UIU QUESTION BANK

MID-TERM QUESTION SOLUTIONS

DATA STRUCTURE AND ALGORITHMS II

CSE 2217

SOLUTION BY

NURUL ALAM ADOR

UPDATED TILL ***SPRING 2024***

nurulalamador.github.io/**UIUQuestionBank**

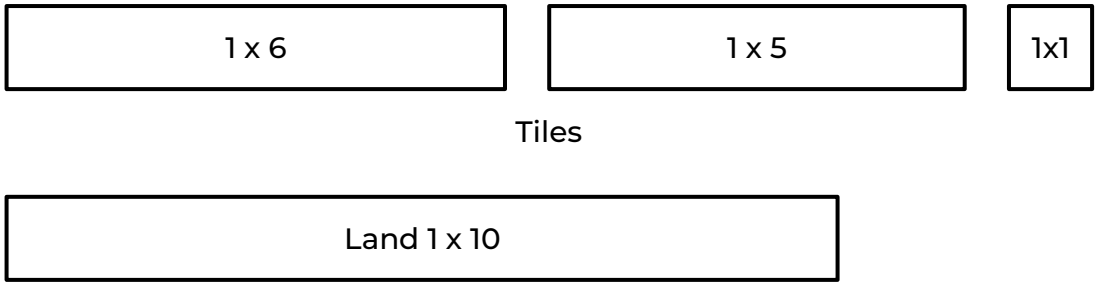
Index

Trimester	Page
Spring 2024	3
Fall 2023	11
Summer 2023	19
Spring 2023	29
Fall 2022	37

Important Note

The simulation of various kinds of algorithms (such as maximum sum subarray and minimum-maximum value) could vary by teacher. In this solution, we follow **Md. Tamzid Hossain** sir's simulation technique.

1. a) Imagine you have a land with an area of 1x10 m^2, and you want to cover it using some tiles. There are three types of tiles available which varies in size: 1x1 m^2, 1x5 m^2 and 1x6 m^2 (See figure). Each type has infinite amount of supplies, so you never run out of tiles. Now, there are many ways that you can cover that land using these tiles, but you want to use as few tiles as possible (Tiles are expensive!)



Using ***Dynamic Programming*** method, find the minimum number of tiles that can cover your land. Which tiles should we use? Describe your solution with detailed calculation.

Solution:

Given,

Total Land $= 1 \times 10\ m^2 = 10\ m^2$

Available Tiles, $T_i = \{T_1, T_2, T_3\}$
$= \{1 \times 6\ m^2, 1 \times 5\ m^2, 1 \times 1\ m^2\}$
$= \{6\ m^2, 5\ m^2, 1\ m^2\}$

Now,

Total Land, $L\ (m^2)$ →

Tiles, $T\ (m^2)$ ↑

	0	1	2	3	4	5	6	7	8	9	10
1	0	1	2	3	4	5	6	7	8	9	10
5	0	1	2	3	4	1	2	3	4	5	2
6	0	1	2	3	4	1	1	2	3	4	2

∴ Minimum tiles needed = 2
Tiles should be use: 2 tiles of 1 x 5 m^2

1. b) Suppose your wallet has capacity to hold only 8 grams of gold coins, and your best friend just offered you 4 gold coins from his own collection. The weights and the values of the coins are as follows: [5g, 4g, 6g, 3g] and [110$, 100$, 120$, 90$]. Using ***Dynamic Programming,*** determine which coins you should take so that your total gain is maximized. Keep in mind that you cannot carry more than 8 grams of gold coins.

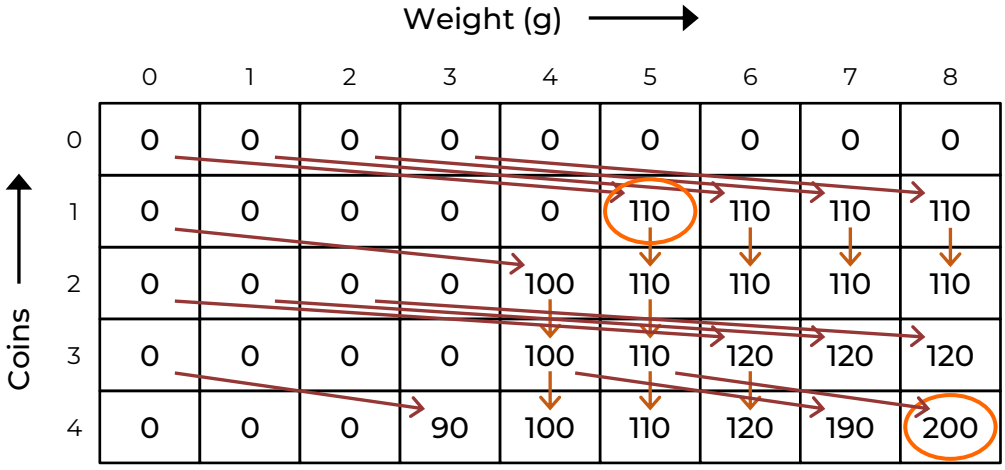
Solution:

Here,

Capacity = 8 g

Coin No	1	2	3	4
Values ($)	110	100	120	90
Weight (g)	5	4	6	3

Now,



∴ Total maximize gain = 200$

Coin should take = 1st (110$) and 4th (90$)

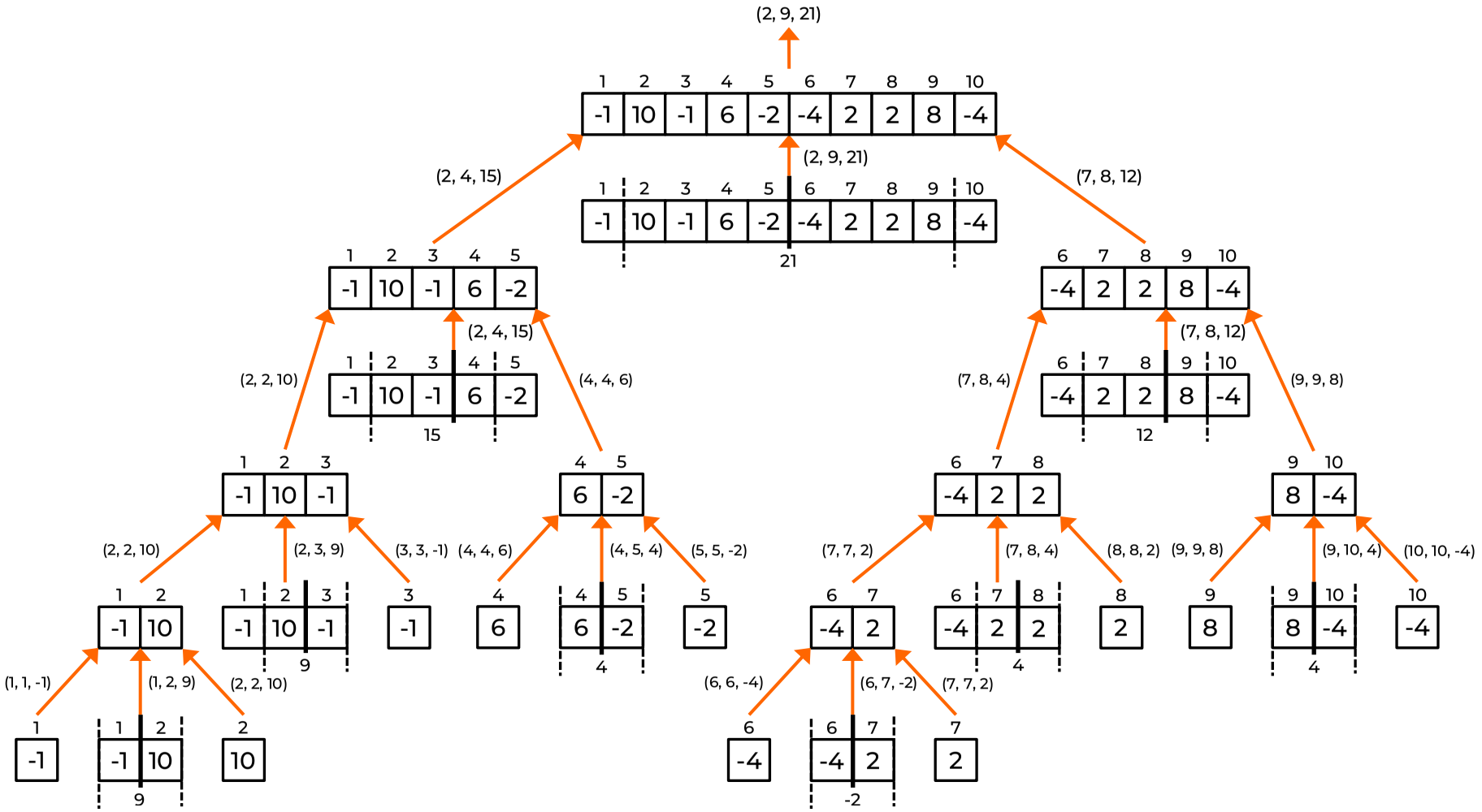
2. a) Consider the following array

-1	10	-1	6	-2	-4	2	2	8	-4

Find the Maximum-sum subarray using divide-and-conquer. You must show the recursion tree and clearly mention left, right and crossing sum for each tree node.

Solution:

[P.T.O]



∴ Maximum continuous subarray is:

2	3	4	5	6	7	8	9
10	-1	6	-2	-4	2	2	8

(Maximum Sum = 21)

2. b) Design an algorithm using the divide and conquer approach that counts the number of elements in an array that end with the digit '1'. The algorithm should make use of the modulus operator to determine if an element ends with '1'. Include the pseudocode for your algorithm, clearly defining the base case and detailing all necessary steps and calculations.

Solution:

The pseudocode has been written below:

```
CountEndsWithOne(arr[], left, right) {
    if (left == right) {
        if(arr[left] % 10 == 1){
            return 1;
        }
        else {
            return 0;
        }
    }

    else {
        mid = (left + right) / 2;

        leftCount = countEndsWithOne(arr, left, mid);
        rightCount = countEndsWithOne(arr, mid + 1, right);

        return leftCount + rightCount;
    }
}
```

Base Case

If it is not base case, dividing the array in two nearly equal halves and merging their return result

3. a) Using recursion tree method or the Master Theorem find out a good asymptotic upper bound on the following recurrence:

T(n) = 3T(n/2) + O(n)

Solution:

Given,

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

Comparing with base equation of Master method $T(n) = aT\left(\frac{n}{b}\right) + O(n^k \log^p n)$,

Here,

$a = 3$

$b = 2$

$k = 1$

$p = 0$

$b^k = 2^1 = 2$

Since $a > b^k$, therefore this recurrence belongs to case 1 of Master Theorem.

∴ Time complexity, $T(n) = O(n^{\log_b a})$

$= O(n^{\log_2 3})$

$= O(n^{1.58})$

∴ Asymptotic upper bound of given recurrence is $O(n^{1.58})$.

3. b) Consider the following function which takes two arrays (A and B) and their lengths (n and m) as inputs respectively.

```
void func1 (int A[], int B[], int n, int m) {
    int sumA = 0;
    for(int i=0; i<n; i++) {
        sumA = sumA + A[i];
    }

    int sumB = 0;
    for(int i=0; i<m; i++) {
        sumB = sumB + B[i];
    }

    int count = 0;

    for(int i=0; i<n; i++) {
        for(int j=0; j<m; j++) {
            if(A[i]*A[i]>B[i] || sumA>sumB) {
                count++;
            } else {
                break;
            }
        }
    }
}
```

Now determine the following:

i. The exact-cost equation for the running-time.
ii. The best case and worst case running time using the Big - Oh notation
iii. Examples of best case and worst case inputs □

Solution:

Line	Worst Case	Best Case
2	1	1
3	$n+1$	$n+1$
4	n	n
7	1	1
8	$m+1$	$m+1$
9	m	m
12	1	1

14	$n+1$	$n+1$
15	$nm+n$	n
16	nm	n
17	nm	0
19	0	n

i) Time complexity, $T(n) = 1+n+1+n+1+m+1+m+1+n+1+nm+n+2nm$
$= 3nm+4n+2m+6$

$\therefore$ Exact-cost equation is $3nm+4n+2m+6$

ii) Best case, $T(n) = 1+n+1+n+1+m+1+m+1+n+1+n+n+n$
$= 6n+2m+6$
$= O(n+m)$

Worst case, $T(n) = 1+n+1+n+1+m+1+m+1+n+1+nm+n+nm+nm$
$= 3nm+4n+2m+6$
$= O(nm)$

iii) Example of best case inputs:

A[ ] = | 1 | 1 | 1 | 1 | B[ ] = | 10 | 10 | 10 | 10 | n=4 m=4

Example of worst case:

A[ ] = | 10 | 10 | 10 | 10 | B[ ] = | 1 | 1 | 1 | 1 | n=4 m=4

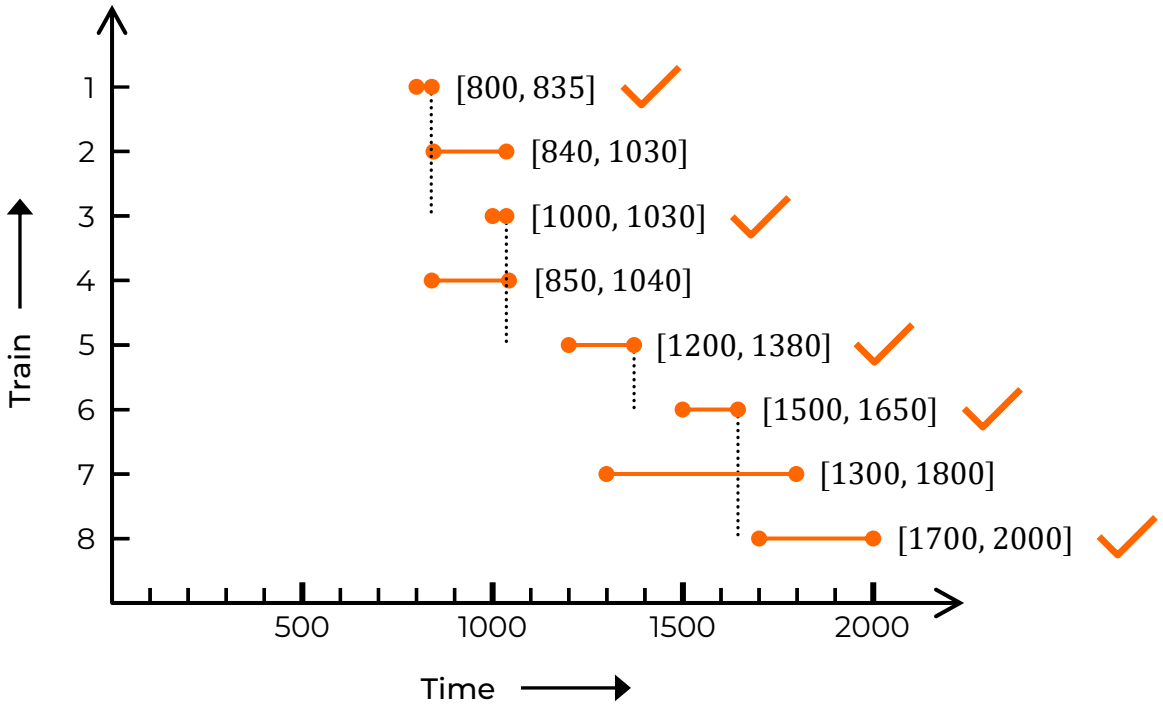
4. a) Given the arrival and the departure times (in minutes) of 8 trains for a railway platform, **find out the maximum number** of trains that can use the platform without any collision, using a **greedy algorithm**. There must be exist at least 10 minutes of safety break between the departure of one train and arrival of the next one.
[1000, 1030], [840, 1030], [850, 1040], [1700, 2000], [800, 835], [1300, 1800], [1500, 1650], [1200, 1380]
Explain your strategy very briefly and show detailed calculations. *No need to write pseudocode*. **Explain** why your solution satisfies the *optimal substructure property*. □

Solution:

Sorting train based on departure time:

Train, t_i	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
Arrival Time	800	840	1000	850	1200	1500	1300	1700
Departure Time	835	1030	1030	1040	1380	1650	1800	2000

Now,



∴ Maximum 5 trains [Scheduled for $[800, 835]$, $[1000, 1030]$, $[1200, 1380]$, $[1500, 1650]$ and $[1700, 2000]$] can use the platform without any collision.

Here, we use Activity Selection strategy. At first, we sort all the train in ascending order on the based on departure time. Then, we select the first train and select next train which arrival time is at least after 10 minutes from first train departure time.

Our solution satisfies the optimal substructure property because here an optimal solution to the entire problem can be constructed from optimal solutions to its sub-problems. By selecting the train with the earliest departure time, the remaining sub-problems can be solved using the same approach, leading to an overall optimal solution.

4. b) "Data encoded using Huffman coding is uniquely decodable" - is the statement true or false? **Justify** your answer. □

Solution:

The following statement is true. Huffman coding ensures unique decodability through its prefix property, where no code is a prefix of any other code. This property allows the encoded data to be parsed unambiguously from left to right without additional delimiters. During decoding, each bit sequence can be traced in the Huffman tree to a unique leaf node, ensuring that each sequence corresponds to a unique symbol. Consequently, Huffman coding guarantees that the encoded data can be uniquely and correctly decoded.

4. c) Suppose you have a cup with capacity 750 ml. The following table shows that there are 5 flavors of drinks. For each flavor, there is only d ml available, and each flavor of drink contains a total of s grams sugar. **At most how many grams of sugar** can you consume if you fill up the cup using **greedy approach**? You need to show all the steps of your calculation.

[P.T.O]

Flavor	Volume (d)	Total sugar (s)
Apple	320	35
Orange	220	20
Pineapple	240	40
Cranberry	200	30
Strawberry	280	25

Solution:

Given,

No of juices, $n = 5$

Capacity, $D = 750\ ml$

Index, i	**1**	**2**	**3**	**4**	**5**
Flavor	**Apple**	**Orange**	**Pineapple**	**Cranberry**	**Strawberry**
Sugar, s_i	35	20	40	30	25
Volume, d_i	320	220	240	200	280

Finding sugar per volume:

	1	2	3	4	5
	Apple	Orange	Pineapple	Cranberry	Strawberry
$\frac{s_i}{d_i}$	0.109	0.091	0.167	0.150	0.089

Sorting item based on $\frac{s_i}{d_i}$:

Pineapple > Cranberry > Apple > Orange > Strawberry

Adding item in knapsack:

i	Flavor	s_i	d_i	D	$x_i = \min\left(1, \frac{D}{d_i}\right)$	$D' = D - d_i x_i$
3	Pineapple	40	240	750	1	750-240 = 510
4	Cranberry	30	200	510	1	510-200 = 310
1	Apple	35	320	310	0.96875	310-(320*0.96875) = 0

$\therefore$ Maximum Sugar $= s_3x_3 + s_4x_4 + s_1x_1$

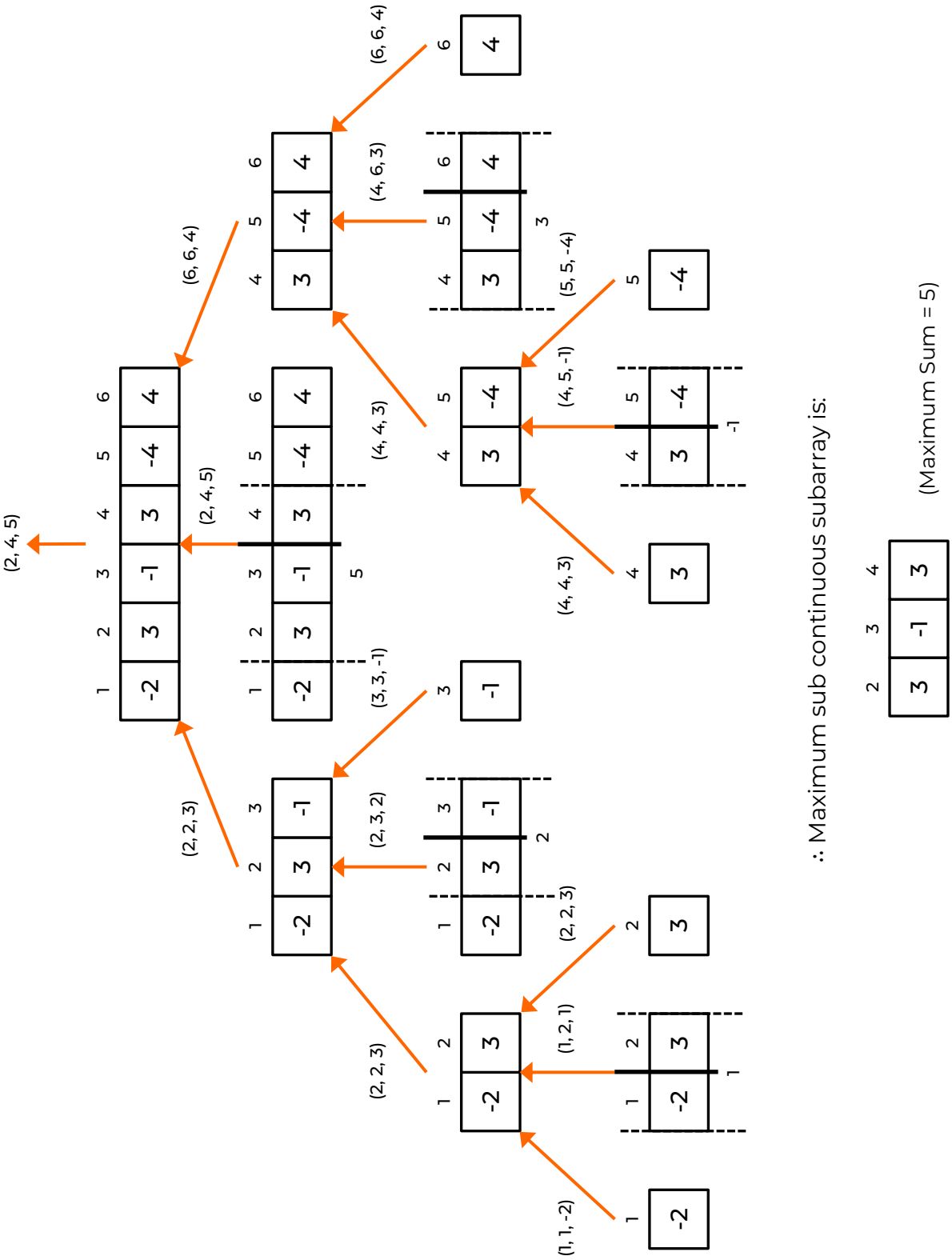
$= 40 \times 1 + 30 \times 1 + 35 \times 0.96875$

$= 103.90\ g$

$\therefore$ At most 103.90 grams of sugar can be consumed.

1. a) Given an array A = {-2, 3, -1, 3, -4, 4}, find the **maximum-sum continuous subarray** using divide-and-conquer approach. You must show the recursion tree and clearly mention left, right and crossing sum for each tree node.

Solution:



1. b) Find out a good asymptotic upper bound on the following recurrence:

$$T(n) = 3T\left(\frac{n}{4}\right) + O(n^2)$$

You may use **Recursion-tree** or **Master method** to solve the recurrence.

Solution:

Given,

$$T(n) = 3T\left(\frac{n}{4}\right) + O(n^2)$$

Comparing with base equation of Master method $T(n) = aT\left(\frac{n}{b}\right) + O(n^k \log^p n)$,

Here,

$a = 3$
$b = 4$
$k = 2$
$p = 0$
$b^k = 4^2 = 16$

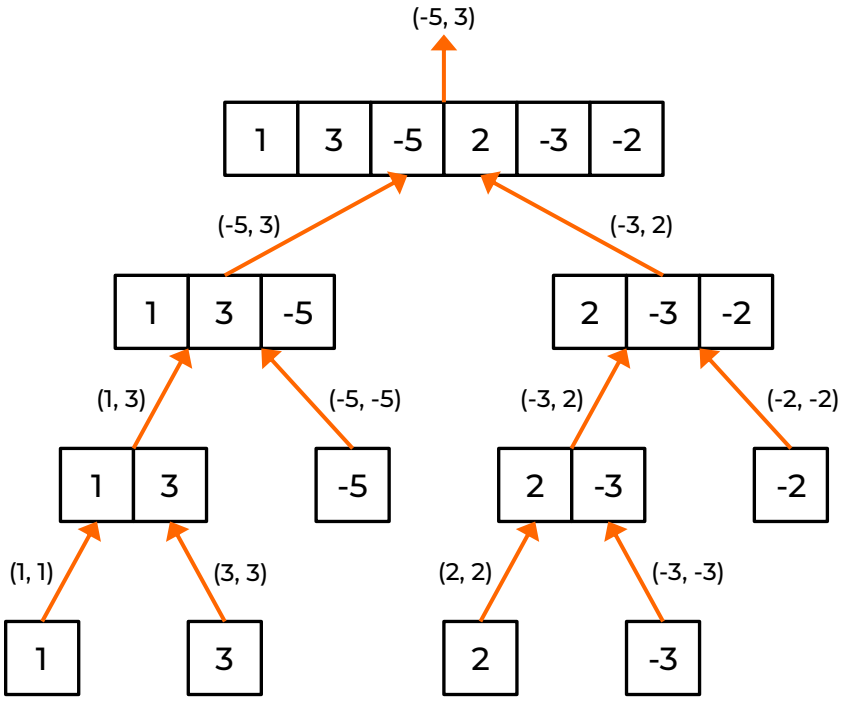
Since $a < b^k$ and $p \geq 0$, therefore this recurrence belongs to 1st condition of case 3.

∴ Time complexity, $T(n) = O(n^k \log^p n)$
$= O(n^2 \log^0 n)$
$= O(n^2)$

∴ Asymptotic upper bound of given recurrence is $O(n^2)$.

1. c) Given an array of integers A = {1, 3, -5, 2, -3, -2}, find the **Maximum** and **Minimum** using divide-and-conquer. Show the necessary steps to support your answer.

Solution:



∴ Maximum value is 3 and minimum value is -5.

2. a) After obtaining your BSCSE degree, you embarked on an entrepreneurial journey and established your own thriving software company. You've been consistently successful in securing projects from a variety of clients, ensuring a steady flow of profits. However, suddenly a situation arises where your decision-making and leadership skills are put to the test.

You have just received **5 project** offers from different clients, but you have only **7 days** to complete the projects. Your project manager prepares the following estimates for each of the projects and presents them to you for your decision.

Net Profit (In Million Dollars)	200	150	100	50	300
Duration (In Days)	3	2	1	2	5

Being an adept CSE graduate, you decide to approach the problem using **dynamic programming**. Determine which of the projects can be taken to **maximize the net profit**.

Note that you cannot partially complete a project. Also, you are unable to work on two projects at the same time on a particular day.

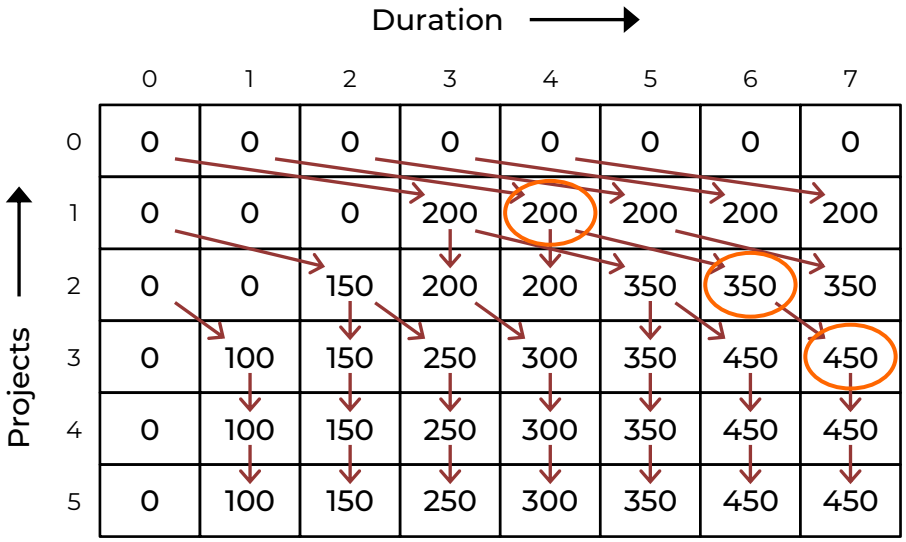
Solution:

Here,

Total times = 7 days

Project	1	2	3	4	5
Net Profit	200	150	100	50	300
Days	3	2	1	2	5

Now,



∴ Total maximize profit = 450 Million Dollars

Selected projects = 1, 2, 3

∴ 1st, 2nd and 3rd project can be selected for maximum profit.

2. b) Both the algorithmic paradigms: Divide-and-Conquer and Dynamic Programming solve a problem by breaking it into smaller problem instances, and by solving them. What is the **fundamental difference** between these two paradigms?

Solution:

Both the algorithmic paradigms Divide-and-Conquer and Dynamic Programming solve a problem by breaking it into smaller sub-problems but there are some fundamental difference. These differences are:

Divide and Conquer	Dynamic Programming
Break the problem into independent sub-problems.	Break the problem into overlapping sub-problems.
Solve each sub-problem recursively.	Solve sub-problems in a bottom-up approach, often store previously encountered sub-problems to use later.
Combines the solution of all sub-problems to get the final solution.	Avoid solving the same sub-problems multiple time by reusing stored solution.

2. c) You have found a treasure containing an infinite supply of $23, $16, $9, and $1 Coins. To go back home you need to pay a total of $25. However, despite having an infinite supply of coins you want to pay using the **minimum** number of coins. Give **an example** where the **greedy approach** does not provide an optimal solution in this matter.

Solution:

We have infinite supply of:
$23, $16, $9, $1

We need to pay $25.

In this scenario, if we use greedy method,
We will need to use these coins:
$23, $1, $1.

∴ Total 3 coin will needed in greedy method.

But if we use any other approach, like Dynamic Programming, then we can pay $25 in only 2 coins ($16, $9).

∴ Greedy approach does not provide an optimal solution in this matter.

3. a) Derive the **best-case** and the **worst-case** running time equations for the following function **specialTask** and represent using Asymptotic Notation.

```
1  void specialTask(int arr[], int n) {
2      for(int i=0; i<n; i++) {
3          for(int j=n-1; j>i; j--) {
4              if(arr[j]<arr[j-1]) {
5                  swap(arr[j], arr[j-1]);
6              }
7          }
```

```
 8           printArray(arr, n);
 9           bool flag = true;
10           for(int j=n-1; j>0; j--){
11               if(arr[j]<arr[j-1]) {
12                   flag = false;
13               }
14           }
15           if(flag == true) {
16               return;
17           }
18       }
19       return;
20   }
```

Solution:

Line	Worst Case	Best Case
2	$n+1$	1
3	$\frac{n(n+1)}{2}+n$	n
4	$\frac{n(n+1)}{2}$	$n-1$
5	$\frac{n(n+1)}{2}$	0
8	n	1
9	n	1
10	$n \times n$	n
11	$n(n-1)$	$n-1$
12	$n(n-1)$	0
15	n	1
16	0	1
19	1	1

$$\therefore \text{Best case, } T(n) = 1+n+n-1+1+1+n+n-1+1+1+1$$
$$= 4n+4$$
$$= O(n)$$

$$\therefore \text{Worst case, } T(n) = n+1+\frac{n(n+1)}{2}+n+\frac{n(n+1)}{2}+\frac{n(n+1)}{2}+n+n+n\times n$$
$$+n(n-1)+n(n-1)+n+1$$
$$= 4n+3\frac{n(n+1)}{2}+4+n^2+n^2-n+n^2-n+n$$
$$= 3n^2+\frac{3}{2}n(n+1)+4n+4$$
$$= O(n^2)$$

3. b) Derive the **exact-cost equation** for the running time of the following function **funCTION** and find the time complexity in Big-Oh notation.

```
1   int funCTION(int n, int m) {
2       int sumUP = 0;
3       for(int i=1; i<=n; i=i+2) {
4           int r = m;
5           while(r>=1) {
6               sumUP = sumUP+r;
7               r = r/3;
8           }
9       }
10      return sumUP;
11  }
```

Solution:

Line	Time Cost
2	1
3	$\frac{n}{2}+1$
4	$\frac{n}{2}$
5	$\frac{n}{2}\times\log_3 m+\frac{n}{2}$
6	$\frac{n}{2}\times\log_3 m$
7	$\frac{n}{2}\times\log_3 m$
10	1

$\therefore$ Time complexity, $T(n) = 1+\frac{n}{2}+1+\frac{n}{2}+\frac{n}{2}\times\log_3 m+\frac{n}{2}+\frac{n}{2}\times\log_3 m+\frac{n}{2}\times\log_3 m+1$

$$= \frac{3}{2}n+\frac{3}{2}n\log_3 m+3$$
$$= O(n\log m)$$

$\therefore$ Exact cost equation is $\frac{3}{2}n+\frac{3}{2}n\log_3 m+3$ and time complexity is $O(n\log m)$.

4. a) Find an optimal solution to the **fractional knapsack** instance of n = 5, W = 7, (v1, v2, v3, v4, v5) = (50, 30, 35, 60, 25), and (w1, w2, w3, w4) = (2, 2, 1, 3, 2). **Explain** why your solution satisfies the *optimal substructure property*.

Solution:

Given,

$n = 5$

$W = 7$

[P.T.O]

Item, i_i	i_1	i_2	i_3	i_4	i_5
Value, v_i	50	30	35	60	25
Weight, w_i	2	2	1	3	2

Finding price per unit:

	i_1	i_2	i_3	i_4	i_5
$\frac{v_i}{w_i}$	25	15	35	20	12.5

Sorting item based on $\frac{v_i}{w_i}$:

$i_3 > i_1 > i_4 > i_2 > i_5$

Adding item in knapsack:

Item	v_i	w_i	W	$x_i = \min\left(1, \frac{W}{w_i}\right)$	$W' = W - w_i x_i$
i_3	35	1	7	1	7-1 = 6
i_1	50	2	6	1	6-2 = 4
i_4	60	3	4	1	4-3 = 1
i_2	30	2	1	0.5	1-(2×0.5) = 1-1 = 0

$\therefore$ Total Profit $= v_3x_3 + v_1x_1 + v_4x_4 + v_2x_2$

$= 35 \times 1 + 50 \times 1 + 60 \times 1 + 30 \times 0.5$

$= 160$

This solution satisfies the optimal substructure property. Because if the knapsack capacity was 1, then $35 \times 1 = 35$ was the highest profit, if knapsack size was 3. than $35 \times 1 + 50 \times 1 = 85$ was the highest profit and like that, every sub-problem in here are the optimal solution for that stage.

4. b) What is **activity selection** problem? Is it true that the activity selection problem has one unique optimal solution? Justify your answer.

Solution:

Activity selection is a problem where various activities are given along with their starting and finishing time. We need to find the out maximum how many activities we can select in a specific time period.

It is false that Activity Selection problem has one unique solution. If there are three activities such that $\{[7,12), [8,12), [13,16)\}$ then some algorithm will give solution $\{[7, 12), [13,16)\}$ and other algorithm may give solution $\{[8,12), [13,16)\}$. But number of maximum selected activities will be same.

4. c) You are given the arrival and the departure times of eight trains for a railway platform, and each one is in the format: [arrival time, departure time). Only one

train can use the platform at a time. Suppose that you have got the following train-use requests for the next day.

$$\{ [8, 12), [6, 9), [11, 14), [2, 7), [1, 7), [12, 20), [7, 12) , [13, 19) \}$$

Find the **maximum number of trains** that can use the platform without any collision by using earliest departure time.

Solution:

Given,

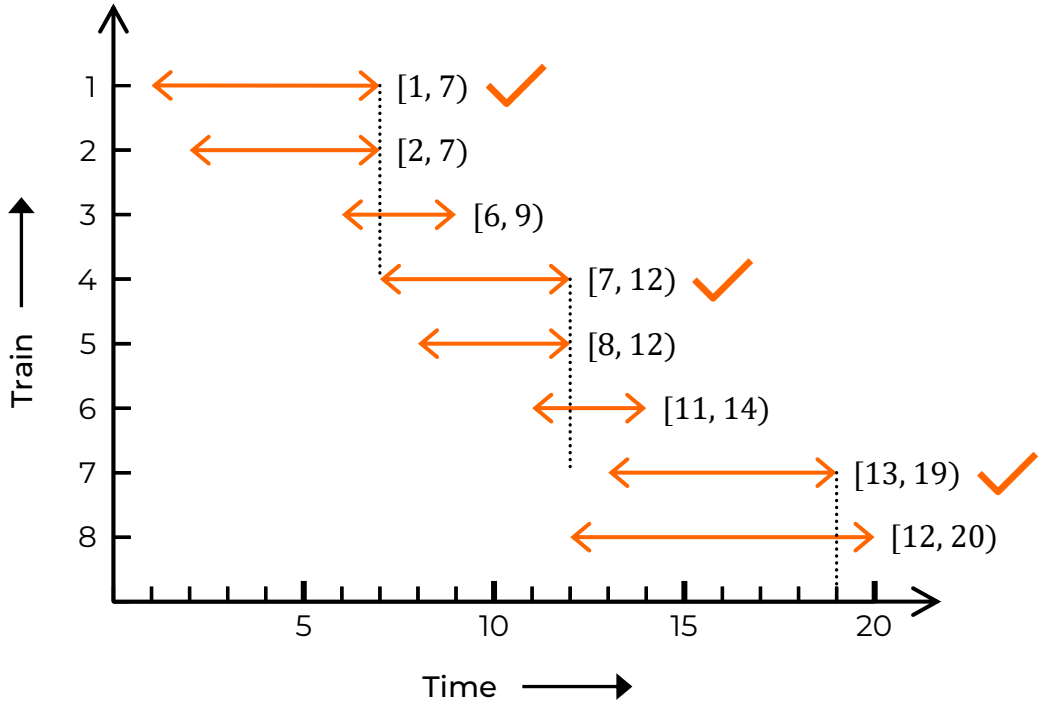
Following train requested for use platform:

$$\{ [8, 12), [6, 9), [11, 14), [2, 7), [1, 7), [12, 20), [7, 12) , [13, 19) \}$$

Sorting train based on departure time:

Train, t_i	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
Arrival Time	1	2	6	7	8	11	13	12
Departure Time	7	7	9	12	12	14	19	20

Now,



∴ Maximum 3 trains [Scheduled for $[1, 7)$, $[7, 12)$ and $[13, 19)$] can use the platform without any collision.

1. a) Derive the **best-case**, and the **worst-case** running time equations for the following function *connectDots* and represent using Asymptotic Notation.

```
1   bool connectDots(int arr[], int n){
2       int numberOfDots = 0;
3       for(int i=0; i<n; i++){
4           if(arr[i]&1){
5               for(int j=i+1; j<=n-1; j++){
6                   numberOfDots++; 7
7               }
8               for(int j=0;j<=i; j++){
9                   numberOfDots *= 2;
10              }
11          }
12          int j = 1; 13
13          while(j<=n){
14              numberOfDots+=j;
15              j = j * 2;
16          }
17      }
18      return (numberOfDots&1);
19  }
```

Solution:

Line	Worst Case	Best Case
2	1	1
3	$n+1$	$n+1$
4	n	n
5	$\frac{n(n+1)}{2}+n$	0
6	$\frac{n(n+1)}{2}$	0
8	$\frac{n(n+1)}{2}+n$	0
9	$\frac{n(n+1)}{2}$	0
12	n	n
13	$n\log_2 n+n$	$n\log_2 n+n$
14	$n\log_2 n$	$n\log_2 n$
15	$n\log_2 n$	$n\log_2 n$
18	1	1

$$\therefore \text{Best case, } T(n) = 1 + n + 1 + n + n + n\log_2 n + n + n\log_2 n + n\log_2 n + 1$$
$$= 3n\log_2 n + 4n + 3$$
$$= O(n\log n)$$

$$\therefore \text{Worst case, } T(n) = 1 + n + 1 + \frac{n(n+1)}{2} + n + \frac{n(n+1)}{2} + \frac{n(n+1)}{2} + n + \frac{n(n+1)}{2} + n$$
$$+ n\log_2 n + n + n\log_2 n + n\log_2 n + 1$$
$$= 4\frac{n^2+n}{2} + 3n\log_2 n + 5n + 3$$
$$= 2n^2 + 7n + 3n\log_2 n + 3$$
$$= O(n^2)$$

1. b) Derive the **exact-cost equation** for the running time of the following function and find the time complexity in big-oh notation.

```
1   for (int i = 1; i <= n; i = i * 2){
2       for (int j = 1; j <= i; j++){
3           for (int k = n; k >= i; k--){
4               printf("%d ", k);
5           }
6           printf("\n");
7       }
8       printf("\n");
9   }
```

Solution:

Line	Time Cost
1	$\log_2 n + 1$
2	$\frac{\log_2 n\,(\log_2 n + 1)}{2} + \log_2 n$
3	$\frac{\log_2 n\,(\log_2 n + 1)}{2} \times \frac{\log_2 n\,(\log_2 n + 1)}{2} + \frac{\log_2 n\,(\log_2 n + 1)}{2}$
4	$\frac{\log_2 n\,(\log_2 n + 1)}{2} \times \frac{\log_2 n\,(\log_2 n + 1)}{2}$
6	$\frac{\log_2 n\,(\log_2 n + 1)}{2}$
8	$\log_2 n$

$$\therefore \text{Time complexity, } T(n) = \log_2 n + 1 + \frac{\log_2 n\,(\log_2 n + 1)}{2} + \log_2 n + \frac{\log_2 n\,(\log_2 n + 1)}{2}$$
$$\times \frac{\log_2 n\,(\log_2 n + 1)}{2} + \frac{\log_2 n\,(\log_2 n + 1)}{2} + \frac{\log_2 n\,(\log_2 n + 1)}{2}$$
$$\times \frac{\log_2 n\,(\log_2 n + 1)}{2} + \frac{\log_2 n\,(\log_2 n + 1)}{2} + \log_2 n$$
$$= 2\left[\frac{\log_2 n\,(\log_2 n + 1) \times \log_2 n\,(\log_2 n + 1)}{4}\right] + 3\frac{\log_n(\log_2 n + 1)}{2}$$
$$+ 3\log_2 n + 1$$

$$= \frac{1}{2}(\log_2{}^4 n + 2\log_2{}^3 n + \log_2{}^2 n) + \frac{3}{2}(\log_2{}^2 n + \log_2 n) + 3\log_2 n + 1$$
$$= \frac{1}{2}\log_2{}^4 n + \log_2{}^3 n + \frac{1}{2}\log_2{}^2 n + \frac{3}{2}\log_2{}^2 n + \frac{3}{2}\log_2 n + 3\log_2 n + 1$$
$$= \frac{1}{2}\log_2{}^4 n + \log_2{}^3 n + 2\log_2{}^2 n + \frac{5}{2}\log_2 n + 1$$
$$= O(\log^4 n)$$

∴ Exact cost equation is $\frac{1}{2}\log_2{}^4 n + \log_2{}^3 n + 2\log_2{}^2 n + \frac{5}{2}\log_2 n + 1$ and time complexity is $O(\log^4 n)$.

2. a) Solve the following **recurrence equation**, where $T(1) = O(1)$.

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

Solution:

Given,

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

Comparing with base equation of Master method $T(n) = aT\left(\frac{n}{b}\right) + O(n^k \log^p n)$,
Here,

$a = 4$
$b = 2$
$k = 1$
$p = 0$
$b^k = 2^1 = 2$

Since $a > b^k$, therefore this recurrence belongs case 1 of Master Theorem.

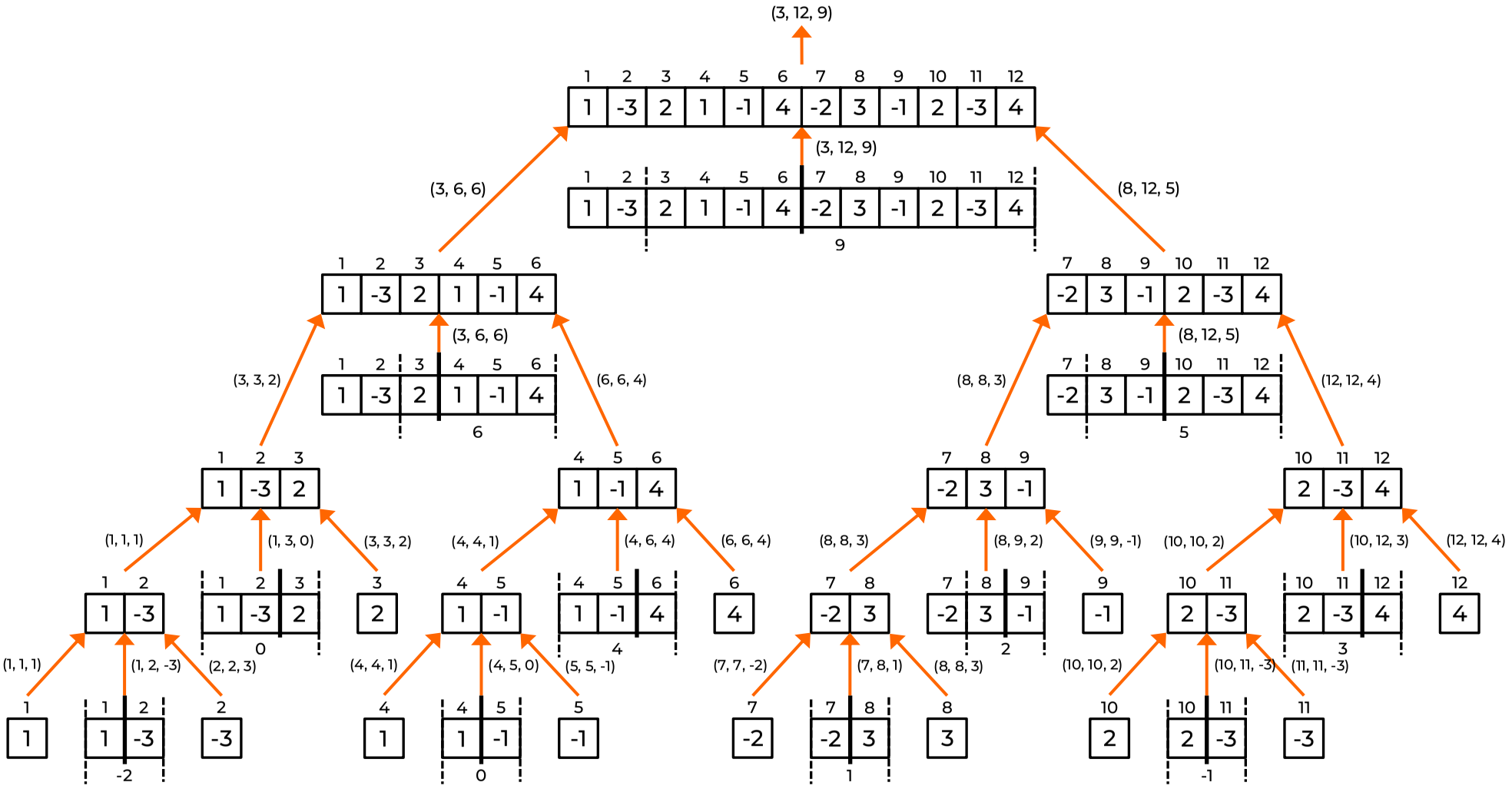
∴ Time complexity, $T(n) = O(n^{\log_b a})$
$= O(n^{\log_2 4})$
$= O(n^2)$

∴ The solution of recurrence equation is $O(n^2)$.

2. b) You are given an array of integers A = {1, -3, 2, 1, -1, 4, -2, 3, -1, 2, -3, 4}, find the **maximum sum subarray** using divide-and-conquer approach. You must show the recursion tree and clearly mention left, right and crossing sum for each tree node.

Solution:

[P.T.O]



∴ Maximum continuous subarray is:

3	4	5	6	7	8	9	10	11	12
2	1	-1	4	-2	3	-1	2	-3	4

(Maximum Sum = 9)

2. c) Suppose we have two sorted sub-arrays: L: 1, 5, 7, 8, 10, 12 and R: 4, 6, 7, 9, 13, 14. Perform the **procedure Merge** on L and R to find the final sorted array A. Show each step of your answer and the number of comparisons required in each step.

Solution:

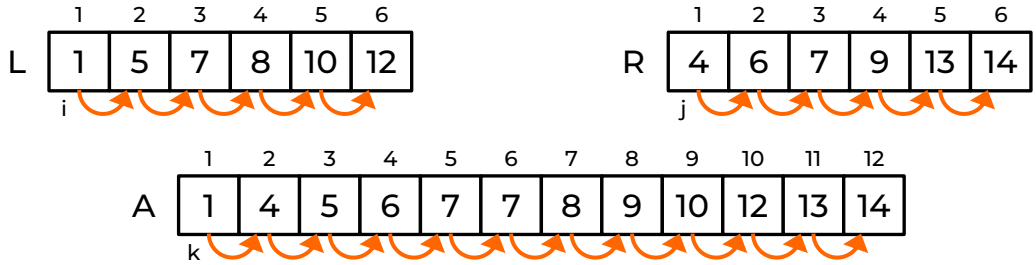
Let,

$i = 1$

$j = 1$

$k = 1$

Now,



1. $i = 1, j = 1, k = 1$
   $L[1] \leq R[1]; \therefore A[1] = L[1] = 1; i{+}{+}; k{+}{+};$
2. $i = 2, j = 1, k = 2$
   $L[2] \geq R[1]; \therefore A[2] = R[1] = 4; j{+}{+}; k{+}{+};$
3. $i = 2, j = 2, k = 3$
   $L[2] \leq R[2]; \therefore A[3] = L[2] = 5; i{+}{+}; k{+}{+};$
4. $i = 3, j = 2, k = 4$
   $L[3] \geq R[2]; \therefore A[4] = R[2] = 6; j{+}{+}; k{+}{+};$
5. $i = 3, j = 3, k = 5$
   $L[3] \geq R[3]; \therefore A[5] = R[3] = 7; j{+}{+}; k{+}{+};$
6. $i = 3, j = 4, k = 6$
   $L[3] \leq R[4]; \therefore A[6] = L[3] = 7; i{+}{+}; k{+}{+};$
7. $i = 4, j = 4, k = 7$
   $L[4] \leq R[4]; \therefore A[7] = L[4] = 8; i{+}{+}; k{+}{+};$
8. $i = 5, j = 4, k = 8$
   $L[5] \geq R[4]; \therefore A[8] = R[4] = 9; j{+}{+}; k{+}{+};$
9. $i = 5, j = 5, k = 9$
   $L[5] \leq R[5]; \therefore A[9] = L[5] = 10; i{+}{+}; k{+}{+};$
10. $i = 6, j = 5, k = 10$
    $L[6] \leq R[5]; \therefore A[10] = L[6] = 12; i{+}{+}; k{+}{+};$
    $i = 6, j = 5, k = 11$
    $A[11] = R[5] = 13; j{+}{+}; k{+}{+};$
    $i = 6, j = 6, k = 12$
    $A[12] = R[6] = 14; j{+}{+}; k{+}{+};$

$\therefore$ We need 10 element comparisons. After that, there will be only one array (R) left and that will not need comparisons for further steps.

3. a) What is **optimal substructure** property? Write down the optimal substructure property of the **coin change** problem.

Solution:

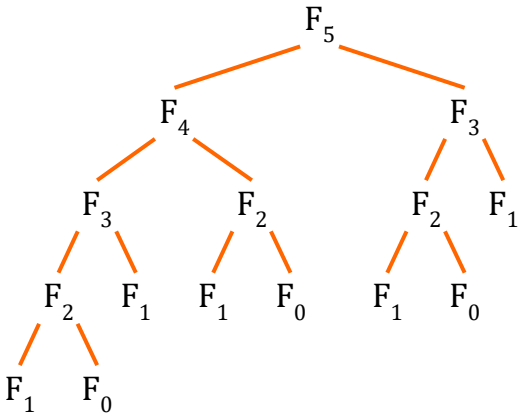
The optimal substructure property is a key concept of Dynamic Programming. It essentially states that an optimal solution of a problem can be constructed from optimal solutions to its sub-problems.

In this concept, every sub-problem should be a ultimate optimal solution at that sub-problem's current stage. For example, we have a coin change problem with total coins $C = \{1,2,3,5\}$ and we will need to make \$10. Now in a sub-problems of that problem where we have only coins $C = \{1,2\}$, then here we will got the ultimate optimal solution for making \$10 with only coins $C = \{1,2\}$.

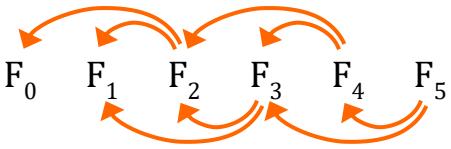
2. b) Demonstrate why the recursive approach to calculate a Fibonacci number is inefficient, by calculating the Fibonacci number F_5. How does the dynamic programming approach for the same solve this inefficiency? (Consider $F_0=0$, $F_1=1$)

Solution:

Recursive approach for finding F_5:



Dynamic programming approach for finding F_5:



In recursive approach, we are solving same repeated sub-problems for multiple time such as F_3, F_2 for calculating Fibonacci number F_5. One the other hand, in dynamic programming approach, we store the sub-problem after solving it and reuse it later instead of solving same overlapping sub-problem multiple times. In this way, dynamic approach solve the inefficiency to calculate a Fibonacci number.

3. c) A smuggler enters a warehouse to find the items listed in the following table. He has a bag to carry the smuggled goods, but it can carry only 8 kg weight at best. The smuggler wants to leave with the items that will result in a maximum profit for him. Note that he cannot take an item partially; he either will take the item, or will not.

Using dynamic programming, **calculate the maximum** profit the smuggler can earn.

Item no.	1	2	3	4
Weight	3	5	4	6
Profit	10	30	25	50

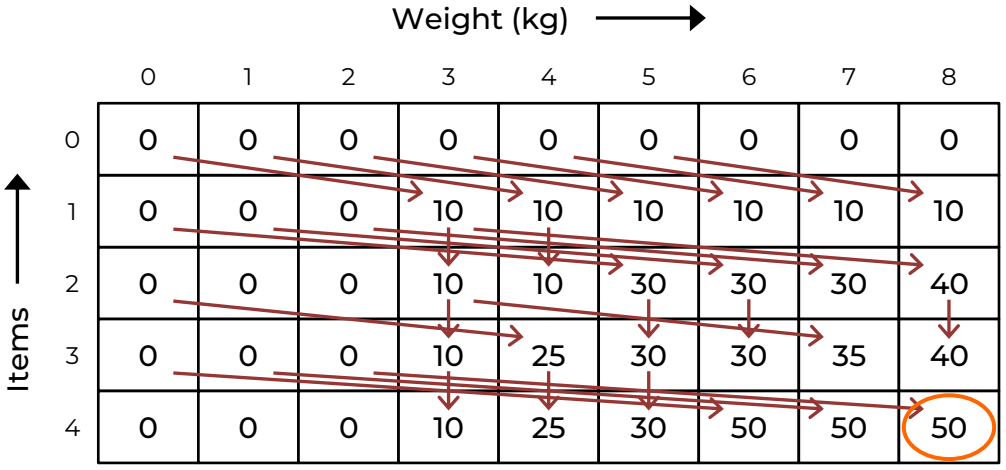
Solution:

Here,

Weight capacity = 8 kg

Item	1	2	3	4
Weight	3	5	4	6
Profit	10	30	25	50

Now,



∴ Total maximize profit = 50

Selected item: 4th item

4. a) Following items are available in a grocery shop:

- 12 kilogram rice grain which costs 840 taka
- 10 kilogram salt which costs 870 taka
- 8 kilogram saffron powder which costs 2000 taka and
- 5 kilogram sugar which costs 500 taka

A group of thieves (Thief 1, Thief 2, ... Thief M) have come to steal from that shop, **each with a knapsack of capacity 9 kg**. The thieves are entering in serial, Thief 2 enters after *Thief 1* is done with stealing, *Thief 3* enters after *Thief 2* is done with stealing and so on. *Since each thief wants to maximize his/her profit*, **how many thieves** will be needed in the group to empty the grocery shop and **what are the items** that each of those thieves carry? Show details of the calculation.

Solution:

Given,

Capacity of each thief, $W = 9\ kg$

Item	Rice Grain	Salt	Saffron Powder	Sugar
Value (taka), v	840	870	2000	500
Weight (kg), w	12	10	8	5

Finding price per unit:

[P.T.O]

$\frac{v_i}{w_i}$	70	87	250	100
	Rice Grain	Salt	Saffron Powder	Sugar

Sorting item based on $\frac{v_i}{w_i}$:

Saffron Powder > Sugar > Salt > Rice Grain

Thief 1:

Item	v_i	w_i	W	$x_i = \min\left(1, \frac{W}{w_i}\right)$	$W' = W - w_i x_i$
Saffron Powder	2000	8	9	1	9-8 = 1
Sugar	500	5	1	0.2	1-(5×0.2) = 0

Now, Sugar new weight, $w_i' = w_i - w_i x_i = 5 - 5 \times 0.2 = 4$
Sugar new value, $v_i' = v_i - v_i x_i = 500 - 500 \times 0.2 = 400$

Thief 2:

Item	v_i	w_i	W	$x_i = \min\left(1, \frac{W}{w_i}\right)$	$W' = W - w_i x_i$
Sugar	400	4	9	1	9-4 = 5
Salt	870	10	5	0.5	5-(10×0.5) = 0

Now, Salt new weight, $w_i' = w_i - w_i x_i = 10 - 10 \times 0.5 = 5$
Salt new value, $v_i' = v_i - v_i x_i = 870 - 870 \times 0.5 = 435$

Thief 3:

Item	v_i	w_i	W	$x_i = \min\left(1, \frac{W}{w_i}\right)$	$W' = W - w_i x_i$
Sugar	435	5	9	1	9-5 = 4
Rice Grain	840	12	4	0.33	4-(12×0.33) = 0

Now, Rice Grain new weight, $w_i' = w_i - w_i x_i = 12 - 12 \times 0.33 = 8$
Rice Grain new value, $v_i' = v_i - v_i x_i = 840 - 840 \times 0.33 = 560$

Thief 4:

Item	v_i	w_i	W	$x_i = \min\left(1, \frac{W}{w_i}\right)$	$W' = W - w_i x_i$
Rice Grain	560	8	9	1	9-8 = 1

Here, all items in the grocery shop have been stolen.
∴ Total 4 thieves will be needed in the group to empty the grocery shop.

4. b) A document to be transmitted over the internet contains the following characters with their associated frequencies as shown in the following table:

Character	a	e	l	n	o	s	t
Frequency	74	105	44	55	73	57	49

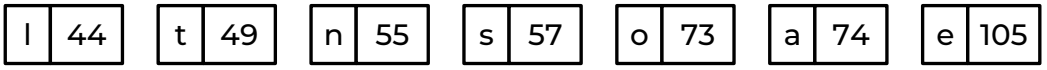
Use Huffman technique to answer the following questions:

i. Build the Huffman code tree for the message and find the codeword for each character. **Encode** "*stolen*" using the codewords.

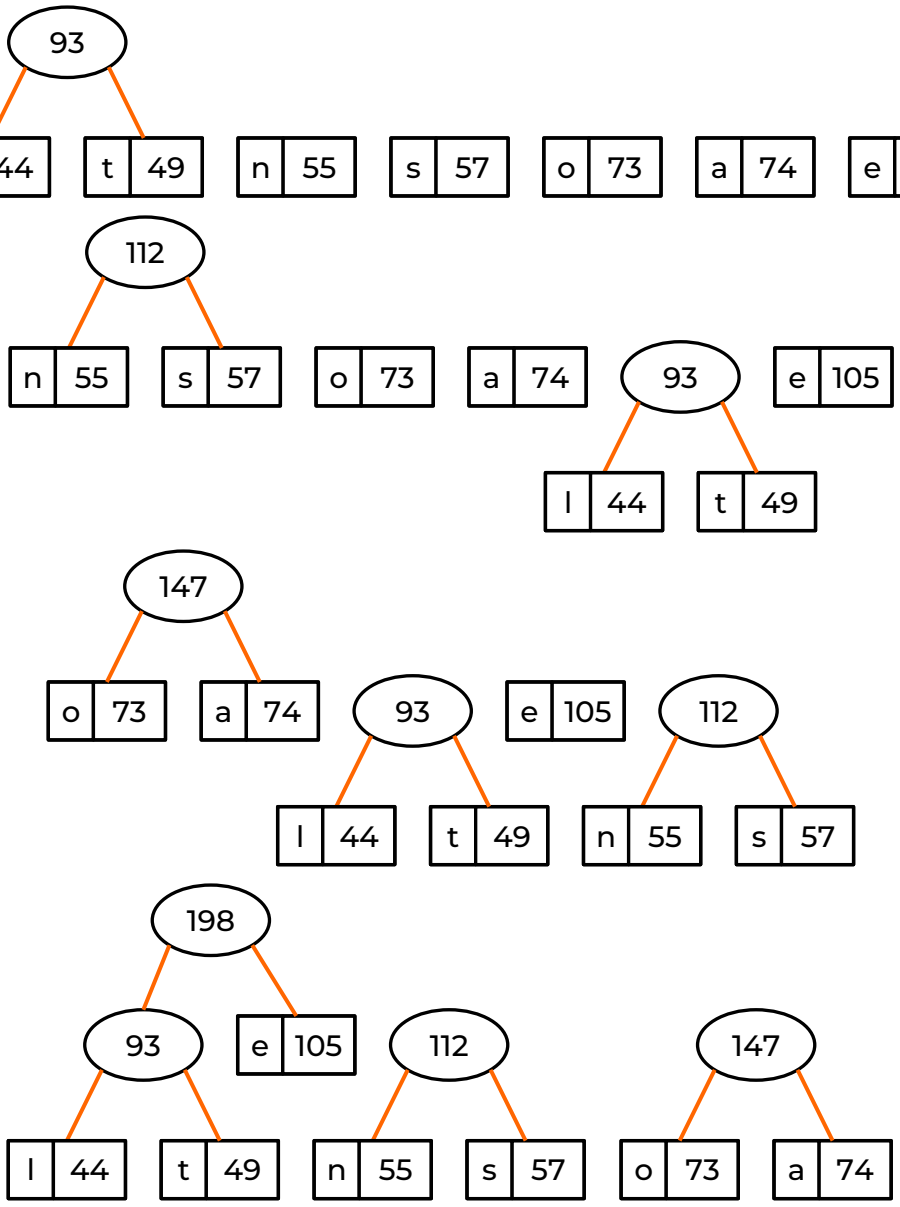
ii. What is the **percentage saving** if the data is sent with fixed-length code values without compression?

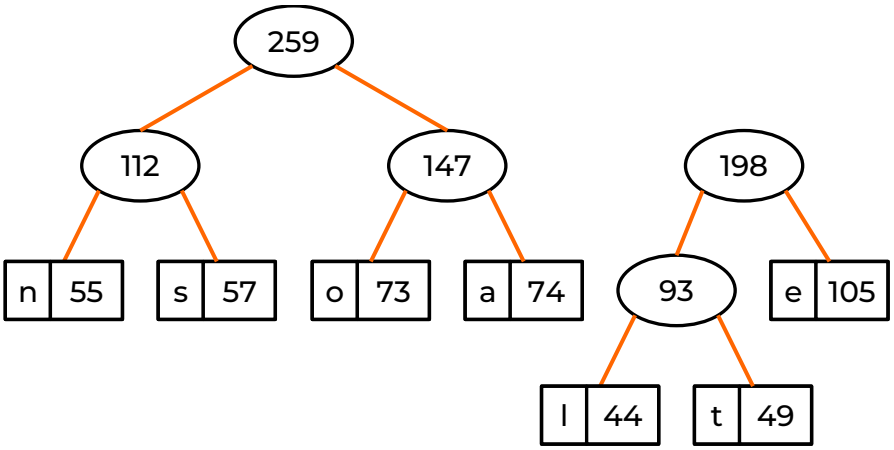
Solution:

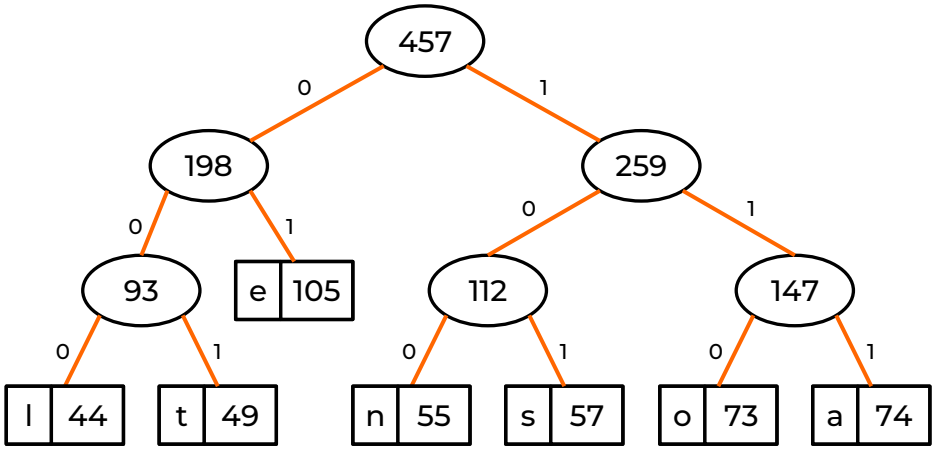
i. Sorting character based on frequency:



Building Huffman code tree:







This is our final Huffman coding tree.

Finding codeword for each character from Huffman tree:

e: 01	a: 111
o: 110	s: 101
n: 100	t: 001
l: 000	

∴ Encode of “stolen” = 10100111000001100

ii.

Spring 2023

1. a) Consider a modified version of the Merge sort algorithm as follows:

If the array size is less than or equal to 2, then it sorts the array at **constant time**.

Otherwise, it divides the array of size n into 3 subarrays, each with a size of n/3. This division takes $\boldsymbol{O(n^2)}$ time. Then the algorithm sorts the subarrays recursively, and then merges their solutions in time $\boldsymbol{O(n \log n)}$. **Write a recurrence relation** for the running time $\boldsymbol{T(n)}$ of this algorithm.

Solution:

Recurrence relation for the running time $T(n)$ of this algorithm is:

$$T(n) = \begin{cases} O(1) & \text{if } n \leq 2 \\ 3T\left(\frac{n}{3}\right) + O(n^2) + O(n \log n) & \text{if } n > 2 \end{cases}$$

In the recursive relation, the time complexity includes:

- $3T\left(\frac{n}{3}\right)$ for sorting 3 subarrays recursively
- $O(n^2)$ for the time taken for the division into 3 subarrays
- $O(n \log n)$ for merging the sorted subarrays

1. b) Prove that the divide and conquer method will sort an array in $\boldsymbol{O(n \log n)}$ time when n>1.

Solution:

In divide and conquer method, for sorting an array, we divide the array into two halves and solve them recursively.

So we have,

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

Finally we merge these subarray, which take $O(n)$ times.

∴ Here our recursive relation is,

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

Comparing with base equation of Master method $T(n) = aT\left(\frac{n}{b}\right) + O(n^k \log^p n)$,

Here,

$a = 2$
$b = 2$
$k = 1$
$p = 0$
$b^k = 2^1 = 2$

Since $a = b^k$ and $p >$ -1, therefore this recurrence belongs to 1st condition of case 2 of Master Theorem.

∴ Time complexity, $T(n) = O\left(n^{\log_b a} \log^{p+1} n\right)$

$= O\left(n^{\log_2 2} \log^{0+1} n\right)$

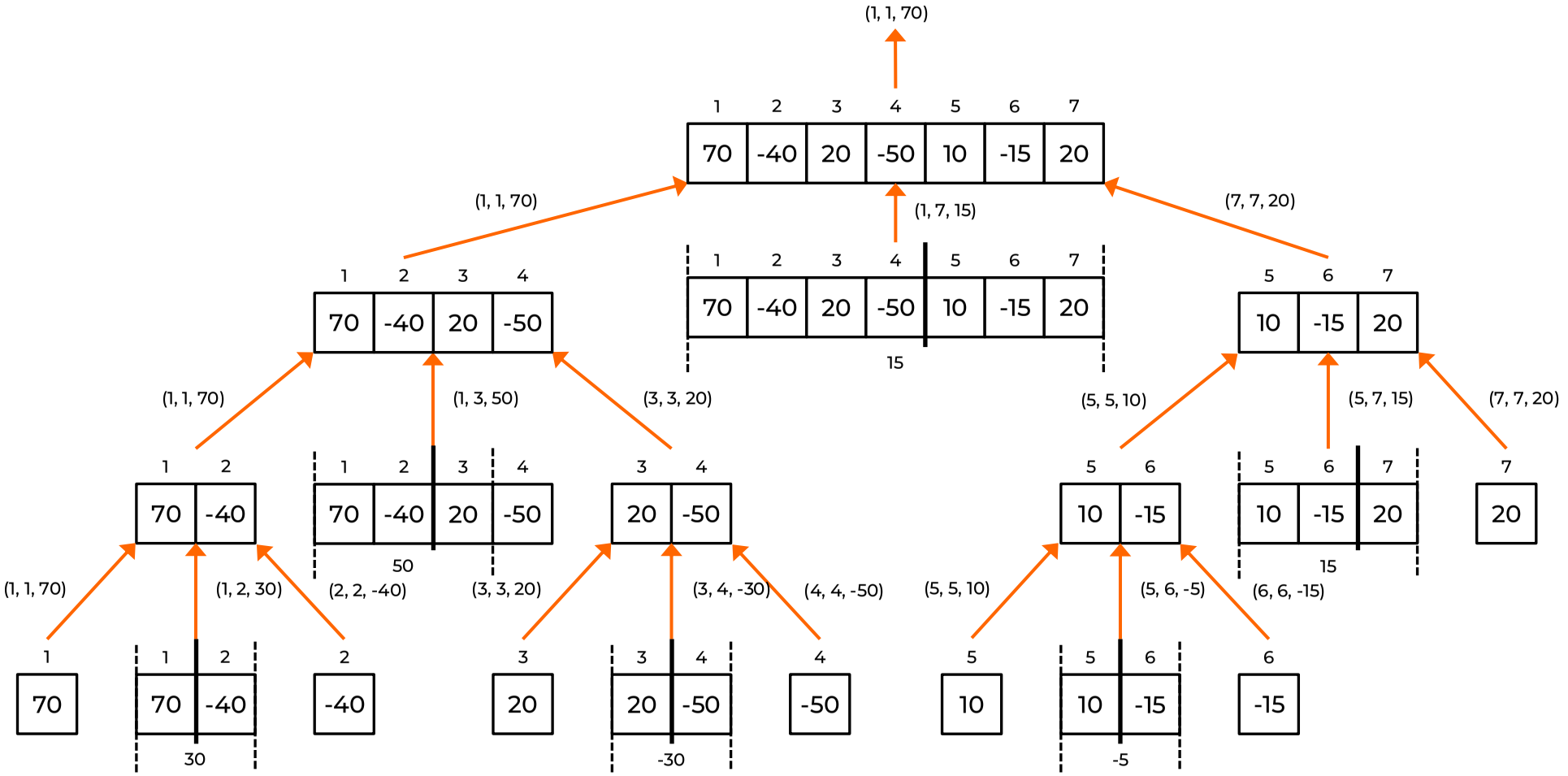
$= O(n \log n)$

∴ Divide and conquer method will sort an array in $O(n \log n)$. (Proved)

1. c) Given an array of integers **A =** `{70, -40, 20, -50, 10, -15, 20}`, find the **Maximum-sum Continuous Subarray** using divide-and-conquer. You must show the recursion tree and clearly mention **left, right and crossing sum** for each tree node.

Solution:

[P.T.O]



∴ Maximum sub continuous subarray is:

1

70

2. a) What is the main difference between **Dynamic Programming** and **Divide-and-Conquer** algorithms? When should we try to solve a problem using **Dynamic Programming**?

Solution:

Divide-and-Conquer algorithm divide a problem into independent sub-problems and solve each sub-problem and combines the solutions recursively. On the other hand, Dynamic Programming algorithm break the problem into overlapping sub-problems and avoid solving same repeated sub-problems multiple time by store and reusing the solution of sub-problems. This is the main difference between Dynamic Programming and Divide-and-Conquer.

We should try to solve a problem using Dynamic Programming when we have more chance to having repeated overlapping sub-problems to remove inefficiency. For example, calculating Fibonacci number.

2. b) Suppose you are a motorcycle dealer buying cars wholesale and then selling them at retail price for a profit. You have a budget of **7 lac Tk**. You went to the wholesale market, and you saw the following items on sale:

Car Name	**Yamaha R15**	**Harley Davidson**	**Apache RTR 160**	**Honda CBR 150R R**
Wholesale price (in lac Taka)	3	6	1	4
Retail Price (in lac Taka)	5	14	2	6

You want to get the **maximum profit** from selling all these motorcycles, but your budget restricts you from buying all of them. Find the maximum profit you can obtain by buying some of these motorcycles and then selling them, provided the total wholesale cost of the motorcycles you selected **do not exceed your budget of 7 lac Tk**. *Note that you can only buy one of these motorcycles at a time, so if you purchased a Yamaha R15 from the wholesale market, you cannot purchase it again.* Find the solution to this problem by using **dynamic programming** and creating a **lookup table**.

Solution:

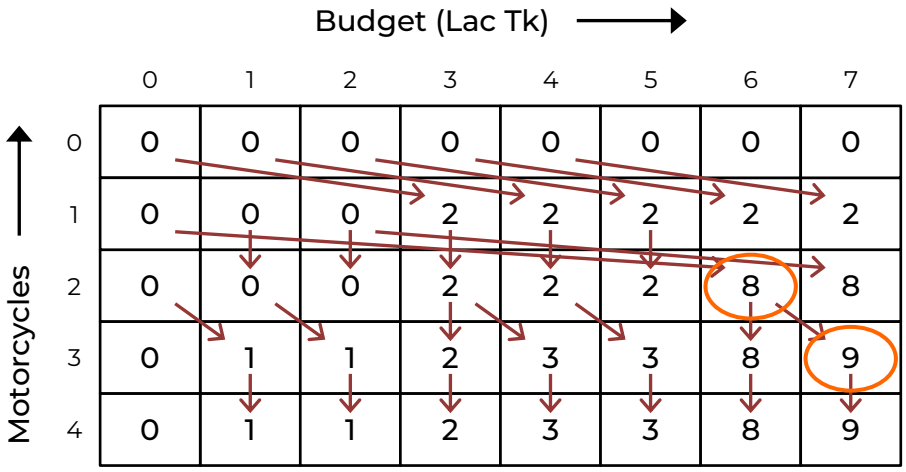
Here,

Total budget = 7 Lac Tk

Index	1	2	3	4
Motorcycle Name	Yamaha R15	Harley Davidson	Apache RTR 160	Honda CBR 150R R
Wholesale Price, w_i	3	6	1	4
Retail Price	5	14	2	6
Profit, v_i	2	8	1	2

Now,

[P.T.O]



∴ Total maximize profit = 9 Lac Tk
Selected motorcycles : 2 (Harley Davidson), 3 (Apache RTR 160)

2. c) Suppose now you have an **infinite supply** of motorcycles in the wholesale market, which means that if you purchased a Yamaha R15 from the wholesale market, you could purchase it again (*if you have the required amount of money*). What **modification should you make** to the algorithm you used in **question (b)** to make it work for an infinite supply of motorcycles? (You do not need to show any lookup tables here).

Solution:

We use 0/1 knapsack algorithm in question (b). In the 0/1 knapsack, the algorithm is based on choosing an item or not. For the infinity supply of item, we must consider the possibility of taking the same item multiple times. Normally we use 2D array in 0/1 knapsack, but for this case we should reduce the 2D array to a 1D array since we are no longer concerned with distinct items being selected only once. Let, our capacity is j and our array is $dp[j]$. For each capacity $dp[0], dp[1] \dots dp[j]$, we have to check all items to improves the value. This loop effectively considers each item multiple times. Then we will get our final answer in last index of array, $dp[j]$.

3. a) Calculate the time complexity (Best Case and Worst Case) of the following code snippets.

(i)
```
for (int i =1; i<n; i=i*2) {
    p++;
}
for(int j=1; j<p; j=j*2){
    printf("hello");
}
```

(ii)
```
for(int i =1; i*i<n; i++){
    printf("hello");
}
```

(iii)
```
for(int i =1; i<n; i=i*2){
    for(int j=1; j<i; j++){
        printf("hello");
```

```
        }
}
```

Solution:

Since there are no use of condition, return, break, continue; therefore best the best case and worst case time complexity will be same for all of these code snippets.

i)

Line	Time Cost
1	$\log_2 n + 1$
2	$\log_2 n$
4	$\log_2 n\,(\log_2 n) + 1$
5	$\log_2 n\,(\log_2 n)$

$$\therefore \text{Time complexity, } T(n) = \log_2 n + 1 + \log_2 n + \log_2 n\,(\log_2 n) + 1 + \log_2 n\,(\log_2 n)$$
$$= 2\log_2 n + 2\log_2 n\,(\log_2 n) + 2$$
$$= O(\log n)$$

ii)

Line	Time Cost
1	$\sqrt{n} + 1$
2	$\sqrt{n}$

$$\therefore \text{Time complexity, } T(n) = \sqrt{n} + 1 + \sqrt{n}$$
$$= O(\sqrt{n})$$

iii)

Line	Time Cost
1	$\log_2 n + 1$
2	$\frac{\log_2 n\,(\log_2 n + 1)}{2}$
3	$\frac{\log_2 n\,(\log_2 n + 1)}{2} + 1$

$$\therefore \text{Time complexity, } T(n) = \log_2 n + 1 + \frac{\log_2 n\,(\log_2 n + 1)}{2} + 1 + \frac{\log_2 n\,(\log_2 n + 1)}{2}$$
$$= \log_2 n + 2\left(\frac{\log_2{}^2 n + \log_2 n}{2}\right) + 2$$
$$= \log_2{}^2 n + 2\log_2 n + 2$$
$$= O(\log^2 n)$$

3. b) Given $f(n) = 5n^3 + 6n^2 + 3n + 9$; $g(n) = n^4$; find the values of $\boldsymbol{c}$ and $\boldsymbol{n_0}$ such that when $n > n_0$, $\boldsymbol{f(n) = O(g(n))}$

Solution:

Given,

$f(n) = 5n^3 + 6n^2 + 3n + 9$

$g(n) = n^4$

Proving $f(n) \leq c \cdot g(n)$ for any value of c and n_0 such that $n > n_0$ is enough to prove $f(n) = O(g(n))$.

Now,

$f(n) \leq c \cdot g(n)$

or, $5n^3 + 6n^2 + 3n + 9 \leq c \cdot n^4$

or, $\frac{5}{n} + \frac{6}{n^2} + \frac{3}{n^3} + \frac{9}{n^4} \leq c$

∴ Big-Oh notation $f(n) = O(g(n))$ holds for any value for n_0 such that $1 > n_0$ and $c = 23$. $\left[\frac{5}{1} + \frac{6}{1^2} + \frac{3}{1^3} + \frac{9}{1^4} = 23\right]$

4. a) Find an optimal solution to the **fractional knapsack** instance of n = 4, W = 5, (v1, v2, v3, v4) = (50, 30, 35, 60), and (w1, w2, w3, w4) = (2, 2, 1, 3).

Solution:

Given,

$n = 5$

$W = 7$

Item, i_i	i_1	i_2	i_3	i_4
Value, v_i	50	30	35	60
Weight, w_i	2	2	1	3

Finding price per unit:

	i_1	i_2	i_3	i_4
$\frac{v_i}{w_i}$	25	15	35	20

Sorting item based on $\frac{v_i}{w_i}$:

$i_3 > i_1 > i_4 > i_2$

Adding item in knapsack:

Item	v_i	w_i	W	$x_i = \min\left(1, \frac{W}{w_i}\right)$	$W' = W - w_i x_i$
i_3	35	1	5	1	5-1 = 1
i_1	50	2	4	1	4-2 = 2
i_4	60	3	2	0.667	2-(3×0.667) = 2-2 = 0

$\therefore$ Total Profit $= v_3 i_3 + v_1 i_1 + v_4 i_4$

$= 35 \times 1 + 50 \times 1 + 60 \times 0.667$

$= 125$

4. b) Suppose we want to encode the symbols {A, B, C, D} in binary. **Is the following a valid Huffman code?**

$$\{A: 0;\ B: 10;\ C: 110;\ D: 111\}$$

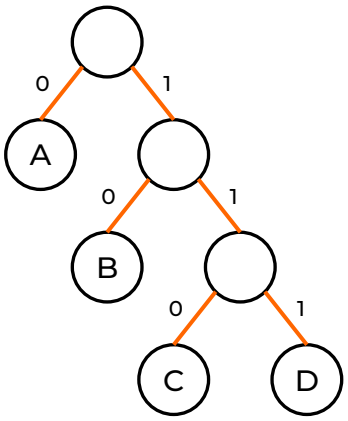
If it is, build the code tree; if not, explain why you can't.

Solution:

Given,

$\{A: 0;\ B: 10;\ C: 110;\ D: 111\}$

If we can make Huffman tree for these given Huffman code without any collision, then the given Huffman code is valid.



Analyzing the Huffman code tree, we can see that every character is accessable and there are not any overlapping issue.

$\therefore$ This is a valid Huffman code.

4. c) You are given the arrival and the departure times of eight trains for a railway platform, and each one is in the format: [arrival time, departure time). Only one train can use the platform at a time. Suppose that you have got the following train-use requests for the next day.

$$\{\ [8, 12), [6, 9), [11, 14), [2, 7), [1, 7), [12, 20), [7, 12)\ , [13, 19)\ \}$$

Find the **maximum number of trains** that can use the platform without any collision by using earliest departure time.

Solution:

*Repeat of **Fall 2023 Question 4(c)***

Fall 2022

1. a) Suppose, A problem X of size n can be divided into three subproblems each of size n/4, each of the problem can be solved recursively in time T(n/4) respectively. The cost of dividing the problem and combining the results of the subproblems is $O(nlogn)$. **Formulate** the recurrence relation assuming, $T(1) = O(1)$.

Solution:

Recurrence relation for the given problem is:

$$T(n) = 3T\left(\frac{n}{4}\right) + O(n \log n)$$

In the recursive relation, the time complexity includes:

- $4T\left(\frac{n}{4}\right)$ for solving 4 sub-problems size of $n/4$ recursively
- $O(n \log n)$ for merging the solved sub-problems

1. b) **Solve** the following recurrence equation: $T(n) = 3T(n/3) + O(1)$, where $T(1) = O(1)$.

Solution:

Given,

$$T(n) = 3T\left(\frac{n}{3}\right) + O(1)$$

Comparing with base equation of Master method $T(n) = aT\left(\frac{n}{b}\right) + O(n^k \log^p n)$,

Here,

$a = 3$
$b = 3$
$k = 0$
$p = 0$
$b^k = 3^0 = 1$

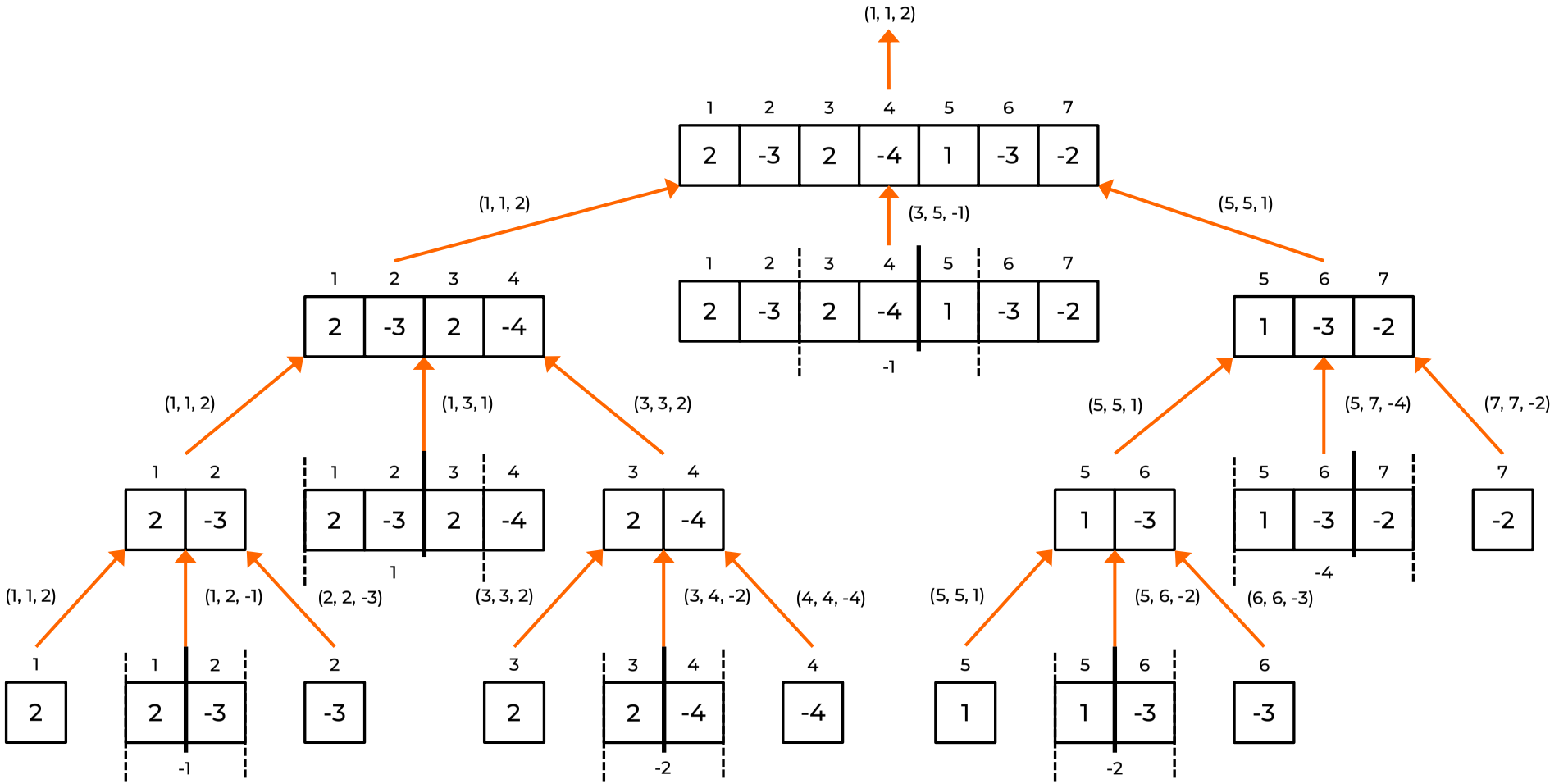
Since $a > b^k$, therefore this recurrence belongs to case 1 of Master Theorem.

∴ Time complexity, $T(n) = O(n^{\log_b a})$
$= O(n^{\log_3 3})$
$= O(n)$

∴ Solution of following recurrence equation is $O(n)$.

1. c) Given an array of integers **A =** {2, −3, 2, −4, 1, −3, −2}, find the ***Maximum-sum Continuous Subarray*** using divide-and-conquer. You must show the **recursion tree** and clearly mention **left, right and crossing sum** for each tree node.

Solution:



∴ Maximum sub continuous subarray is:

1

2

4. a) Following items are available in a grocery shop:

- 10 kilogram rice grain which costs 800 taka
- 10 kilogram salt which costs 890 taka
- 8 kilogram saffron powder which costs 2000 taka and
- 4 kilogram sugar which costs 500 taka

A group of thieves (Thief 1, Thief 2, … Thief M) have come to steal from that shop, **each with a knapsack of capacity 8 kg**. The thieves are entering in serial, *Thief 2* enters after *Thief 1* is done with stealing, *Thief 3* enters after *Thief 2* is done with stealing and so on. *Since each thief wants to maximize his/her profit*, **how many thieves** will be needed in the group to empty the grocery shop and **what are the items** that each of those thieves carry? Show details of the calculation.

Solution:

Given,

Capacity of each thief, $W = 8\ kg$

Item	Rice Grain	Salt	Saffron Powder	Sugar
Value (taka), v	800	890	2000	500
Weight (kg), w	10	10	8	4

Finding price per unit:

[P.T.O]

$\frac{v_i}{w_i}$	80	89	250	125
	Rice Grain	Salt	Saffron Powder	Sugar

Sorting item based on $\frac{v_i}{w_i}$:

Saffron Powder > Sugar > Salt > Rice Grain

Thief 1:

Item	v_i	w_i	W	$x_i = \min\left(1, \frac{W}{w_i}\right)$	$W' = W - w_i x_i$
Saffron Powder	2000	8	8	1	8-8 = 0

Thief 2:

Item	v_i	w_i	W	$x_i = \min\left(1, \frac{W}{w_i}\right)$	$W' = W - w_i x_i$
Sugar	500	4	8	1	8-4 =4
Salt	890	10	4	0.4	4-(10×0.4) = 4-4 = 0

Now, Salt new weight, $w_i' = w_i - w_i x_i = 10 - 10 \times 0.4 = 6$

Salt new value, $v_i' = v_i - v_i x_i = 890 - 890 \times 0.4 = 534$

[P.T.O]

Thief 3:

Item	v_i	w_i	W	$x_i = \min\left(1, \frac{W}{w_i}\right)$	$W' = W - w_i x_i$
Sugar	534	6	8	1	8-6 = 2
Rice Grain	800	10	2	0.2	2-(10×0.2) = 2-2 = 0

Now, Rice Grain new weight, $w_i' = w_i - w_i x_i = 10 - 10 \times 0.2 = 8$
Rice Grain new value, $v_i' = v_i - v_i x_i = 800 - 800 \times 0.2 = 640$

Thief 4:

Item	v_i	w_i	W	$x_i = \min\left(1, \frac{W}{w_i}\right)$	$W' = W - w_i x_i$
Rice Grain	640	8	8	1	8-8 = 0

Here, all items in the grocery shop have been stolen.
∴ Total 4 thieves will be needed in the group to empty the grocery shop.

2. b) A document to be transmitted over the internet contains the following characters with their associated frequencies as shown in the following table:

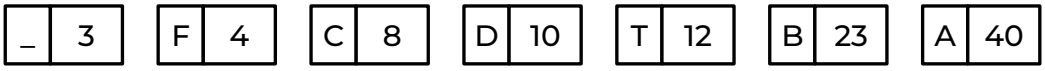
Character	A	B	C	D	F	T	_
Frequency	40	23	8	10	4	12	3

There are a total 1000 characters in the document.

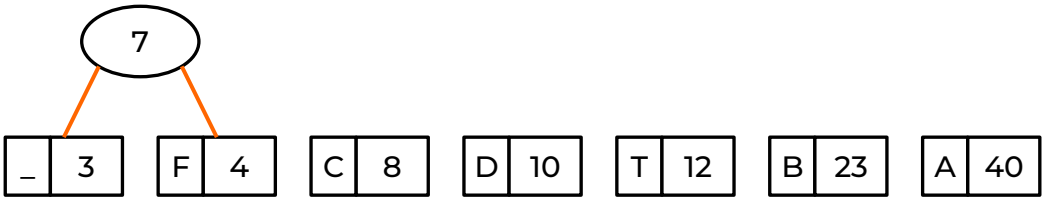
i. **Build** the Huffman code tree for the message and find the **codeword** for each character.
ii. Decode **"0110001111"** using codewords generated in (i).

Solution:

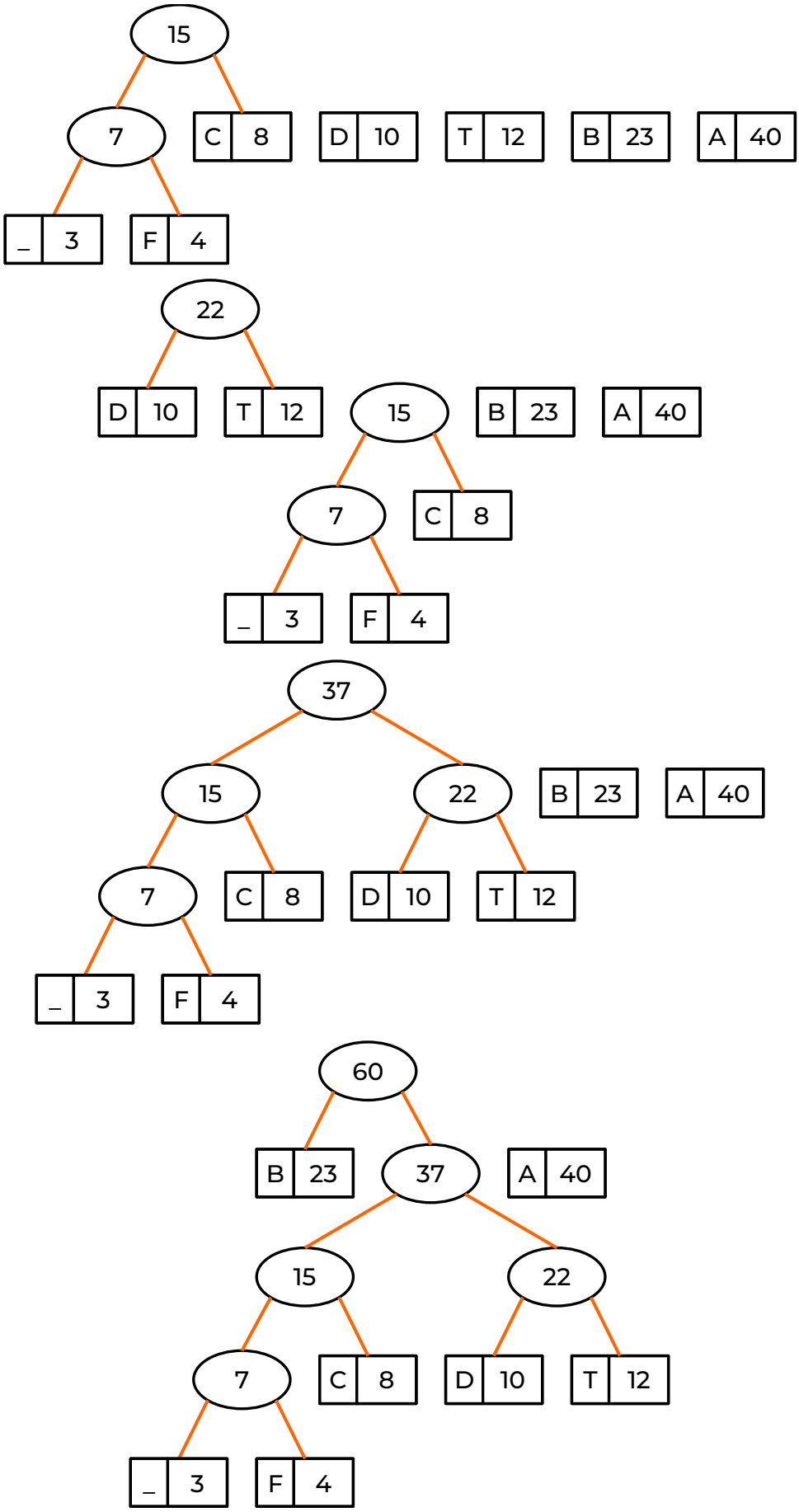
i. Sorting character based on frequency:



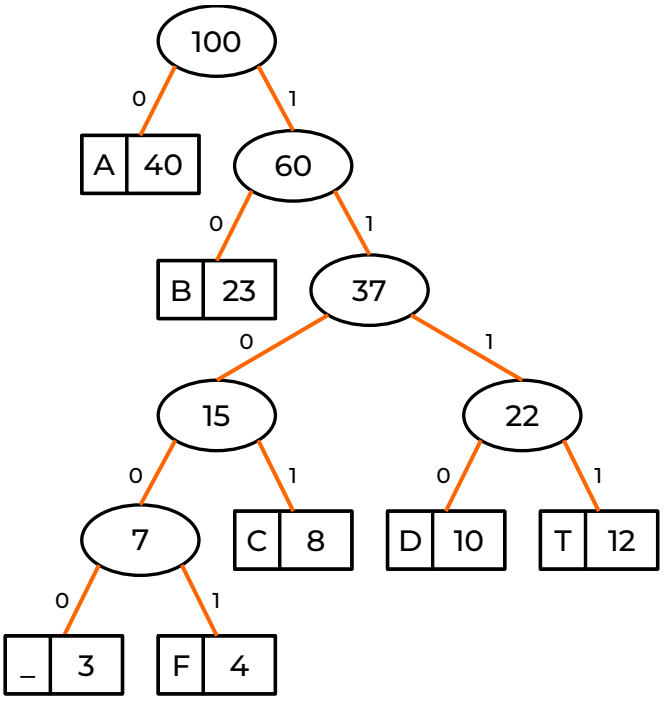
Building Huffman code tree:



[P.T.O]


15
7
C 8
D 10
T 12
B 23
A 40
_ 3
F 4
22
D 10
T 12
15
B 23
A 40
7
C 8
_ 3
F 4
37
15
22
B 23
A 40
7
C 8
D 10
T 12
_ 3
F 4
60
B 23
37
A 40
15
22
7
C 8
D 10
T 12
_ 3
F 4

[P.T.O]



This is our final Huffman coding tree.

Finding codeword for each character from Huffman tree:

A: 0 B: 10
C: 1101 D: 1110
F: 11001 T: 1111
_: 11000

ii. Decoding "0110001111" using codewords from (i):

0110001111 = A_T

3. a) Suppose you have computed a **Fibonacci** series using **dynamic programming**. **Justify** the following statements with **an example**:

I. **Overlapping Subproblems** property has been satisfied in your computation.

II. Dynamic programming gives you a **more efficient** solution than an obvious recursive algorithm.

Solution:

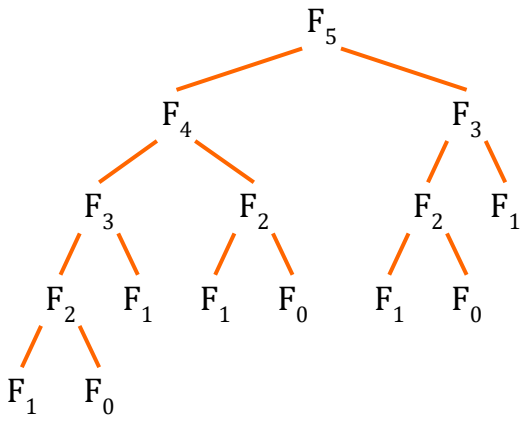
The overlapping sub-problems property means that a problem can be broken down into sub-problems that are reused multiple times. When compute a Fibonacci number, we get same sub-problems for solving repeatedly.

For example, for computing $F(5)$,

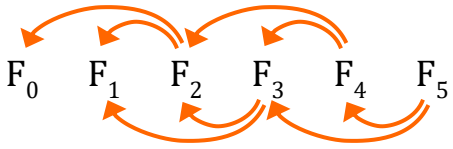
$F(5)$ requires $F(4)$ and $F(3)$
$F(4)$ requires $F(3)$ and $F(2)$
$F(3)$ requires $F(2)$ and $F(1)$
$F(2)$ requires $F(1)$ and $F(0)$

As a result, $F(3)$ is computed twice, $F(2)$ is computed multiple times, and so on. This repetition indicates overlapping sub-problems. In recursive algorithm, we solve and merge solution of each sub-problems with overlapping sub-problems. But in dynamic programming, we store the results of these sub-problems and reuse them and reduce the inefficiency which created on recursive algorithm.

Recursive approach for calculating F_5:



Dynamic programming approach for calculating F_5:



3. b) What is 'Optimal Substructure' property? How does Dynamic Programming differ from Divide-and-Conquer problems in terms of handling subproblems?

Solution:

A problem is said to have optimal substructure if an optimal solution can be constructed from optimal solutions of its sub-problems. In other words, a problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to its sub-problems.

A problem can have some repeated sub-problems, which are known as overlapping sub-problems. In Divide-and-Conquer algorithm, it solve each of the sub-problems, including repeated overlapping sub-problems. It create inefficiency for many problems, such as Fibonacci number calculating. In Fibonacci number problem, we have so many repeated sub-problems multiple time. Recursive algorithm solve and merge each of repeated sub-problems. But Dynamic Programming solve and store sub-problems solution and use it later and reduce inefficiency. It is the main difference in terms of handling sub-problems between Divide-and-Conquer and Dynamic Programming algorithm.

3. b) Suppose, CoffeeLand Coffee Shop charges **50 BDT** (Bangladesh Taka) for each cup of small Americano with an additional vat of **3%**. You bought 2 cups of small Americano and gave the cashier **110 taka**. The cashier has got a huge supply of the following types of coins: **1 taka, 2 taka, and 5 taka** in the cashbox. You don't want to carry many coins, so you asked the cashier to return the change using a **minimum number of coins**. Determine **the number** and **type of coins** the cashier should return in this scenario by applying the **Dynamic Programming** Approach.

Solution:

Given,

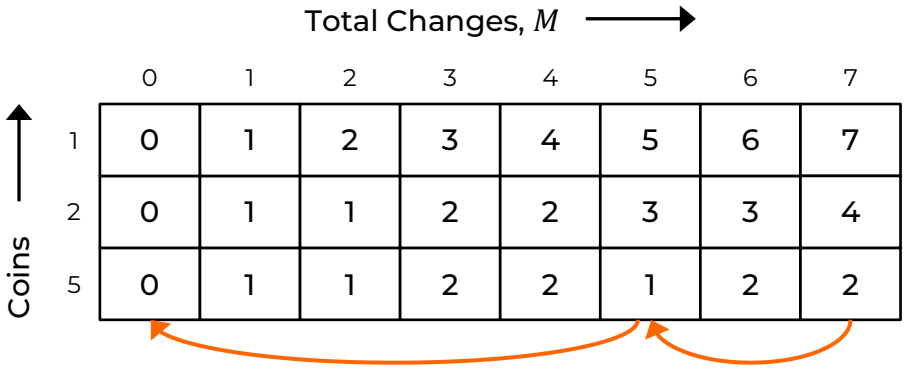
$$\text{Total charges} = (50 \times 2) + [(50 \times 2) \times 3\%]$$
$$= 100 + (100 \times 3\%)$$
$$= 100 + 3$$
$$= 103$$

$$\text{Given cash} = 110$$

$$\therefore \text{Total Changes}, M = 110 - 103$$
$$= 7$$

$$\text{Coins}, C_i = \{C_1, C_2, C_3\} = \{1,2,5\}$$

Now,



∴ Minimum coins needed = 2
Selected coins = 1 coins of 2 Taka, 1 coins of 5 Taka

4. a) Derive the best-case and the worst-case running time equations for the following function ***calculate*** and represent using Asymptotic Notation.

```
1   void calculate(int n, int p, int A[]) {
2       int prod = 0;
3       for(int i = 1; i<=n; i++) {
4           for(int j = 1; j <= i*i; j++) {
5               prod *= pow(i,j)
6           }
7       }
8
9       for(int m = 2; m <= P; m++) {
10          if(A[m] < 100 ) {
11              break;
12          }
13
14          prod = prod * A[m];
15      }
16
17      cout<<prod<<endl
18  }
```

Solution:

Line	Worst Case	Best Case
2	1	1
3	$n+1$	$n+1$
4	n^3+n	n^3+n
5	n^3	n^3
9	p	1
10	0	1
11	0	1
14	$p-1$	0

17	1	1

$\therefore$ Best case, $T(n) = 1 + n + 1 + n^3 + n + n^3 + 1 + 1 + 1 + 1$
$= 2n^3 + 2n + 6$
$= O(n^3)$

$\therefore$ Worst case, $T(n) = 1 + n + 1 + n^3 + n + n^3 + p + p - 1 + 1$
$= 2n^3 + 2n + 2p + 2$
$= O(n^3)$ [Considering $n^3 > p$]

4. b) Derive the exact-cost equation for the running time of the following function and show that time complexity in $O(n \log n \log_5 n)$:

```
1   int funFunction(int n)
2   {
3       int sum = 0;
4       for (int k = 0; k < n; k*=2){
5           for (int j = 2/n; j <= n; j++) {
6               for (int i = n; i >= 1; i=i/5) {
7                   sum += (i+j+k);
8               }
9           }
10      }
11
12      cout<<sum<<endl;
13
14  }
```

Solution:

Line	Time Cost
3	1
4	$\log_2 n + 1$
5	$\log_2 n \times \frac{n}{2} + \log_2 n$
6	$\log_2 n \times \frac{n}{2} \times \log_5 n + \log_2 n \times \frac{n}{2}$
7	$\log_2 n \times \frac{n}{2} \times \log_5 n$
10	1

$\therefore$ Time complexity, $T(n) = 1 + \log_2 n + 1 + \log_2 n \times \frac{n}{2} + \log_2 n + \log_2 n \times \frac{n}{2} \times \log_5 n$
$+ \log_2 n \times \frac{n}{2} + \log_2 n \times \frac{n}{2} \times \log_5 n + 1$
$= 1 + \log_2 n + 1 + \frac{1}{2} n \log_2 n + \log_2 n + \frac{1}{2} n \log_2 n \log_5 n$
$+ \frac{1}{2} n \log_2 n + \frac{1}{2} n \log_2 n \log_5 n + 1$

$$= 2\log_2 n + n\log_2 n + n\log_2 n \log_5 n + 3$$
$$= O(n \log n \log_5 n)$$

∴ Exact cost equation is $2\log_2 n + n\log_2 n + n\log_2 n \log_5 n + 3$ and time complexity is $O(n \log n \log_5 n)$. (Showed)