

CS263: Design and Analysis of Algorithm

Name: Hritik Kumar

Roll no.: 202051088

1 Problem 1

Let us assume you got the assignments on different courses 1 to C. For each assignment, let d_i denote the number of days required to complete the assignments. For each day of delay before starting your i^{th} assignment, a penalty of p_i is imposed. We are required to find a sequence to complete the assignments so that the overall penalty is minimized. We can only work on one assignment at a time and if you start any assignment finish it then start another. Write an efficient algorithm for the given problem.

Algorithm:

Since the net penalty depends on two different factors, we have to consider both of them, hence we will be needing a greedy approach for this.

Penalty is directly proportional to number of days it takes and is inversely proportional to penalty per day, therefore we will look for the least value of d_i/p_i .

CODE:

```
import java.util.*;

public class Jobs {

    public static void sort(double[] arr, int[] p, int[] l)
    {
        int n = arr.length;
        for (int i = 1; i < n; ++i) {
            double key = arr[i];
            int key1 = p[i];
            int key2 = l[i];
            int j = i - 1;

            while ((j >= 0) && (arr[j] > key)) {
                arr[j + 1] = arr[j];
                p[j + 1] = p[j];
                l[j + 1] = l[j];
                j--;
            }
            arr[j + 1] = key;
            p[j + 1] = key1;
            l[j + 1] = key2;
        }
    }
}
```

```

        l[j + 1] = l[j];

        j = j - 1;
    }
    arr[j + 1] = key;
    p[j + 1] = key1;
    l[j + 1] = key2;
}
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    System.out.println("Enter i :");
    int i = sc.nextInt();

    int[] d = new int[i];
    System.out.println("Enter deadlines");
    for (int a=0;a<i;a++){
        d[a] = sc.nextInt();
    }

    int[] p = new int[i];
    System.out.println("Enter penalties");
    for (int a=0;a<i;a++){
        p[a] = sc.nextInt();
    }

    System.out.println(Arrays.toString(p));
    System.out.println(Arrays.toString(d));

    double[] ratio = new double[i];
    for (int a=0;a<i;a++){
        ratio[a] = d[a]/(double)p[a];
    }

    sort(ratio,p,d);

    System.out.println(Arrays.toString(ratio));
    System.out.println(Arrays.toString(p));
    System.out.println(Arrays.toString(d));
}

```

```

        int sum = 0, time =0;
        for(int x=0; x<i-1; x++){
            time += d[x];
            sum += p[x+1]*time;
        }
        System.out.println("Total Penalty is: " +sum);
    }
}

```

Output:

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\Lenovo\Desktop\c++files\Assignment\Assignment 7> java Jobs
Enter i :
3
Enter deadlines
4 6 7
Enter penalties
2 1 4
[2, 1, 4]
[4, 6, 7]
[1.75, 2.0, 6.0]
[4, 2, 1]
[7, 4, 6]
Total Penalty is: 25
PS C:\Users\Lenovo\Desktop\c++files\Assignment\Assignment 7>

```

Time complexity: -

Complexity to create map: - $O(n)$

Complexity to sort the map by value: - $O(n \log n)$

Complexity to get the key value to result list: - $O(n)$

Net Time Complexity: - $O(n \log n)$

2 Problem 2

Event Organiser team has decided to take few classroom on permission from the Goverment Engineering college to schedule the 1-day Tech event of IIIT Vadodara. They got the start and end timing of all the events, the task is to find the minimum number of classroom required for the events so that no event will wait or get cancelled. We are given two arrays that represent the start and end times of events that need to be conducted.

Algorithm:

We initially must sort the arrays of time of start and time of end so that we can align the start and end time of two consecutive event, After that, we take two pointers and iterate through the start time array and end time array and if the start time of next pointed event is less than the end time of previously pointed pointer then we need one extra room but the one event got finished before starting the next event then one room is going to get empty therefore, we need to have a count of currently how many room are occupied and the max value of this count will be the number of halls we need.

CODE:

```
import java.util.*;
class minMeetingRooms
{
    static int minclassroom(int start[], int end[])
    {
        int l= start.length;
        if(l<=1)
        {
            return l;
        }

        Arrays.sort(start);
        Arrays.sort(end);

        int startpointer = 0;
```

```
int endpointer = 0;
int rooms = 0;

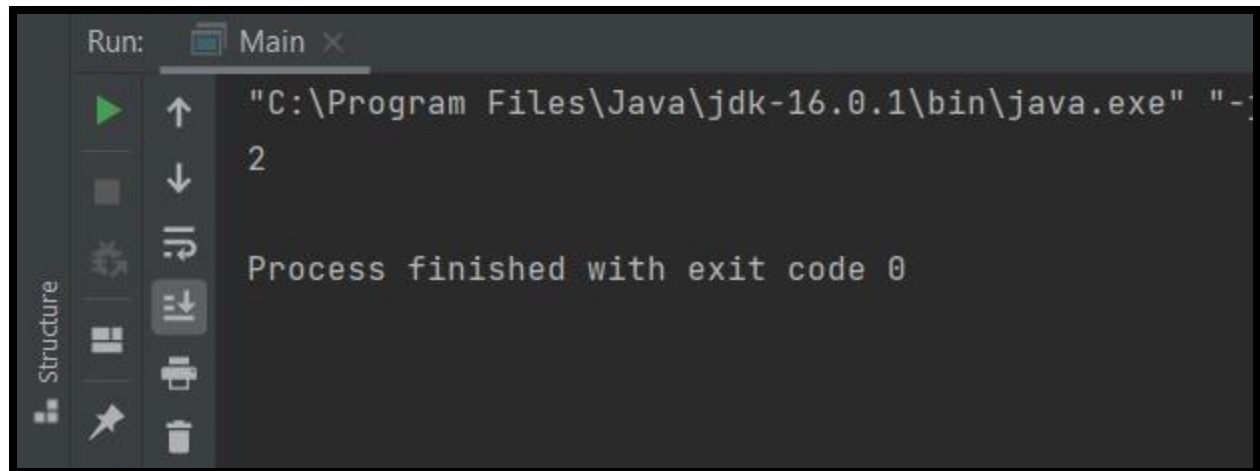
while(startpointer < l)
{
    if(start[startpointer++] >= end[endpointer])
    {
        ++endpointer;
    }
    else
    {
        ++rooms;
    }
}

return rooms;
}

public static void main(String[] args) {
    int start[] = {0,5,15};
    int end[] = {30,10,20};

    int minRooms = minclassroom(start,end);
    System.out.println(minRooms);
}
}
```

Output:



```
Run: Main x
"C:\Program Files\Java\jdk-16.0.1\bin\java.exe" "-..."
2
Process finished with exit code 0
```

Time complexity: -

Time complexity to sort the arrays = $O(2n \log n)$

Time complexity to iterate through the arrays = $O(2n)$

Therefore,

Net Time Complexity = $O(n \log n)$

3 Problem 3

There are N stations on the route of a train Gandhinagar to Jaipur. If there exists any direct train between any two stations i and j , then the ticket cost for all pairs of stations (i, j) is given where j is greater than i . For example: There are Five stations A, B, C, D. A is the source and D is the destination.

Cost of route A-B - Rs100 by Train X,

Cost of route A-C- 200 by Train Y,

Cost of route A-D- 300 by Train Z

Cost of route B-C-No train

Cost of route B-D- 100 by train W

Cost of route C-D- 250 by train V

Find the minimum cost route that will help you to reach the destination at minimum cost.

Algorithm:

Start

Recursive method to find min cost.

Declare min variable, which is the minimal cost for each route.

When the destination is reached, return the minimal cost for that path.

The for loop starts from the initial position to the destination with the recursive approach.

Finally return the minimal cost out of all the routes for each path.

End

CODE:

```
import java.util.*;

class MinCost {
    static int highCost = 1000000, N = 4;
    public static void main(String[] args){

        int [][]cost = { {0, 100, 200, 300},
                          {highCost, 0, highCost, 100},
                          {highCost, highCost, 0, 250},
                          {highCost, highCost, highCost, 0}
        };
        System.out.println("The Minimum cost to reach station " + N +
" is " + min_cost(0, N-1, cost));
    }
    public static int min_cost(int initial, int dest, int [][] arr){
        int min_sum = arr[initial][dest];
        if(min_sum < 0)
            return highCost;
        if(initial == dest || initial + 1 == dest){
```



```
        return min_sum;
    }
    for (int i = initial + 1; i < dest; i++){
        int sum = min_cost(initial, i, arr) + min_cost(i, dest,
arr);
        if(min_sum > sum)
            min_sum = sum;
    }
    return min_sum;
}
```

Output:

PROBLEMS 19 OUTPUT DEBUG CONSOLE TERMINAL

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

PS C:\Users\Lenovo\Desktop\c++files\Assignment\Assignment 7> javac MinCost.java

PS C:\Users\Lenovo\Desktop\c++files\Assignment\Assignment 7> java MinCost

The Minimum cost to reach station 4 is 200

PS C:\Users\Lenovo\Desktop\c++files\Assignment\Assignment 7> █