Question 1 [5 Marks]

For the given 5 items below, find the most valuable subset of the items that fits into the 0-1 knapsack of capacity 5 using dynamic programming technique?

item	weight	value
1	2	Rs. 15
2	3	Rs. 20
3	1	Rs. 12
4	2	Rs. 18
5	3	Rs. 25

Write recursive definition using Dynamic Programming technique for solving 0-1 Knapsack problem?

$$F(i, j) = \begin{cases} \max\{F(i-1, j), v_i + F(i-1, j-w_i)\} & \text{if } j - w_i \geq 0, \\ F(i-1, j) & \text{if } j - w_i < 0. \end{cases}$$

Fill the table using the recursive definition provided above for solving the 0-1 Knapsack problem;

Solution:

	i	0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2, v_1= 15$	1	0	0	15	15	15	15
$w_2 = 3, v_2= 20$	2	0	0	15	20	20	35
$w_3 = 1, v_3= 12$	3	0	12	15	27	32	35
$w_4 = 2, v_4= 18$	4	0	12	18	30	33	45
$w_5 = 3, v_4= 25$	5	0	12	18	30	37	45

Show the complete trace of the items selected with the help of the table computed (above) using recursive definition of dynamic programming technique for 0-1 Knapsack.

Solution:
F(5,5)=F(4,5) item 5 is not selected,
F(4,5)>F(3,5) item 4 is selected,

F(3, 5-2)
F(3,3)>F(2,3) item 3 is selected
F(2,3-1)

F(2,2)=F(1,2) item 2 is not selected
F(1,2)>F(0,2) item 1 is selected

Show final list of selected items here;
Solution: Item 4, Item 3, Item 1

Question 2 [5 Marks]

Determine an LCS (Longest Common Subsequence) of X={bdcabdab} and **Y={abcdb}.**
Construct two Tables "c" and "b" for finding the LCS. Table "c" is for finding the length of the longest common subsequence and table "b" is for finding the LCS. Also, write the recursive definition for finding LCS using dynamic programming technique.

Recursive Definition:

$$c[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ c[i-1,j-1]+1 & \text{if } x[i]=y[j], \\ \max\{c[i-1,j],\, c[i,j-1]\} & \text{otherwise.} \end{cases}$$

Table c: [It computes the length of the Longest Common Subsequence]

Solution

```
| abcdb
-+------
 |000000
b|001111
d|001122
c|001222
a|011222
b|012223
d|012233
a|012233
b|012234
```

Table b: [It keeps track of the LCS; hint: table with arrows]

Solution:

```
|  abcdb
-+------
 |000000
b|0|\---
d|0|||\\
c|0||\--
a|0\-|||
b|0|\||\
d|0|||\|
a|0|||||
b|0||||\
```

Longest Common Subsequence:
Bcdb

Question 3 [1+2+1=4 Marks]

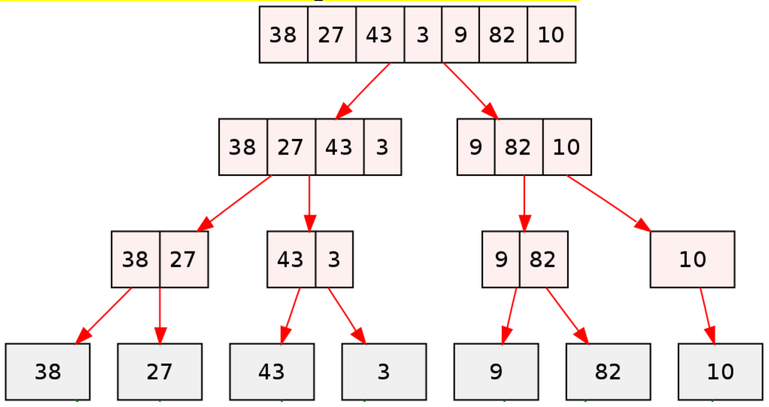
A. Explain under what circumstances dynamic programming technique is better then divide and conquer technique?

Solution/;
Divide-and-conquer algorithms partition the problem into disjoint subproblems, solve the subproblems recursively, and then combine their solutions to solve the original problem. In contrast, dynamic programming applies when the subproblems overlap—that is, when subproblems share subsubproblems. In this context, a divide-and-conquer algorithm does more work than necessary, repeatedly solving the common subsubproblems. A dynamic-programming algorithm solves each subsubproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time it solves each subsubproblem.

B. Show why divide and conquer technique is a suitable approach instead of a dynamic programming technique in order to sort numbers using merge sort? Provide your reasoning with the help of a complete example.

Solution

Merge Sort cannot use Dynamic Programming, because **the subproblems are not overlapping in any way**
Following merge sort tree shows each sub problem is distinct;



C. What happens when dynamic programming technique is applied to a problem? Discuss in terms of time and space complexities?

Solution:
DP uses the memorization technique; it always stores the previously calculated values. Due to this, the time complexity is decreased but the space complexity is increased.

Question 4 [4 Marks]

Analyze the given hash function based on the four properties of a good hash function *(sequence of hash properties does not matter but property should be mentioned explicitly)*. Provide a justification against each of the four properties and also show a complete step by step dry run of the string “ab”. Assume that there is no error in the code. The ASCII value of character a is 97.

```
int fun(char *p)
{
        char  m = 7;
        char h = 0x00;
        char g;
        while ( *p != NULL )
        {
             h = ( h << 1 ) + ( *p );
             if ( g = h & 0xF0 )
             {
                    h = h ^ g;
             }
             p = p+1;
        }
        return h % m;
}
```

Following are the details of the function:

- Each number is of one byte and 0x.. represents hex number.
- ‘m’ is the size of hash table, and it is always a prime number.
- “<<” is a bitwise left shift operator.
- “^” is bitwise XOR operator.
- p is the received actual key.

Rule 1:
The hash value is fully determined by the data being hashed.
Satisfied because hash depends on the input string.
Rule 2:
The hash function uses all the input data.
Satisfied because all input characters are used to make a hash value.
Rule 3:
The hash function uniformly distributes the data across the entire set of possible hash values.
Satisfied because left shift operation mutates the previous value of ‘h’ in each iteration and hence the distribution is approximately uniform due to almost equal probability of occurrence of each hash value.
Rule 4:
The hash function generates very different hash values for similar strings.
Satisfied because h(ab) is not equal to h(ba) and slight change in strings mostly produces a different hash value.

Dry Run:

Rough Work:

Test ab and ba
 	
ab:
 	
h= 00000000b + 01100001b = 01100001b
g= 01100001b & 11110000b = 01100000b
h= 01100001b XOR 01100000b = 00000001b
 	
h= 00000010b + 01100010b = 01100100b
g= 01100100b & 11110000b = 01100000b
h= 01100100b XOR 01100000b = 00000100b
return 4d % 7d
 	
h(ab)= 4
 	
ba:
 	
h= 00000000b + 01100010b = 01100010b
g= 01100010b & 11110000b = 01100000b
h= 01100010b XOR 01100000b = 00000010b
 	
h= 00000100b + 01100001b = 01100101b
g= 01100101b & 11110000b = 01100000b
h= 01100101b XOR 01100000b = 00000101b
 	
return 5d % 7d
 	
h(ba)= 5	

Question 5 [4 Marks]

Perform the amortized analysis of the dynamic hash tables using the aggregate and accounting method. Use proper methodology to find the time complexity of insertion into a hash data structure.

Aggregate Method:

i	**$Size_i$**	**C_i**
1	**1**	1
2	2	2
3	4	3
4	4	1
5	8	5
6	8	1
7	8	1
8	8	1
9	16	9
10	16	1
11	16	1
12	16	1
13	16	1
14	16	1
15	16	1
16	16	1
17	32	17

18	32	1
19	32	1
20	32	1

Cost of n insertions = θ(n)

Accounting Method:
Note: The amortized cost (C_i^) of first insertion is 2 and all other insertions is 3.

i	$Size_i$	C_i	C_i^	$bank_i$
1	**1**	1	2	1
2	2	2	3	2
3	4	3	3	2
4	4	1	3	4
5	8	5	3	2
6	8	1	3	4
7	8	1	3	6
8	8	1	3	8
9	16	9	3	2
10	16	1	3	4
11	16	1	3	6
12	16	1	3	8
13	16	1	3	10
14	16	1	3	12
15	16	1	3	14
16	16	1	3	16
17	32	17	3	2
18	32	1	3	4
19	32	1	3	6
20	32	1	3	8

Cost of n insertions = θ(n)

Question 6 [4 Marks]

Suppose you have to drive from Islamabad to Lahore on M2 motorway by a car. The gas tank of your car is initially full, and it holds enough gas to travel 'm' miles. You have a direction map with distances between all Gas Stations along the route.
Let $d_1<d_2<\ldots<d_n$ be the locations of all Gas Stations along the route, where d_i is the distance from Islamabad to the Gas Station. The distance between neighboring Gas Stations is at most 'm' miles and tank must be filled completely if a car is stopped at any Gas Station.
Design a Greedy Algorithm (pseudocode) to get an optimal solution having minimum number of Stops where car should stop to get refilled with Gas.
Demonstrate with a simple example of 10 gas stations and m=40, that your solution is optimal.

Pseudocode	**Rough Work**

d[ ] = 0, 10, 11, 12, 39, 40, 53, 60, 80, 100, 121
m = 40
 	
FOR (i: 1 to d.length-1)   // considering array from 0th index
    diff = d[i] - d[i-1]
    m = m - diff	

diff = d[i+1] - d[i]
 	
IF (m – diff < 0)
m = 40
print d[i]
END IF
END FOR	

Example: Example: Assume the following distances d_1=10 d_2= 11 d_3= 12 d_4= 39 d_5= 50 d_6= 53 d_7= 60 d_8= 70 d_9= 80 d_{10}= 91 Greedy Solution= d_4, d_8 = 2 stops Optimal Solution= 2 stops	

Question 7 [7*2=14 Marks]

Part A. Given an array X[1...n] of n numbers, we want to compute an array A such that A[i] is the average of elements X[1]...... X[i] for i = 1..... n, that is

$$A[i] = \frac{(X[1] + X[2] + ... + X[i])}{i}$$

Analyze the algorithm given below line by line and find out the asymptotic run time.

```
Algorithm $prefixAverages(X[1...n])$
// input: an n-element array X[1 ... n]
// output: an n-element array A[1... n] such that
// A[i] is the average of elements X[1],....,X[i]
1) for $i \leftarrow 1$ to $n$ do
2)       $sumX \leftarrow 0$
3)       for $j \leftarrow 1$ to $i$ do
4)                $sumX \leftarrow sumX + X[j]$
5)       $A[i] \leftarrow sumX/i$
6) output $A[1....n]$
```

1+2+3+......+N = N(N+1)/2 = O(N^2)

Re-write the algorithm for the same problem, ensuring that its asymptotic run time complexity is of the order n (linear time complexity).

```
SUM = 0
FOR i = 1 to n DO
    SUM = SUM+ X[i]
    A[i] = SUM/i
END FOR
```

Part B. HEAPSORT, as we studied it, puts its output in an array and sorts in ascending order. Explain how to modify HEAPSORT so that it creates a linked list in descending order. Your modifications should be as efficient as possible and not use any memory that is not necessary. Please note that you do not need to write algorithm and your explanation is enough.

We want the first element extracted to be the smallest, so we use a min heap instead of a max heap. At each pass, insert the node that was the heap's root at the head of the linked list. (Insertion is thus always $\Theta(1)$ since it's just pointer assignments.)

Part C. Write recurrence equation for Strassen's matrix multiplication algorithm. Please note that the recurrence equation for the standard matrix multiplication algorithm is: $T(n) = 8T(n/2) + \Theta(n^2)$.

$T(n) = 7T(n/2) + \Theta(n^2)$

Part D. Compute the Knuth/ Morris/ Pratt prefix function (Pi vector) for pattern aabaabcab

n	1	2	3	4	5	6	7	8	9
pi(n)	**0**	**1**	**0**	**1**	**2**	**3**	**0**	**1**	**0**

Part E. Write Dijkstra's algorithm (in pseudocode form) of "Single Source Shortest Path" problem using Linear array as a queue (in place of binary min-heap) and do its asymptotic time complexity analysis using Big-O notation.
Using array as a queue, EXTRACT_MIN takes O(V) time (because of a loop to find min. value) and there are |V| such operations. Therefore, a total time for EXTRACT_MIN in while-loop is $O(V^2)$.
Since the total number of edges in all the adjacency list is |E|. Therefore for-loop iterates |E| times with each iteration taking O(1) time (due to direct array access of the vertex index). Hence, the running time of the algorithm with array implementation is $O(V^2 + E) = O(V^2)$.

Part F. Suppose that in a graph for all edges e ε E and w(u,v)=1 for (u,v) ε E. Can Dijkstra's algorithm be improved to find the shortest path? What will be the asymptotic time complexity of the improved version?
In this case, BFS can be used to find the shortest path. Time Complexity is $\Theta(V+E)$.

Part G. When do we need to use Bellman-Ford algorithm instead of Dijkstra's algorithm for "Single Source Shortest Path" problem in a graph? Explain with the help of a graph example where Dijkstra's algorithm does not provide correct results and instead we need to use Bellman-Ford algorithm.
In case a graph contains a negative weight cycle.