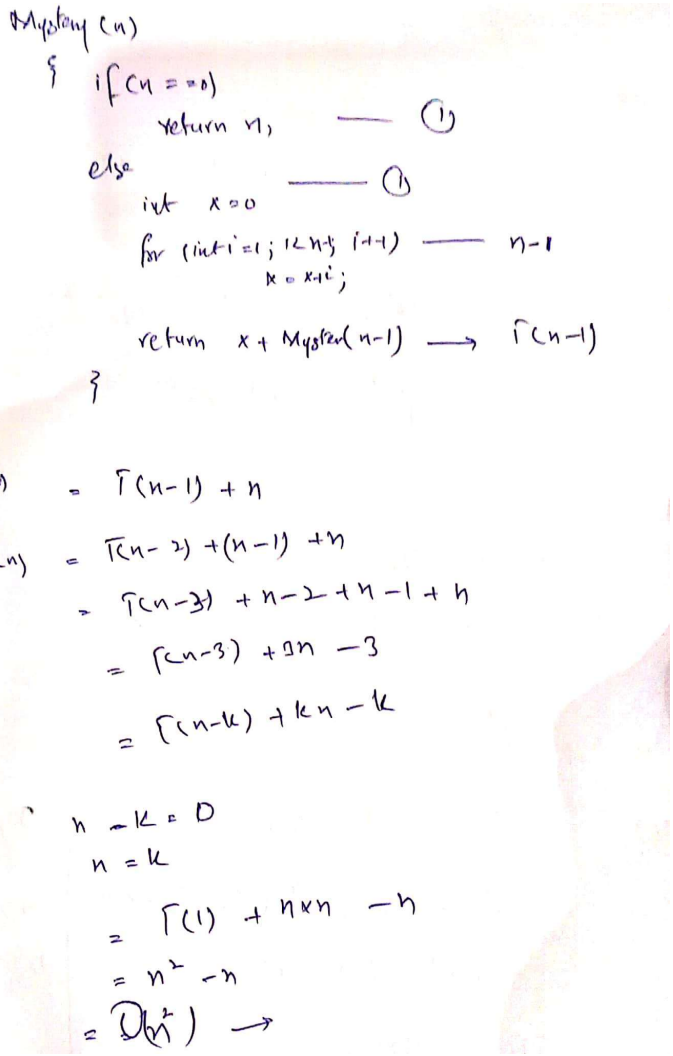
Part A [35 Marks]

Q. No. 1. Devise the recurrence relation for the following recursive function and then solve it for writing the time complexity in Big-O or Theta notation without using the Master theorem. Assume that there is no error/ bug in the code. Show complete working. **[3 Marks]**

```
int Mystery(int n)
{
        if(n==0)
        {
                return n;
        }
        else
        {
                int x=0;
                for(int i=1; i < n-1; i++)
                        x=x+i;
                return x + Mystery(n-1);
         }
}
```

Answer:



Q. No. 2. Solve the following recurrence relation without using master theorem. **[3 Marks]**

$$T(n) = 9\,T(n/3) + n^2$$

$$\begin{aligned}
T(n) &= 9\,T(\tfrac{n}{3}) + n^2 \\
&= 9\,(9\,T(\tfrac{n}{3^2}) + (\tfrac{n}{3})^2) + n^2 \\
&= 9^2\,T(\tfrac{n}{3^2}) + 2n^2 \\
&= 9^2\,(9\,T(\tfrac{n}{3^3}) + (\tfrac{n}{3^2})^2) + 2n^2 \\
&= 9^3\,T(\tfrac{n}{3^3}) + 3n^2 \\
&\ldots \\
&= 9^k\,T(\tfrac{n}{3^k}) + kn^2
\end{aligned}$$

If $3^k = n$, then recursion the will reach base case (i.e $T(\frac{n}{3^k})$ = T(1), which is constant), so

$k = log_3 n$

$$\begin{aligned}
T(n) &= 9^{log_3 n} + log_3 n \times n^2 \\
&= n^2 + n^2 log_3 n \\
&= \theta\,(n^2 log_3 n)
\end{aligned}$$

Q. No. 3. Suppose that you are required to choose one of the following 3 algorithms: **[6 Marks]**

1. Algorithm A solves the problem of size n by dividing it into 5 sub-problems of size n/2, recursively solves each sub-problem, and then combines the solutions in linear time.
2. Algorithm B solves the problem of size n by recursively solving two sub-problems of size n − 1 and then combines the solutions in constant time.
3. Algorithm C solves the problem of size n by dividing it into nine sub-problems of size n/3, recursively solves each sub-problem, and then combines the solutions in $O(n^2)$ time.

What is the asymptotic time complexity of each algorithm? Which one would you choose and why?

Answer

1. **Using case 1 of Master Theorem; $T(n) = \Theta(n^{2.32})$**
2. **$O(2^n)$**
3. **Using case 2 of Master Theorem; $T(n) = \Theta(n^2 lgn)$**

Algorithm C is to be chosen as it has the best performance.

Q. No. 4. Is there any difference between negative weights and negative cycle? How can we detect a negative cycle in a graph? **[2 Marks]**

Answer

Yes, a negative cycle is a cycle in a graph where the sum of the weights of all the edges in the cycle is negative. We can detect it by using the Bellman-Ford Algorithm.

Q. No. 5. Greedy-Activity-Selector. **[4 Marks]**
Consider the following set of activities S={(6,9),(1,10),(2,4),(1,7),(5,6),(8,11),(9,11)}.

For the first activity (6,9), 6 is the start time and 9 is the finish time. Select the largest possible number of tasks from S that can be completed without incompatibilities (two activities are incompatible iff they overlap) by using the Greedy-Activity-Selector algorithm learnt in the class. Show all steps (i.e. the complete trace of the algorithm).

Third set of activities is the answer.

The following schedules are all allowable

$(1, 10)$
$(1, 7), (8, 11)$
$(2, 4), (5, 6), (9, 11)$

Q. No. 6. [15 Marks]
Find the optimal parenthesizing for multiplying the given matrices A x B x C x D x E, by considering all possibilities. Show step by step computation to fill up table M and S. Use table S to reconstruct the optimal solution.

A=3x1, B=1x4, C=4x3, D= 3x2, E=2x4

M-Table

0	**12**	**21**	**24**	**38**
	0	**12**	**18**	**26**
		0	**24**	**40**
			0	**24**
				0

S-Table

0	**1**	**1**	**1**	**1**
	0	**2**	**3**	**4**
		0	**3**	**4**
			0	**4**
				0

Reconstructed Optimal Solution

A (((B C) D) E)

Q. No. 7. Choose either True or False/ [2 Marks]
1. In the recursive definition of the Floyd–Warshall algorithm:

$$d^{(k)}uv = \min\{d^{(k-1)}uv, d^{(k-1)}uk + d^{(k-1)}kv\},$$

$d^{(k)}uv$ represents the length of the shortest path from vertex u to vertex v that contains at most k edges.

a) True

b) False

2. Negating all the edge weights in a positive-edge weighted undirected graph G and then finding the minimum spanning tree gives us the maximum-weight spanning tree of the original graph G.
   a) True
   b) False

Part B [35 Marks]

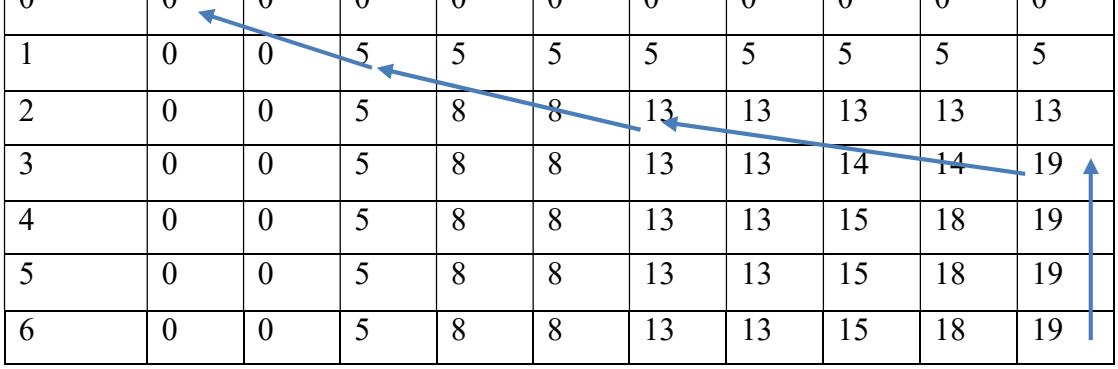
Q. No. 1. (0/1 Knapsack using DP) [15 Marks]

You have a knapsack with a weight (W) capacity of 9. There are 6 items available, each with its own weight and value:
Item 1: Weight = 2, Value = 5
Item 2: Weight = 3, Value = 8
Item 3: Weight = 4, Value = 6
Item 4: Weight = 5, Value = 10
Item 5: Weight = 7, Value = 12
Item 6: Weight = 9, Value = 14

Using dynamic programming, you are required to determine the items that maximize the total value while not exceeding the weight capacity of the knapsack. Complete the following table and show how you select the items that will give the maximum value.

Solution:

Capacity Items	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	5	5	5	5	5	5	5	5
2	0	0	5	8	8	13	13	13	13	13
3	0	0	5	8	8	13	13	14	14	19
4	0	0	5	8	8	13	13	15	18	19
5	0	0	5	8	8	13	13	15	18	19
6	0	0	5	8	8	13	13	15	18	19



Selected Items:
Item 3, Item 2, and Item 1.

Q. No. 2. Time Complexity Analysis [2+2=4 Marks]

Write the asymptotic running time of the following codes. Assume that there is no error/ bug in the codes. Express your answer as a function of n using Θ-notation.

	Algorithm	Time Complexity

1.	`int func(int n)`
	`{`
	`  int count = 0;`
	`  for (int i = 0; i < n; i++)`
	`  for (int j = i; j > 0; j--)`
	`  count = count + 1;`
	`  return count;`
	`}`
		$\Theta(N^2)$

2.	`void func(int n, int arr[])`
	`{`
	`  int i = 0, j = 0;`
	`  for (; i < n; ++i)`
	`  while (j < n && arr[i] < arr[j])`
	`  j++;`
	`}`
		Θ(N) = Variable j is not initialized for each value of variable i. So, the inner loop runs at most n times.

Q. No. 3. Asymptotic Growth (4+2=6 Marks)

i) For each pair of functions $f(n)$ *and* $g(n)$ given below:

- Write Θ in the box if $f(n) = \Theta(g(n))$
- Write O in the box if $f(n) = O(g(n))$
- Write Ω in the box if $f(n) = \Omega(g(n))$
- Write X in the box if none of these relations holds

If more than one such relation holds, write only the strongest one. No explanation needed. No partial credit.

O, Θ, Ω, or X	*f(n)*	*g(n)*
O	n^2	n^3
Ω	$n \lg n$	n
Θ	1	$2 + \sin n$
Ω	3^n	2^n
Θ	4^{n+4}	2^{2n+2}

O	$n \lg n$	$n^{101/100}$
Θ	$\lg \sqrt{10\, n}$	$\lg n^3$
O	$n\,!$	$(n+1)\,!$

ii) State whether each statement below is True or False.

a. If $f(n) = \Theta(g(n))$ *and* $g(n) = \Theta(h(n))$, then $h(n) = \Theta(f(n))$

True. Θ is transitive.

b. If $f(n) = O(g(n))$*and* $g(n) = O(h(n))$, then $h(n) = \Omega(f(n))$

True. O is transitive, and $h(n) = \Omega(f(n))$is the same as $f(n) = O(h(n))$

Q. No. 4. STRING MATCHING (2 x 5= 10 Marks)

	Questions	Answers
1.	In String matching algorithms (studied in the class), which algorithm takes the longest pre-processing time?	Finite automaton takes O(***m***|∑|) pre-processing time.
2.	In Rabin Karp algorithm, how can you reduce the spurious hits?	By increasing the value of 'q' while taking running hash using mod q.
3.	What is the basic formula applied in Rabin-Karp Algorithm to get the pre-processing computation time as Θ(m)?	The pattern can be evaluated in time Θ(m) using Horner's rule: p = P[m] + 10(P[m-1] + 10(P[m-2] +…+ 10(P[2]+10P[1])…)).
4.	Can string matching algorithms be used for discovering plagiarism in a sentence? (Yes/No only)	Yes
5.	How is KMP an improvement over Naïve String matching algorithm?	KMP computes a prefix (π) table that contains info. About how pattern matches itself and it can be used to avoid testing useless shifts that the naïve algorithm does.

Part C [30 Marks]

Q. No. 1. Suppose that you have an array A of positive integers. You want to find the minimum number of coins needed to make change for a given amount of money. You have an unlimited supply of coins with denominations (values) d1, d2, ..., dn. Write a recursive definition and provide a dynamic programming solution to find out the minimum number of coins that are needed to make a change for the given amount.

For example, suppose that the array of denominations is {1, 3, 4} and the minimum number of coins are needed to make change for the amount 6. The optimal solution is to use two coins, each with a denomination of 3. Note that this is just an example and your solution should work for a generic case. **[14 Marks]**

Answer:

$d_1=1$ ensures that we can make any amount using these coins.

$1=d_1<d_2<d_3<...<d_n$

Let's say M_x is the minimum number of coins needed to make the change for the value x.

Let's start by picking up the first coin i.e., the coin with the value d_1. So, we now need to make the value of $x-d_1$ and M_{x-d_1} is the minimum number of coins needed for this purpose. So, the total number of coins needed are $1+M_{x-d_1}$ (1 coin because we already picked the coin with value d_1 and M_{x-d_1} is the minimum number of coins needed to make the rest of the value).

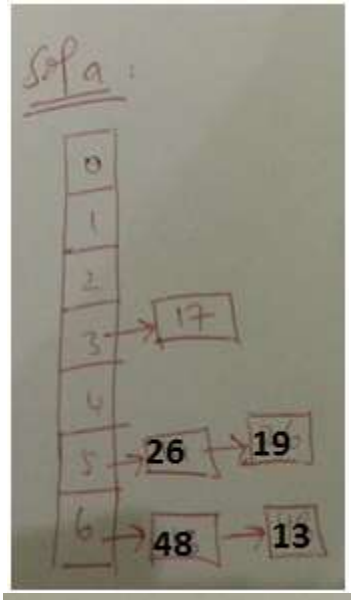
Recursive definition

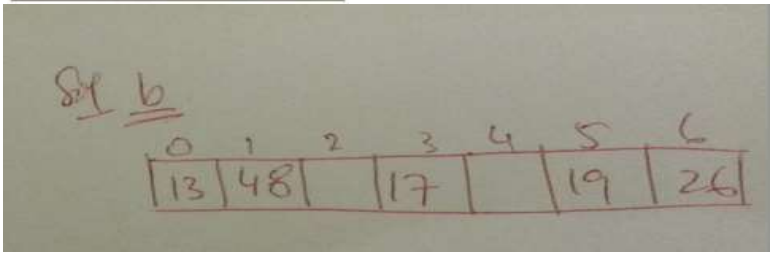
$$M_x = \begin{cases} \min_{i:d_i \le x}\{M_{x-d_i}+1\}, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \end{cases}$$

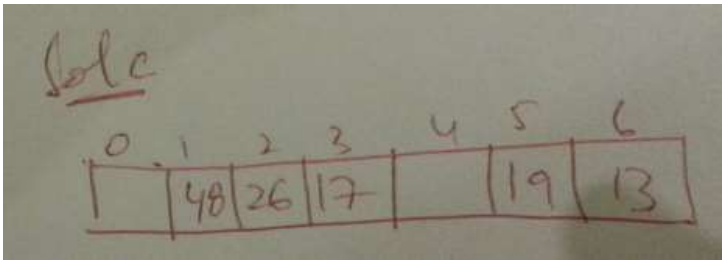
Q. No. 2. Short Questions [2+2+2=6 Marks]

Consider a hash table of size 7 with hash function h(k) = k mod 7. Draw the table that results after inserting in the given order, the following values: 19, 26, 13, 48, 17 for each of the three scenarios below:

a) When collisions are handled by chaining
b) When collisions are handled by linear probing
c) When collisions are handled by double hashing using a second hash function
h'(k) = 5 – (k mod 5)







Q. No. 3. Short Questions [5×2=10 Marks]

	Questions	Answers
1.	What is the worst case time complexity to find the minimum element in a min-heap?	$\theta(1)$ – Minimum value id is always at the root node in min-heap
2.	What is the worst case time complexity to find the maximum element in a min-heap?	$\theta(n)$ – The maximum element could be anywhere in the bottom level
Choose the best Sorting technique for each of the following MCQs.		
3.	You are running a library catalog. You know that the books in your collection are almost in sorted ascending order by title, with the exception of one book which is in the wrong place. You want the catalog to be completely sorted in ascending order. 1. Insertion Sort 2. Merge Sort 3. Radix Sort 4. Heap Sort 5. Counting Sort	*Insertion sort will run in O(n) time in this setting.*
4.	You are working on an embedded device (an ATM) that only has 4KB (4,096 bytes) of free memory, and you wish to sort the 2,000,000 transactions' withdrawal history by the amount of money withdrawn (discarding the original order of transactions). 1. Insertion Sort 2. Merge Sort 3. Radix Sort 4. Heap Sort 5. Counting Sort	*Heap sort, because it is in-place.*
5.	To determine which of your Facebook friends were early adopters, you decide to sort them by their Facebook account ids, which are 64-bit integers. (Recall that you are super popular, so you have many Facebook friends.) 1. Insertion Sort 2. Merge Sort 3. Radix Sort 4. Heap Sort 5. Counting Sort	*Radix sort*