Instructions: There are six problems in this exam. Some of them consist of several subproblems. Make sure you look through them carefully. Please write legibly.

1. Sorting.
   a. Explain why all comparison-based sorting algorithms must run in $\Omega(n \log n)$ time on an input of $n$ numbers.
   b. Explain why the worst-case runtime of Randomized Quicksort is $O(n^2)$ but its average runtime is $O(n \log n)$ for sorting $n$ numbers.
2. True or false. If your answer is true, please provide a brief explanation; if it is false, please provide a counterexample.

(a) Let $G$ be a connected graph and $s$ be some node of $G$. Suppose both Alice and Bob implemented and ran BFS on $G$ starting at $s$. It must be the case that the BFS-tree produced by Alice’s code is exactly the same as the BFS-tree produced by Bob’s code.

(b) Let $G$ be a connected graph and $s$ be some node of $G$. It is possible that a BFS-tree of $G$ rooted at $s$ is exactly the same as the DFS-tree of $G$ rooted at $s$.

(c) Let $G$ be a connected graph whose edges have non-negative weights. If no two edges have the same weight, then there is only one shortest path from $u$ to $v$ for any two nodes $u$ and $v$. 
3. Given a sorted array of distinct integers $A[1 \ldots n]$ (some of which can be negative), we are interested in determining if there is an index $i$ in the array so that $A[i] = i$. Of course, one can simply scan the array to solve the problem. We want to do better.

a. Design an $O(\log n)$-time algorithm that solves the problem. (Hint: Keep in mind that the elements of $A[1 \ldots n]$ are distinct integers and already sorted.)

b. Why is your algorithm correct, and why is its runtime $O(\log n)$?
4. Let $P$ be a path on $n$ nodes where the $i$th node is $v_i$. Suppose $v_i$ has weight $w_i$ for $i = 1, \ldots, n$. Let $S$ be a subset of nodes in $P$. The weight of $S$ is simply the sum of the weights of the nodes in $S$. We say that $S$ is an independent set if no two nodes in $S$ are joined by an edge. Our goal is to find an independent set in $P$ whose total weight is as large as possible. Consider the example below. It is a path on 5 vertices where the maximum weight of an independent set is 14.

(a) Here’s a very natural algorithm to consider - a “heaviest-first” greedy algorithm:

Initialize $S$ to the empty set.
While some node remains in $P$
  Pick a node $v_i$ of maximum weight.
  Add $v_i$ to $S$.
  Delete $v_i$ and its neighbors from $P$.
Endwhile
Return($S$).

As you know by now, greedy methods are deceiving – they’re the first algorithms we think about but they don’t always work. By providing a counterexample, show that the above algorithm does not always find an independent set of maximum total weight.

(b) On the other hand, dynamic programming (DP) is quite powerful. It can solve problems that greedy methods can’t. Design a DP-based algorithm that finds an independent set of maximum total weight in $P$. Note: Your algorithm must output an independent set of maximum weight, and not just the maximum weight. What is the runtime of your algorithm? (Hint: Let $OPT(i)$ be the maximum weight of an independent set in the path consisting of nodes $v_1, v_2, \ldots, v_i$.)
5. *DAGs.*

a. Find a topological ordering of the vertices of the DAG above.

b. Using this topological ordering, find the shortest path distances from *a* to every other node in the graph. Please show your work.
6. Suppose we are given a graph $G = (V, E)$, and each edge $e$ has a probability $p(e)$ with $0 \leq p(e) \leq 1$ associated with it. Let path $P = e_1, e_2, \ldots, e_k$. The reliability of $P$ is defined as the product of the probabilities of its edges; i.e.,

$$\text{reliability of } P = \prod_{i=1}^{k} p(e_i).$$

In the context of computer networks, if $p(e)$ represents the probability of edge $e$ not failing, then reliability of path $P$ is the probability that $P$ is working assuming that the edge failures are independent events. If a network wants to send a message from $u$ to $v$, it’s natural for it to choose a $u$-$v$ path that has the highest reliability. The goal of the maximum reliability problem is to determine a maximum reliable path from a specified source node $s$ to every other node in the network. Solve this problem by modifying Dijkstra’s algorithm. Please make sure you do the following:

a. Define what $D[u]$ represents for each vertex $u$.

b. What are the base cases for the $D$-values?

c. Specify that type of priority queue you should use and the keys of the items stored in the priority queue.

d. How should the $D$-values be updated?

e. Why is the framework of Dijkstra’s algorithm valid in this situation?

f. What is the runtime of your modified algorithm?