CS 112 Spring 2020 – Final Exam

Name: _________________________________ Net Id: ___________

This exam is worth 100 points. It will be effectively scaled to 250 in Gradebook, for a 25% weight.

Write the answer to each question directly under the question. You may use as much space as needed to write your answer, and your answer can span multiple pages. Do not change the order of the questions. Type all your text answers (do not handwrite). If you need to draw something, you may draw it in Word itself. Or, you may draw it on a separate piece of paper (NEATLY!), take a picture, and insert it in this document where it should go in your answer.

When you are finished, convert this document to PDF, and submit to Gradescope.
It would help if you wrote your answers in a different color than black.

Q1. Quicksort [14 pts]

Trace the quicksort algorithm on the following array, and show the full recursion tree, i.e. all splits, on the original array and all subarrays. Use the first item as the pivot when doing a split.

```
24     6     72     28     12     9     3
```

Q2. Heap [15 pts; 7 for algo + 8 for big O derivation]

Given a max heap of *n* items, you want to print all values in the heap that are greater than a given value *k*.
Describe the fastest (in worst case big O) algorithm to do this. You can work directly with the heap array storage.
Assuming there are *m* values in the heap that are greater than *k*, derive the worst case big O running time of your algorithm, with reasoning that accounts for all entities that are counted toward the running time.
You may assume that printing an item takes O(1) time.
If your algorithm is incorrect, you will get NO credit.
If your algorithm is not the fastest possible, you will get NO credit for the algorithm, and at most 4 points for the derivation.

Q3. Dijkstra's Algorithm [14 pts]

A weighted undirected graph has vertices P,Q,X,Y,Z and edges (X,Y,2),(X,P,3),(Y,Z,6),(Y,P,2),(Y,Q,5),(P,Q,3),(P,Z,1).
So edge x—y has weight 2, edge x—p has weight 3, etc.
Dijkstra's algorithm is run on this graph with X as the source vertex, using a min-heap for the fringe.

Draw the fringe **heap ordered array (NOT tree)** for each iteration, just before a vertex is removed from the fringe. For each item in the heap ordered array, write the pair of (vertex name, distance from X) like this: [(v1,3),(v2,5), ...] where (v1,3) is the root of the heap, (v2,5) is the left child of v1, etc.

Note: If you draw the heap tree instead of writing out the heap ordered array, you will get at most half the credit.

Q4. Sorting [14 pts]

You are given traffic data (number of page hits) for *n* websites for some time period (*n* is a large number, in the order of hundreds of thousands.) You are asked to print the top *k* websites with greatest amount of traffic (*k* is much smaller than *n*).
Which of insertion sort, mergesort, quicksort, or heapsort may be used to construct the fastest (in worst case big O) algorithm for this task? What would be its worst case big O running time, in terms of *n* and *k*?

Note: You must pick from one of the given sorting algorithms, anything other than these will get NO credit. You cannot modify the steps in any of these given sorting algorithms, but you can stop it as soon as you get the top *k*. You can use the running times of any of these sorting algorithms (and the components/sub algorithms they may use) without derivation.

Non-fastest solution will get at most half credit.

Q5. Strongly Connected Directed Graphs [15 pts]

A *strongly connected* directed graph is one in which every vertex can reach **all** other vertices via paths with one or more edges.
List the steps of an algorithm/pseudo code (no Java code!) to tell if a directed graph is strongly connected or not. You are not required to derive the running time.

Q6. Directed Graph Reversing Edges [14 pts]

Given a directed graph without edge weights, implement a method to create a new graph whose edges are the reverse of the original graph. In other words, every edge x → y in the original graph should be replaced with the edge y → x in the new graph - note that the new graph will have the same number of edges as the original, just reversed.

Complete the method `reverseEdges` below. The original graph must not be modified in any way. You may not modify the given classes in any way. You may implement helper methods if needed, but you may not use any classes other than the ones defined here.

```
public class DirectedGraph {

       class Edge {
           int vnum; Edge next;
           Edge(int vnum, Edge next) {
                this.vnum = vnum;
                this.next = next;
           }
       }

       int n; // number of vertices
       Edge[] adjLists; // adjacency linked lists, array length is n

       public class DirectedGraph() { } // empty body

       // Returns a new graph whose edges are the reverse
       // of the ones in this graph.
       // Order of edges in the adjacency linked lists does not matter
       public DirectedGraph reverseEdges() {
             /* COMPLETE THIS METHOD */
       }
}
```

Q7. Heap [14 pts; 4+10]

Suppose that we have *k* max heaps, with *n* items in each. We wish to combine these into a single max heap with *n*k* items in it.

Each of the following parts (7a) and (7b) describes one approach to combining the heaps, for which you are asked to derive the worst case big O running time. You must show your derivation, detailing all the components first, then showing how they add up to the total running time.
If your answer is incorrect, you will get NO credit.
If your answer is correct, but you do not show derivation, you will get 1 point.

7a) Create a new array A of length *n*k*. Copy all items from each of the *k* heap arrays to A. Then heapify (build a heap order) A. What is the worst case big O running time? Show your derivation. Assume array allocation is O(1) time.

7b) Group the *k* heaps into *k/2* pairs, and apply the algorithm from part (7a)
on each pair, thus getting *k/2* heaps. Repeat this process until a single
(final) heap remains. What is the worst case big O running time? Show your derivation.
Assume that *k* is a power of 2, and array allocation is O(1) time.