

Midterm

100 points. 50 minutes. You are allowed to look at your notes, books, and the lecture slides. You are not allowed to search or roam the web.

1. (30 points; 6 each)
   a. Define *shortest-path distance* in a weighted graph.
   b. Which is asymptotically larger, $n^2 \log n$ or $n^{2.01}$?
   c. Express $2^{\log n} + n^{1.5} + n \log \log n$ in O-notation.
   d. Express $T(n) = 18T(n/3) + cn^3$ in O-notation.
   e. Define a *minimum spanning tree*.

2. (20 points) The following is the routine SEENTREESE, a non-critical component of something-or-other. Derive a recurrence for the time complexity of SEENTREESE, and express that time complexity in Θ-notation. Do not worry about floors and ceilings.

```
int SEENTRESE(A, left, right)
     if (right - left < 4) {
          return 1;
     }
     int sum = 0;
     for(int i=left; i<right; i++) {
          sum = sum + A[i];
          A[i] = A[i] + 1;
     }

     int mid =  (left + right) / 2;
     int fquart = (left + mid) / 2;
     int tquart = (mid + right) / 2;

     gormA = SEENTRESE(A, mid, right);
     gormB = SEENTRESE(A, left, mid);
     gormC = SEENTRESE(A, fquart, tquart);
     gormD = SEENTRESE(A, mid, right);

     int max = max(gormA, gormB, gormC, gormD);
     return sum + max;
```

3. (25 points) Suppose you are given a weighted graph and you must solve the Longest Path Problem. This problem is to find a sequence of vertices $v_0, v_1, \ldots v_k$ such that the sum of the weights on the edges $(v_0, v_1), (v_1, v_2), \ldots(v_{k-1}, v_k)$ is maximum, under the constraint that no vertex appears twice in $v_0, v_1, \ldots v_k$.

Your friend Zimdor proposes a greedy algorithm, maintaining a set S of chosen edges and T of remaining edges. At each step, it adds the largest-weight edge of T to S, removing it from T. Then it removes all edges from T that cannot be used in a longest path that includes all of S.

If this technique is correct, prove it. If not, disprove it.

4. (25 points) Suppose we are given a collection of *n* cubes $c_1, c_2, \ldots c_n$ where each cube c_i contains a priority list $c_i.priority$ of other cubes. We want to have a data structure that always gives us a cube that has an "empty" priority list: every cube originally on the list has been given out previously. (We erase a cube from all the priority lists still in the data structure when we give it out.) We are guaranteed by the person giving us the cubes that there is a way to do this and eventually get all of the cubes from the data structure.

Let s be the sum of the sizes of the priority lists. By using a priority queue, where the key for each cube is the number of other cubes on its priority list, Zimdor can solve this problem in (n+s)log n time. The removeMin operation on the priority queue gives us the cubes.

If you can solve the problem faster, show how (pseudocode is accepted but not necessary). If not, show the details that Zimdor's algorithm must include (pseudocode mandatory). Showing Zimdor's algorithm is worth 18 marks if there is a faster solution, and 25 marks if there is not.