NAME: _____

CSE 331
Introduction to Algorithm Analysis and Design
Sample Final Exam

DIRECTIONS:

- Open Book, Open Notes, No electronics
- Time Limit: 3 hours.

1	/40
2	/25
3	/20
4	/15
Total	/100

A FEW REMINDERS/SUGGESTIONS:

- You can quote any result that we covered in class or any problem that was there in a homework (but remember to explicitly state where you are quoting a result from).
- If you get stuck on some problem for a long time, move on to the next one.
- The ordering of the problems is somewhat related to their relative difficulty. However, the order might be different for you!
- You should be better off by first reading all questions and answering them in the order of what you think is the easiest to the hardest problem. Keep the points distribution in mind when deciding how much time to spend on each problem.

1. ($8 \times 5 = 40$ points) Answer True or False to the following questions and **briefly JUSTIFY** each answer. A correct answer with no or totally incorrect justification will get you 2 out of the total 5 points. (Recall that a statement is true only if it is logically true in all cases while it is is false if it is not true in some case).

   (a) Every undirected connected graph on n vertices has exactly $n - 1$ edges.

   (b) If all the edge weights of an undirected connected graph G are distinct, then G has a unique minimum spanning tree.

   (c) Given n numbers $a_1, \ldots, a_n$, the median of the smallest ten numbers and the largest ten numbers among them can be computed in $O(n)$ time.

   (d) The *maximum* spanning tree problem (i.e. given a connected undirected weighted graph output a spanning tree with the maximum weight) is an NP-complete problem. (Recall that we have shown that the *minimum* spanning tree problem is in P.)

(e) $2^{O(n)}$ is $O(2^n)$. (Or more precisely, every function $f(n)$ that is $2^{O(n)}$ is also $O(2^n)$.)

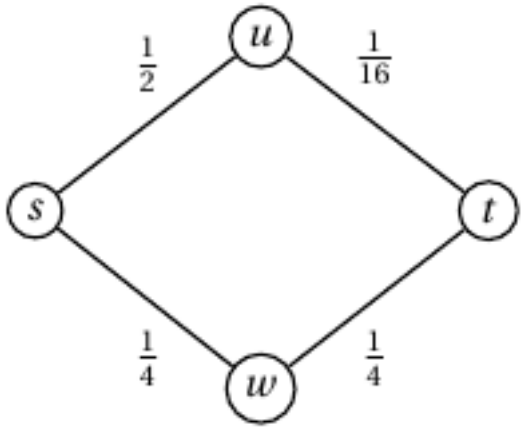
(f) Given n numbers $a_1, \ldots, a_n$, where for every $1 \leq i \leq n$, $a_i \in \{-5, 9, 100\}$; their sorted order can be output in $O(n)$ time.

(g) Given an undirected unweighted graph G in n vertices and m edges and two distinct vertices $s \neq t$, the shortest $s-t$ path can be computed in $O(m+n)$ time.

(h) Let G be an undirected connected graph. If G has a unique minimum spanning tree, then all the edge weights in G are distinct.

2. (25 points) You're given the Internet as a graph $G = (V, E)$ such that $|V| = n$ and $|E| = m$. Further for each edge $e \in E$, you're given it's probability $0 \leq p_e \leq 1$ that it'll transmit a packet (or equivalently, it is the probability it will not *fail*). Further, the probabilities are *independent*, i.e. given a path with edges $e_1, \ldots, e_\ell$, the probability of that the entire path does not fail is given by $\prod_{i=1}^{\ell} p_{e_i}$. Design an $O(m \log n)$ time algorithm, which given two distinct vertices $s \neq t \in V$, outputs the $s - t$ path with the largest probability of not failing. (If you were deciding to route packets from s to t, you should use this path.)

   For simplicity, you can assume that each p_e is a power of $\frac{1}{2}$. For example, in the graph below,



   The path s, u, t has a probability of not failing of $\frac{1}{2} \times \frac{1}{16} = \frac{1}{32}$, while the path s, w, t has a probability of not failing of $\frac{1}{4} \times \frac{1}{4} = \frac{1}{16}$. Thus, your algorithm should output s, w, t.

   Argue the correctness of your algorithm (formal proof is not required). Also briefly justify the run time of the algorithm.

   (*Hint:* It *might* be useful to transform the input and then apply a known algorithm on the transformed input. Further, these identities might be useful: $\log(ab) = \log a + \log b$ and $\log(a/b) = \log a - \log b$. (These identities hold irrespective of the base in the logarithms and hence are not explicitly stated.))

3. (20 points) (This is an CS job interview question.) Given two arrays A and B each with n numbers $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ respectively, the algorithm needs to output the order $a_1, b_1, a_2, b_2, \ldots, a_n, b_n$. E.g. if $n = 4$ and $A = (1, 3, 5, 7)$ and $B = (2, 4, 6, 8)$, then the output is $(1, 2, 3, 4)(5, 6, 7, 8)$. The output must be in the same array as the input. You can think of the input as an array of size $2n$ which contains A and B and your algorithm must rearrange these values as specified.

   What makes the problem interesting that you are only allowed $O(\log n)$ amount of *temporary space*. By temporary space, I mean the total amount of space among all data structures and temporary variables *except* the arrays A and B. In particular, the output has to be written in A and B.

   Give an $O(n \log n)$-time algorithm that uses $O(\log n)$ temporary space. To receive full credit, you must fully specify your algorithm. Argue why your algorithm is correct and justify the running time and temporary space usage of your algorithm.

   (*Note:* If you present an $O(n^2)$ time algorithm with $O(\log n)$ temporary space or a $O(n \log n)$ time and $n + O(1)$-temporary space algorithm then you can get at most 5 points.)

   (*Hint:* It *might* be useful to think of a divide and conquer algorithm.)

4. (15 points) Recall that the Bellman-Ford algorithm for input graph $G = (V, E)$ with costs c_e in each edge $e \in E$ builds an $n \times n$ matrix M such that $M[i, u]$ for $0 \leq i \leq n-1$ and $u \in V$ contains $OPT(i, v)$, i.e. the cost of a shortest $u - t$ path using at most i edges (for a given target $t \in V$). In particular, it computes

$$M[i, u] = \min \left(M[i-1, u], \min_{(u,w) \in E} \left(M[i-1, w] + c_{(u,w)} \right) \right).$$

We can compress the matrix above: if $M[i, u] = M[i-1, u]$, then we do not need to explicitly store $M[i, u]$. In this problem, you will show that even with this compression idea a naive implementation of the Bellman-Ford algorithm still needs to store $\Omega(n^2)$ entries. (Note that the algorithm only needs to store two columns at a time– what this problem is saying is that the idea of *compression alone* will not give you any benefit over the naive algorithm.)

In particular, define an entry (i, u) to be *fresh*, if $M[i, u] < M[i-1, u]$. For every large enough $n \geq 1$ show that there exists a problem instance to the shortest path problem (with possibly negative weights but not negative cycle) on n vertices such that $\Omega(n^2)$ entries in the matrix M as computed by the Bellman-Ford algorithm are fresh.

(*Note*: If you present an example for any fixed $n \geq 4$ such that the matrix has M has at least $n(n-1)/2$ fresh entries, then you can get up to 3 points.)