National University of Computer and Emerging Sciences, Lahore Campus

|			
Course Name:	Design and Analysis of Algorithms	Course Code:	CS2009
Degree Program:	BSCS, BSSE	Semester:	SPRING 2023
Exam Date:	Saturday, June 10, 2023	Total Marks:	55
Section:	ALL	Page(s):	10
Exam Type:	Final (Solution)	Duration	2 hours 30 minutes

Roll No._____ **Section:**_____

Instruction/Notes: Attempt all questions in the answer sheet in the space provided. You can use extra sheet or rough work, but it should not be submitted. Extra sheet will not be graded.

Question #	Total Marks	Marks Obtained		
		CLO 1	CLO 2	CLO 3
Q1:	10			
Q2(a):	3			
Q2(b),(c):	12			
Q3:	10			
Q4:	10			
Q5(a):	4			
Q5(b),(c):	6			
Total Marks Obtained (**CLO wise**)				
Total Marks Obtained				

Question 1: [5+5] Marks

a) A long string consists of five characters Q, W, E, R, T, Y. They appear with the frequency 40, 20, 90, 72, 80 and 2 respectively. Draw the Huffman Tree and write down the Huffman encoding for these six characters. (**No partial marks for this part**)

Alphabet	Huffman Encoding
Q	001
W	0001
E	11
R	01
T	10
Y	0000

The Tree:
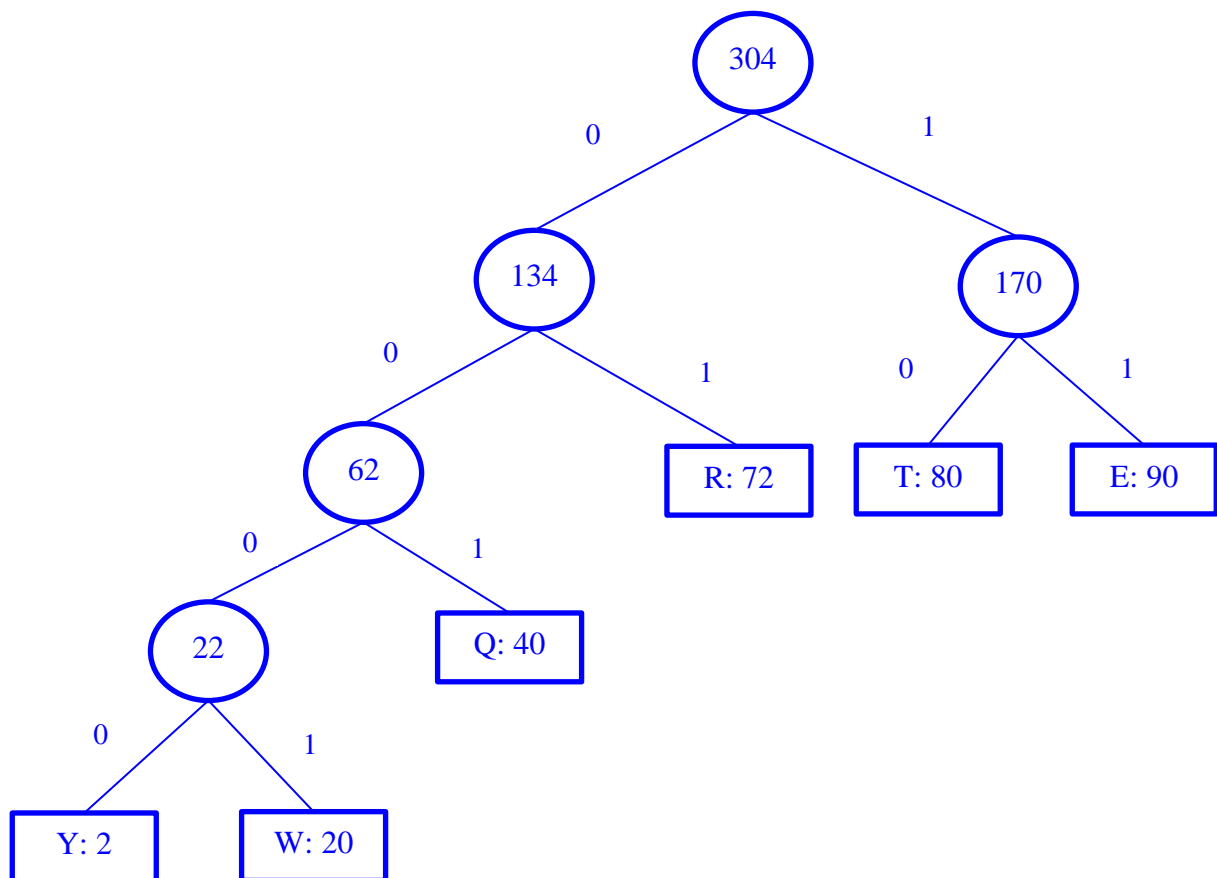
b) In the activity selection problem, suppose that instead of always selecting the first activity to finish, we select the last activity to start that is compatible with all previously selected activities. Describe how this approach is a greedy algorithm, and prove that it yields an optimal solution.

As we choose an activity to add to our optimal solution without having to first solve all the subproblems, the approach mentioned above is a greedy one. Now we show that it yields an optimal solution.

Let X be a set of activities and let x be an activity with latest start time in X. Let Y be a maximum-size subset of mutually compatible activities in X; and let y be an activity with latest start time in Y. If y is the same activity x, we are done. Otherwise, let $Z = Y - \{y\} \cup \{x\}$ be a new set of activities.

> [**Rubric**] **3 Marks** up to this point. One must have
>
> - isolated an activity with latest start time in the problem
> - isolated an activity with latest start time in a solution
> - made a new solution by exchanging the two activities
>
> Note: It would be **all three points or none.**

The activities in Z are disjoint, which follows because the activities in Y are disjoint, y is the last activity in Y to start, and $s_y \leq s_x$ (i.e. start time of y is no later than the start time of x).

> [**Rubric**] **1 Mark** to show that new solution comprises of disjoint activities.

Since $|Z| = |Y| - 1 + 1 = |Y|$, we conclude that Z is a maximum-size subset of mutually compatible activities of X, and it includes x.

> [**Rubric**] **1 Mark** to show that new solution comprises of maximum number of disjoint activities; and it includes the already isolated activity with latest start time in the problem.

Question 2:   [3+5+7] Marks

a) Give an example of a weighted directed graph, G, with negative-weight edges such that Dijkstra's algorithm fails to find the correct shortest path. You must mention the start vertex and the vertex for which Dijkstra's algorithm will compute incorrect shortest path. (**No partial marks for this part**)
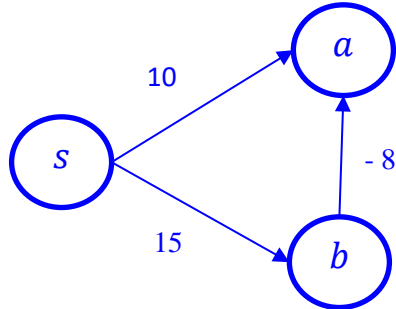
start vertex: s

$a.d = 10$

$\delta(s, a) = 7$

b) Suppose that we are given a weighted, directed graph G = (V , E) in which edges that leave the source vertex **s** may have negative weights, all other edge weights are non-negative, and there are no edges going to the source vertex **s**. Describe an algorithm **(basic idea only)** to find the shortest path from **s** to all other vertices.

- Your algorithm must be asymptotically better than the Bellman Ford algorithm.
- Justify (informally, *i.e without using any formal techniques like loop invariant etc.*) the correctness of your algorithm.

Dijkstra's algorithm will work fine in this situation.

<div align="right">

[**Rubric**] **3 Mark**

</div>

Justification: As there are no negative weight edges starting from an *intermediate* vertex on any path (including the shortest paths), once, we have extracted a vertex v, it is not possible that $v.d$ could have been decreased later. Hence, for each vertex $v \in G.V$, $v.d = \delta(s, v)$ at the time v was extracted.

<div align="right">

[**Rubric**] **2 Mark**

</div>

A more formal argument (optional): We claim that in the given scenario, for each vertex $v \in G.V$, $v.d = \delta(s, v)$ at the time v was extracted. For the sake of contradiction, let's assume that this is not the case. *For the sake of simplicity, let's assume that there are no zero weight*

---

edges. Let y be the *first* vertex, such that $y.d > \delta(s, y)$ at the time y was extracted. Let x be the last vertex on *a shortest path* from s to y that was extracted before y. At that time, $Relax(x, v, w)$ was called for all $v \in G.Adj[x]$.

- If there were some intermediate vertex $z \in G.Adj[x]$ between x and y on *that* path, then after $Relax(x, z, w)$, $z.d < y.d$ (as $z.d = \delta(s, z) < \delta(s, y) < y.d$), hence, z must have been extracted before y. But that leads to a contradiction (as x is the last vertex on *that* shortest path from s to y that was extracted before y).
- If there were no intermediate vertex $z \in G.Adj[x]$ between x and y on that path, then after $Relax(x, y, w)$, $y.d = \delta(s, y)$, once again leading to a contradiction (as $y.d = \delta(s, y) \rightarrow y.d \ngtr \delta(s, y)$).

Hence, no such vertex y exists such that $y.d > \delta(s, y)$ at the time y was extracted. In other words, for each vertex $v \in G.V$, $v.d = \delta(s, v)$ at the time v was extracted.

c) Given a graph G and a source vertex s, you have already constructed a minimum spanning tree T using Prim's algorithm (hence, the information $v.\pi$ is available with you for each $v \in G.V$). Suppose that now the weight of one of the edges (u, v) not in T is decreased (moreover, u is also an ancestor of v in T).

Describe an $O(V)$ algorithm **(basic idea only)** for finding the minimum spanning tree in the modified graph (assume that $w(x, y)$ returns the weight of the edge (x, y) in $\Theta(1)$ time).

As u is an ancestor of v in T, we have to see the edges on the path from u to v in T and find the edge with maximum weight on that path.

> [**Rubric**] **1 Mark** to show that we need to find an edge with maximum weight on the path from u to v in T

As we have only π information of the vertices available at hand, we will traverse this path in reverse order; i.e. we start at $(v, v.\pi)$ then gor to $(v.\pi, v.\pi.\pi)$ and so on till we reach some vertex v' such that $v'.\pi = u$, the last edge in this sequence would be (v', u).

This whole process can be done in $O(V)$ time as there are at most $|V| - 1$ edges on this path and the weight of each edge can be obtained in $\Theta(1)$ time.

> [**Rubric**] **1 Mark** to show that this can be done in $O(V)$ time

Let (x, y) be an edge with maximum weight on that path.

If weight of (x, y) is not greater than the new (reduced) weight of (u, v), then we do not have to do anything; T will remain an MST for the modified graph.

> [**Rubric**] **1 Mark** to show that MST will remain unchanged if $w(u, v) \geq w(x, y)$

Otherwise, (i.e. $(u, v) < w(x, y)$), we remove (x, y) from T, this will break T into two components. Adding (u, v) reconnects them and gives us a new MST for the modified graph.

> [**Rubric**] **2 Mark** to show how to modify the MST if $w(u, v) < w(x, y)$

This can be done by setting the π value of each vertex in the path form $v.\pi$ to y to its child vertex, e.g. let $a.\pi = b$, then we set $b.\pi = a$. As there are at most $|V|$ vertices on the path from $v.\pi$ to y, this whole process takes $O(V)$ time. At the end, we make v child of u (i.e we set $v.\pi = u$) which can be done in $\Theta(1)$ time.

> [**Rubric**] **2 Mark** to show that this can be done in $O(V)$ time

---

Question 3: [3+7] Marks

In a certain course there is an MCQ exam with help sheet. This help sheet contains **L** lines. There are *n* topics included in the exam. For each topic i, it is known that there will be **Q[i]** questions in the exam and the marks for these questions will be guaranteed if the information about the topic i is included in the help sheet. For any topic i, there are **H[i]** lines available and the student can either write all of these **H[i]** lines in their help sheet or do not write anything about that topic. The task at hand is to select the topics whose lines you want to include in the help sheet such that the marks guaranteed is maximized.

For this question you are required to design a bottom-up dynamic programming algorithm (iterative algorithm) that calculates the maximum marks guaranteed for the above scenario.

For each topic i ($1 \leq i \leq n$), the number of questions and the number of lines for the help sheet are available at the ith index of arrays **Q** and **H** respectively. Also note that each question carries equal marks.

Basic Idea (in Plain English):

There are two dimensions (parameters) of this problems, namely, # of lines (L), and # of topics (n). The question is about maximum marks using (up to) n topics and (at most) L lines. So, we would solve this problem in a bottom-up fashion along these two dimensions.

[**Rubric**] **3 Marks**

[**Optional**] Following recurrence would be used to solve the problem:

$$Marks[i, j] = \max(Marks[i - 1, j], Marks[i - 1, j - H[i]] + Q[i])$$

Pseudocode:

```
MaxMarks(L, H, Q)

1    n = H.length
2    let Marks[0…n, 0…L] be a new array
3    for i = 0 to n
4        Marks[i,0] = 0
5    for j = 1 to L
6        Marks[0,j] = 0
7    for i = 1 to n
8        for j = 1 to L
9            if H[i]>j
10                Marks[i,j]=Marks[i-1,j]
11            else Marks[i,j]=max(Marks[i-1,j],
                                  Marks[i-1,j-H[i]]+Q[i])
12   return Marks[n,L]
```

Mathew is a passionate mountaineer. This season, he climbed **n** mountains labelled as mount_1, mount_2, …, mount_n. He started his climbing adventure at mount_start ($1 \leq$ **start** $\leq$ **n**) and finished it at mount_end ($1 \leq$ **end** $\leq$ **n**).

Write an algorithm that prints the mountain labels in the order in which the mountains were climbed by Mathew. Following information would be available to your algorithm as input:

The variables **start** (the mountain from where he started his adventure) and **end** (the mountain where he finished his adventure), and the 2-D array **Highest**$[1 \dots n, 1 \dots n]$.

- **Highest**$[i, j] = k$ if mount_k is the highest mountain climbed by Mathew while going from mount_i to mount_j.
- **Highest**$[i, j] =$ **NIL** if
  - ❖ Either, he climbed mount_j immediately after mount_i (hence, no intermediate mountain to climb)
  - ❖ Or, he climbed mount_j before mount_i (hence going from mount_i to mount_j is not a possibility)

Your algorithm MUST finish in $O(n)$ time.

Basic Idea in Plain English:

We first print the starting mountain, then we **recursively** print all the intermediate mountains, finally we print the ending mountain.

[**Rubric**] **3 Marks** if it is mentioned that the intermediate mountains are printed *recursively*.

[**Rubric**] **1 Mark** otherwise (provided there is a mention of printing the starting mountain and ending mountain)

Pseudocode:

```
PrintTour(start, end, Highest)

    print "mount_"start

    PrintIntermediate(start, end, Highest)

    print "mount_"end
```

[Rubric] **2 Marks**

```
PrintIntermediate(i, j, Highest)

    if Highest[i,j]≠NIL

        PrintIntermediate(i, Highest[i,j], Highest)

        print "mount_"Highest[i,j]

        PrintIntermediate(Highest[i,j], j, Highest)
```

[Rubric] **5 Marks**

Question 5: [4+3+3] Marks

a) In the radix sort algorithm, what possibly may go wrong if the sorting algorithm used by the radix sort as subroutine is not a stable sorting algorithm? Explain with the help of an example. (**No partial marks for this part**)

It may not correctly sort the input array. Let $A = \langle 55,53 \rangle$, then after the first pass, we have $A[1] = 53$ and $A[2] = 55$ (as $3 < 5$). However, in the second pass there would be a tie between two occurrences of 5. If the subroutine is not a stable sorting algorithm, it may pick 5 corresponding to 55 first and the final output would be $A = \langle 55,53 \rangle$, which is not correct.

b) For the following algorithm, write down the recursive expression for $T(n)$. The first call is ALGO(A, 1, n), where A is an array of size n.

$ALGO(A, i, j)$

```
1   ret = i
2   if (i ≤ j) then
3       ret = ALGO(A, i, (i + j)/2)
4       k = (i + j)/2
5       while (k ≤ j)
6           ret = ALGO(A, k, j) + ret − k
7           k = k + 1
8   return ret
```

2 Marks

1 Mark

$$T(n) = T\left(\frac{n}{2}\right) + \sum_{m=1}^{\frac{n}{2}} T(m)$$

c)

i.   If $T(n) = 8\,T(n/3) + \Theta(n^2)$, then $T(n) = \Theta(n^2)$.

ii. If $T(n) = 9\,T(n/3) + \Theta(n^2)$, then $T(n) = \Theta(n^2 \lg n)$.

iii. If $T(n) = 10\,T(n/3) + \Theta(n^2)$, then $T(n) = \Theta(n^{\log_3 10})$.