Sample Solutions - Past Exam 3

Q1

This specific recurrence relation of this question could be solved using methods such as telescoping, Master's theorem or recursion trees. Below we present a detailed solution using the telescoping method, which is quite general. Note that in the exam question you only had to select the correct asymptotic complexity.

We first have that

$$T(n) = 2T\left(\frac{n}{4}\right) + cn$$

We want to substitute in a lower recurring term, we can do this via using our recurrence relation to find $T(\frac{n}{4})$ and appropriately subbing into the above. Doing this we get

$$T(n) = 2\left[2 \cdot T\left(\frac{n}{4^2}\right) + c\left(\frac{n}{4}\right)\right] + cn = 2^2 \cdot T\left(\frac{n}{4^2}\right) + cn\left(1 + \frac{1}{2}\right)$$

We keep repeating this procedure, by again using our recurrence relation to find $T(\frac{n}{4^2})$ and appropriately subbing into the above. Doing this we get

$$T(n) = 2^2 \cdot \left[2T\left(\frac{n}{4^3}\right) + c\left(\frac{n}{4^2}\right)\right] + cn\left(1 + \frac{1}{2}\right) = 2^3 \cdot T\left(\frac{n}{4^3}\right) + cn\left(1 + \frac{1}{2} + \frac{1}{2^2}\right)$$

From here we can infer the general form which is just

$$T(n) = 2^k \cdot T\left(\frac{n}{4^k}\right) + cn\sum_{i=0}^{k-1}\left(\frac{1}{2}\right)^i$$

We now want to get rid of the recurrence term, we can find an appropriate value of k such that we invoke the base case. This occurs when $T(\frac{n}{4^k}) = T(1) = b$, hence $k = \log_4(n)$. Subbing this into our general form we get

$$\begin{aligned}
T(n) &= 2^{\log_4(n)} \cdot b + cn\sum_{i=0}^{\log_4(n)-1}\left(\frac{1}{2}\right)^i \\
&= n^{\log_4(2)} \cdot b + cn\left(\frac{1 - \left(\frac{1}{2}\right)^{\log_4(n)}}{1 - \frac{1}{2}}\right) \\
&= n^{1/2} \cdot b + 2cn\left(1 - n^{\log_4(1/2)}\right) \\
&= n^{1/2} \cdot b + 2cn\left(1 - n^{\log_4(4^{-1/2})}\right) \\
&= n^{1/2} \cdot b + 2cn\left(1 - n^{-1/2}\right) \\
&= n^{1/2} \cdot b + 2cn - 2cn^{1/2} \\
&= 2cn + \sqrt{n}(b - 2c)
\end{aligned}$$

Thus clearly $T(n) = \Theta(n)$.

Q2

Answer: **B**, **D**

(A) This is false, this can be seen by looking at the initialisation of the proposed loop invariant. At initialisation $x = 0$, $i = 1$, the invariant states "$x = 0$ is the number of items with factor of m in $\mathsf{L}[1 \dots 1]$", but $\mathsf{L}[1]$ could be a factor of m, hence the value of x is not true for any input.

(B) This is true, as the invariant at this point would state "x is the number of items with factor of m in $\mathsf{L}[1 \dots n+1-1] = \mathsf{L}[1 \dots n]$", which is the entire list, which would showcase the algorithm is correct.

(C) This is false, as the invariant at this point would state "x is the number of items with factor of m in $\mathsf{L}[1 \dots n-1]$", this implies the algorithm is incorrect as it hasn't checked the entire list.

(D) This is true. Initialisation: $x = 1$ or $x = 0$, $i = 1$, the invariant states "$x = 0$ is the number of items with factor of m in $\mathsf{L}[1 \dots 1] = \mathsf{L}[1]$", which is true, as x is either 1 if $\mathsf{L}[1]$ has a factor of m or 0 if $\mathsf{L}[1]$ if it hasn't. Maintenance: assume upon completing the k-th iteration that the invariant is maintained/correct, therefore we have that "x is the number of items with factor of m in $\mathsf{L}[1 \dots k]$", lets say $x = c$. We then complete the next iteration $i = k+1$. If $\mathsf{L}[k+1]$ has a factor of m then $x = c+1$ or it hasn't then $x = c$. In either case x is now the number of items with factor m in $\mathsf{L}[1 \dots k+1]$, which correctly bring us to the next invariant statement. Termination: the loop terminates at the end of the n-th iteration, at this point the invariant states "x is the number of items with factor of m in $\mathsf{L}[1 \dots n]$", which is the entire list and implies correctness of the algorithm.

Q3

We first convert all the N numbers to base N. This means each number has a most $\log_N(N^5) = 5$ digits. The conversion cost $\Theta(1)$ for each number and hence $\Theta(N)$ for every number.

We now perform radix sort, which will be a sequence of 5 counting sorts. Each count sort costs $\Theta(N + N)$.

Hence the total cost is $\Theta(N)$.

Q4

Answer: $\boldsymbol{\Theta(\mathbf{V})}$

Need to iterate over a row in the matrix and sum the weights.

Answer: $\boldsymbol{\Theta(\mathbf{1})}$

Can directly index the matrix to see if the edge exists.

Answer: $\mathbf{\Theta(V^2)}$

In BFS for each vertex we need to examine its outgoing edges, which requires iterating over a row in the adjacency matrix representation. There are V vertices and the iterating over a row cost $\Theta(V)$ giving us $\Theta(V^2)$.

Answer: $\mathbf{\Theta(V)}$

Must iterate over the edges of vertex A and iterate over the edges of Vertex B, which in the worst case for both is $\Theta(V)$. Note that the question stated that the interior list are unsorted.

Q5

Answer: **TRUE**

On one hand, in order to compute the in-degree of every vertex we need to access all edges and all vertices, thus it has time complexity $\Omega(V + E)$. On the other hand, by accessing all data in adjacency list representation and keeping one counter per vertex (which we can clearly do in time complexity $O(V + E)$) it is possible to get the in-degree of every vertex. Putting those facts together, we get that the time complexity is $\Theta(V + E)$.

Answer: **TRUE**

We won't finalise a node greedily without it being the best choice. Because the only edges with negative weights are outgoing from the source node, all these edges are immediately relaxed, and there is no negative cycles, we know from this point onwards distance to these adjacent nodes can only ever strictly increase, thus it is safe to greedily finalise a path, as attempting to take any other path would be the same length or longer. This would not be true if the negative edges arbitrarily existed within the graph even without the presence of the negative cycles.

Answer: **FALSE**

We can prove this is false by showing that Dijkstra's will produce a tree (that is not a minimum spanning tree) with a small counter example. Consider for instance a graph with a source node s and nodes a and b. Let there be edges (s, a) and (s, b) with weight 2, and an edge (a, b) with weight 1. Dijkstra's algorithm would select edges (s, a) and (s, b). However, on any minimum spanning tree one of the selected edges should be (a, b) as that would result in a smaller total weight.

Q6

Answer: **4**

The largest path from the root downwards in the BFS tree would be $A \to F \to D \to H \to I$.

Q7

Answer: **A**

(A) If no distances are updated in one iteration of Bellman-Ford, it is therefore true that they won't update in the next iteration. Thus you can terminate early.

(B) You only ever need to remember the latest best distance. Hence only $O(V)$ space is needed.

(C) The outer loop purpose is to consider different intermediate vertices. You must consider every intermediate to determine the shortest distances.

(D) The diagonal values in the memo matrix represent the shortest distance from a vertex to itself. If there exists a negative cycle, there will be some way for a vertex to return to itself with a negative distance. Thus there can exist a negative value on the diagonal.

Q8

Answer: **A**

For Problem 2 the sum of the demands don't equate to zero so you can immediately say there is no feasible solution. Problem 1 does have a feasible solution. This can be confirmed via a transformation to a max-flow problem as shown in the seminars/notes/applied classes. The max flow is 6, meaning the source edges are saturated, which confirms that Problem 1 has a feasible solution.

Q9

Answer: **B**, **D**

(A) The fact there is negative edges won't effect the correctness of Kruskal's algorithm.

(B) Kruskal's will now select edges as big weight as possible, thus forming a maximum spanning tree.

(C) There are clear counter examples. For example, take 3 nodes that form a triangle with all the edges being equal weights. We have 3 distinct MST's for this graph.

(D) If the weights of all edges are unique, then the minimum spanning tree of that graph is unique (this is not specific to how Prim's algorithm finds MST's). This can be proved via contradiction. Assume that the MST isn't unique and you can form distinct MST's A and B. Let e be the edge with the smallest weight that is on one of the MST's but not in the other (and let's assume without lack of generality that e is in A). Now consider $C = B \cup \{e\}$. Since B is a MST, C contains a cycle D involving e. We know that A is a tree and therefore contains no cycle. Thus, we have that D contains some edge f that is not in A. We get that the weight of f is bigger than the weight of e as e was chosen as the edge with the smallest weight that is on one of the MST's but not in the other. As e and f are part of a cycle, by setting $B' = (B \cup \{e\}) \setminus \{f\}$ we get a MST with a smaller weight than B, which is a contradiction. Hence the MST is unique if the weights of all edges are unique.

Q10

C is the capacity and input to the 0/1 Knapsack problem. We can represent C with $\log_2(C+1)$ bits. Thus we have $C = 2^B$, where B is the number of bits to specify C.

The 0/1 Knapsack algorithm runs in $O(CN)$, where N is the number of items (an array), and C is the max capacity (a number). Combining what we have established we see that the running time is in fact $O(2^B N)$, thus this bound is exponential in terms of the input.

Q11

Answer: **A**, **B**

(A) False. For certain dynamic programming problems, the use of different approaches may change the amount of sub-problems that need to be solved, but using a bottom up approach is not asymptotically faster than top down approach.

(B) False. This statement implies each sub-problem is $O(1)$ to solve. But there is no reason why sub-problems cannot take longer to compute.

Q12

Answers: **4**, **9**, **6**, **x**, **v**, **y**

Q13

Answer: **B**, **D**

(A) False, linear probing causes primary clustering.

(B) True, quadratic probing causes secondary clustering.

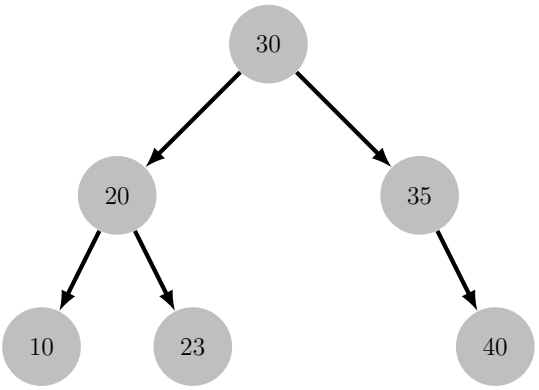
(C) False, quadratic probing does not cause primary clustering.

(D) True, quadratic probing does not necessarily probes all positions of the hash table.

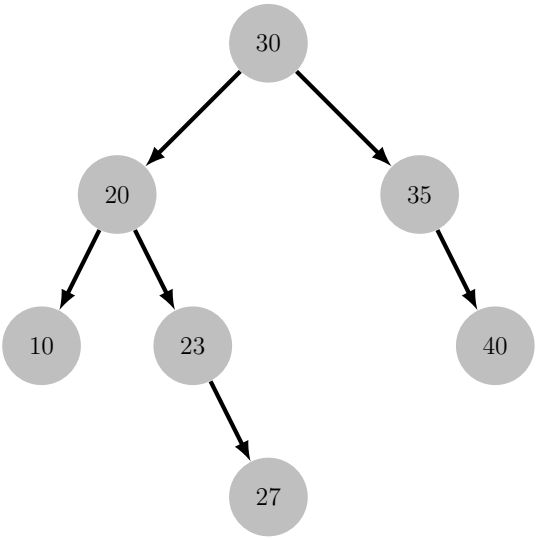
(E) False, linear probing probes all positions of the hash table.

Q14-Q16

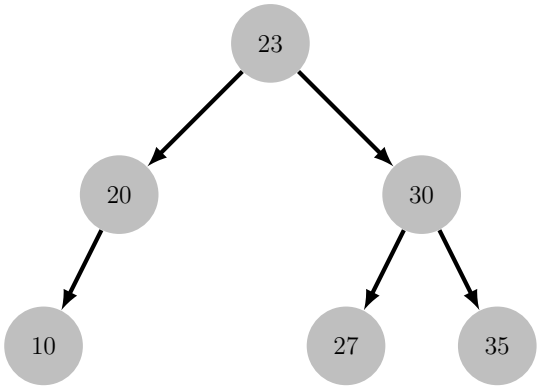
After deleting node 25, node 23 becomes unbalanced and a left-left rotation is needed, resulting in the AVL tree:



The insertion of node 27 does not make any node unbalanced, resulting in the AVL tree:



Finally the deletion of node 40, makes node 30 unbalanced and a left-right rotation is required, which results in the AVL tree:



The initial rotation of the left-right rotation pushes 20 down to become the left child of 23. The second rotation of the left-right rotation causes 23 to be pushed up and subsequently gain a new right child, it therefore needs to give 27 to be a left child of 30. Therefore the answers to questions are as follows.

Answer to Q14: **A**

Answer to Q15: **C**

Answer to Q16: **F**

Q17

Answer: **A**, **C**

We have compared up to length $k = 1$ beforehand. If the ranks of suffixes with IDs x and y are equal, then we need to compare the ranks of suffixes with IDs $x + k$ and $y + k$ to determine their relative rank after sorting on the first $2k$ characters.

(A) $\mathsf{Rank}[4 + k] > \mathsf{Rank}[10 + k]$ hence true.

(B) $\mathsf{Rank}[4 + k] \neq \mathsf{Rank}[6 + k]$ hence false.

(C) $\mathsf{Rank}[3 + k] = \mathsf{Rank}[9 + k]$ hence true.

(D) $\mathsf{Rank}[2 + k] = \mathsf{Rank}[4 + k]$ hence false.

Q18

Refer to the solution of **Problem 6 of Week 9 Applied**.

Q19

Refer to the solution of **Problem 6 of Week 8 Applied**.

Q20

Refer to the solution of **Problem 2 of Week 4 Applied**.

Q21

This problem can be solved utilising the Quickselect algorithm. We can first find the respective top 20% and bottom 80% by utilising Quickselect on the Elo Rating list and finding the $k = 0.8N$ value (assuming ascending order). This will partition the array into the bottom 80% (amateur) and top 20% (pro).

For each respective group list we can again use Quickselect $k = 0.8N - 16$ and $k = 0.2N - 16$ respectively to find the top 16th player for each group. This will partition each group list into a top 16, and the rest into the group stage.

Q22

Answer: **C**

Each superhero can only ever defend two planets at a time, thus the capacity of edges from the source to set L is 2. For each superhero they can connect to every planet in the Set R and the edge capacity is 1 as a hero cannot defend the same planet multiple times, thus we only ever want to allocate a singular flow to a planet at most from a single superhero. For each planet node they only need at most 1 hero, so the edges from Set R to the sink is just 1. Note though the question didn't require it, the in degree (amount of edges) to each planet needs to be limited to 10, and edge allocation should be done in accordance to the requests each planet placed.