Q1. Given that $f(n) = \Theta(u(n))$ and $g(n) = \Theta(v(n))$ are both positive functions, answer True or False to each of the following: You must prove your answers.

a) If $f(n) > g(n)$ for all $n > 0$, then $f(n) - g(n) = \Theta(u(n))$

b) $\max(f(n), g(n)) = \Theta(u(n) + v(n))$

c) $f(g(n)) = \omega(u(v(n)))$

d) if $f(n) = \Omega(g(n))$ then $v(n) = O(u(n))$

e) either $f(n) = \omega(v(n))$ or $g(n) = o(u(n))$

Q2. Solve for the asymptotic runtimes for best case, worst case, and expected case for the following algorithm. You may assume $n$ is a power of 2. Let $G(n)$ be $F(n)$ with the last line changed to read: return $3F(n/2) + y$. What is the runtime of $G(n)$? Does it compute the same result as $F(n)$ in the expected case? Be sure to answer all 5 parts of this question.

$F(n)$:

if $n \leq 1$ then return 1
choose a uniformly random integer $X$ in the interval 1 to $n$
y = 1
for $i = 1$ to $X$
y = $y \cdot i$
return $F(n/2) + F(n/2) + F(n/2) + y$

Q3. Write a version of Quicksort that always partitions the array by selecting the median as the pivot. Make sure to do this using the most (asymptotically) efficient solution you can think of. What are the best, worst, and expected case runtimes of your algorithm? Compare to the best, worst, and expected case runtimes of the unmodified Quicksort algorithm (write these down). Which algorithm has better asymptotic performance? Which algorithm is more practical for sorting randomly permuted inputs?

Q4. Let $H$ be a universal hash function from the universe of all keys $U$ to the interval $\{0, \ldots, m - 1\}$. Suppose we store $n = m/2$ keys in a hash table with $m$ entries using universal hashing, and that collision resolution is done by chaining. What is the probability that the $i$-th insertion into the table results in a chain of $k$ elements (the chain must have contained exactly $k - 1$ elements prior to the insertion)? What is the probability that all $n$ elements end up in the same chain? What is the probability that no chain in the entire table contains more than one element after $n$ insertions? Write down the simplest expression you can for each. Show all work.

Q5. Hand simulate Kruskal’s algorithm on the graph below, and list the output edges. Hand simulate Prim’s algorithm, using vertex $a$ as the root, and list the output edges. You must list output edges in the order that they are accepted by the corresponding algorithms. Hand simulate Dijkstra’s algorithm on the graph below, using vertex $a$ as the source, and list the sequence of path cost updates for each vertex.
Q6. Recall the Longest Common Substring (LCS) problem. Given two strings specified by character arrays $X[1..n]$ and $Y[1..m]$, the LCS problem is to find the longest string $Z[1..k]$ such that $Z$ is a substring of both $X$ and $Y$. (A substring can be formed by removing characters from the original string.)

We now consider the Weighted Common Substring (WCS) problem, which in addition to $X$ and $Y$ specifies a (possibly negative) value $V(c)$ for each character $c$. We wish to find a common substring $Z$ of both $X$ and $Y$ such that sum of the values of the characters in the substring is maximized. That is, we wish to maximize $\sum_{i=1}^{k} V(Z[k])$ subject to the constraint that $Z$ is a substring of both $X$ and $Y$.

a) Give the most efficient algorithm you can think of to solve the WCS problem, and analyze its runtime.

b) Suppose we change the WCS problem to consider subsets, instead of substrings (i.e. the ordering of the characters is no longer relevant). You may assume all the characters in $X$ are distinct, and all the characters in $Y$ are distinct. The problem is to find a subset $Z$ containing only characters which are found within both $X$ and $Y$ such that the above sum is maximized. Give a greedy algorithm to solve this problem in $O(n \log n + m \log m)$ time.

Q7. Design a modified hash table data structure with the following properties. Given that there are $m$ slots in the table and $n$ keys to be stored (where $n$ may be larger than $m$), we wish to use at most $O(m+n)$ storage space while maintaining a worst case lookup time of $O(\log n)$ for any key. Additionally, we want a worst case (but amortized) bound of $O(\log n)$ time for insertions. You need not consider the possibility of key deletions, and you may assume the hash function that maps keys to slots can be computed in constant time. Hint: Each slot in the table should contain an array that is no more than twice as large as the number of keys currently stored in that slot. The contents of each array should be viewed as a data structure that supports $O(\log n)$ time operations. You may reuse the standard claims about runtimes and amortized costs for any standard data structures that apply.
Q8. Answer each of the following multiple choice and fill-in-the-blank questions.

a) Which of the following graph traversal techniques should be used for edge classification: In-order, post-order, BFS, DFS

b) Consider the following claim: There exists a comparison based sorting algorithm that sorts $n$ items in $O(n)$ expected time. Does this claim violate the lower bound for sorting of $\Omega(\text{________})$ that was proved via a decision tree? Yes / No  (be sure to fill in the blank as well)

c) Which of the following most closely describes topological sort:
   DFS combined with a LIFO stack
   DFS combined with FIFO queue
   BFS combined with a LIFO stack
   BFS combined with a FIFO queue

d) Which of the following graph properties most accurately specifies the requirements for correctness of the Bellman-Ford algorithm:
   No negative weight cycles
   Directed Acyclic Graph (DAG)
   Undirected graph
   No negative weight edges

e) What is the runtime of the Floyd-Warshall algorithm on a graph with V vertices and E edges?
   ________