CS302 Final Exam, December 5, 2016 - James S. Plank

Question 1

For each of the following algorithms/activities, tell me its running time with big-O notation. Use the answer sheet, and simply circle the correct running time. If *n* is unspecified, assume the following:

- If a vector or string is involved, assume that *n* is the number of elements.
- If a graph is involved, assume that *n* is the number of nodes.
- If the number of edges is not specified, then assume that the graph has $O(n^2)$ edges.

A: Sorting a vector of uniformly distributed random numbers with bucket sort.
B: In a graph with exactly one cycle, determining if a given node is on the cycle, or not on the cycle.
C: Determining the connected components of an undirected graph.
D: Sorting a vector of uniformly distributed random numbers with insertion sort.
E: Finding a minimum spanning tree of a graph using Prim's algorithm.
F: Sorting a vector of uniformly distributed random numbers with quicksort (average case).
G: Calculating *Fib(n)* using dynamic programming.
H: Performing a topological sort on a directed acyclic graph.
I: Finding a minimum spanning tree of a graph using Kruskal's algorithm.
J: Finding the minimum cut of a graph, after you have found the network flow.
K: Finding the first augmenting path in the Edmonds Karp implementation of network flow.
L: Processing the residual graph in the Ford Fulkerson algorithm, once you have found an augmenting path.

Question 2

Please answer the following statements as true or false.

A: Kruskal's algorithm requires a starting node.

B: The maximum flow of a graph is always a value that is the sum of the weights of some collection of edges in the graph.

C: On a directed, acyclic graph, finding the shortest path from one node to another will be faster using topological sort than Dijkstra's algorithm.

D: Quicksort requires more memory than mergesort.

E: Suppose we are running insertion sort on a large vector. If the starting index of each element is within 50 of its finishing index, then insertion sort will run in *O(n)* time.

F: The "median of three" pivot selection algorithm finds the best pivot for quicksort.

G: Memoization avoids making duplicate recursive calls in a dynamic program.

H: Heap sort requires more memory than quicksort.

I: Bucket sort always runs in linear time.

J: When determining network flow, if there exists an augmenting path, then you can find the one that has maximal flow in $O(|E| \log |V|)$ time.

CS302 Final Exam, December 5, 2016 - Page 2

Question 3

	0	1	2	3	4	5	6	7
0		3	5	15				26
1			5		5	14	11	
2				1	2	13	7	10
3							1	
4						2		8
5								5
6								2
7								

Part A: You are performing Dijstra's shortest path algorithm on a weighted, directed graph that has the adjacency matrix to the right. You are at the point in Dijkstra's algorithm where you have processed nodes 0 and 1. At this point, the multimap is as follows:

{ [5,2] [8,4] [14,6] [15,3] [17,5] [26,7] }

Your job is to show me what the multimap is after the next node is processed in Dijkstra's algorithm. Use the same format as I used above to show me what the multimap is.

Part B: Now, you are performing Prim's algorithm for minimum spanning tree, and again, you have processed nodes 0 and 1. At this point, the multimap is:

{ [5,2] [5,4] [11,6] [14,5] [15,3] [26,7] }

Your job is to show me what the multimap is after the next node is processed.

Question 4

To the right is the class definition for a graph.

```
class Graph {
  public:
    vector < vector <int> > Adj;
    vector <long long> Paths;
    vector <int> Incident;
    long long Num_Paths(int from, int to);
};
```

- **Adj** is a vector of adjacency lists for each node. In other words, **Adj[i]** is a vector of integers which is the adjacency list for node *i*. It is an integer, so that if **Adj[i][j]** equals *k*, then there is an edge from node *i* to node *k*.
- **Paths** is a vector of **long long**'s that you will use. **Paths.size()** is equal to **Adj.size()**.
- **Incident** is another integer array that you will use. Its size is the same as **Paths**.

Now, your job is to implement **Num_Paths()**, which needs to return the number of paths in the graph from node *from* to node *to*. You should make the following assumptions (and pay attention to these, because they are intended to make your life easier, and not harder):

- The graph is a directed acyclic graph.
- When **Num_Paths()** is called, **Incident[i]** will already be set to equal to the number of edges incident to node *i*.
- **Incident[from]** will be equal to zero.
- Every node in the graph is reachable from node **from**.

CS302 Final Exam, December 5, 2016 - Page 3

Question 5

Suppose Alice and Bob are playing a game involving a pile of **P** stones. At each person's turn, he/she can remove *s* stones, where *s* is an element of a vector **S** of integers. If the number of stones is less than the smallest element of **S**, then that person loses.

Alice an Bob are going to play the game optimally. In other words, if Alice can win, she will make a move that cause her to win. Same with Bob.

You are to write the method **Stones** of the class **Game**. It has the following prototype:

```
string Stones(int P, vector <int> &S);
```

This method should return `"Win"` if Alice will win the game, when it is her turn, and there are **P** stones on the pile. If instead, Alice will lose the game, then this method should return `"Lose"`.

I'll give you some Topcoder-like constraints:

- **P** will be less than or equal to 1,000.
- **S** will have at most 20 elements in it.
- Each element of **S** will be between 1 and 1000 (inclusive).
- Your program should run in under two seconds given these constraints.

Answer this on the answer sheet. I already have the class definition and the framework of the method filled in for you. You may add data to the class if you want. Don't bother with any **include** or **using** statements.

Let me give you some hints in case you are having problems.

- If Alice can remove *s* stones, and Bob has to lose when the pile has *P-s* stones, then Alice will win.
- If there is no value of $s \in S$ such that Bob loses when the pile has *P-s* stones, then Alice will lose.
- It doesn't matter if it's Alice or Bob calling **Stones(p, S)**. It will return "Win" if the caller wins, and "Lose" if the caller loses.