Job Sequencing Problem | Greedy Approach

Problem: Given an array of jobs where every job has a deadline and associated profit if the job is finished before the deadline. It is also given that every job takes a single unit of time, so the minimum possible deadline for any job is 1. How to maximize total profit if only one job can be scheduled at a time.

Examples:

Input: Four Jobs with following deadlines and profits

JobID	Deadline	Profit
a	4	20
b	1	10
c	1	40
d	1	30

Output: Following is maximum profit sequence of jobs
c, a

Input: Five Jobs with following deadlines and profits

JobID	Deadline	Profit
a	2	100
b	1	19
c	2	27
d	1	25
e	3	15

Output: Following is maximum profit sequence of jobs

```
c, a, e
```


This is a standard Greedy Algorithm problem. Following is the greedy algorithm to solve the above problem:

```
1) Sort all jobs in decreasing order of profit.
2) Initialize the result sequence as the first job in sorted jobs.
3) Do following for remaining n-1 jobs
.......a) If the current job can fit in the current result sequence
          without missing the deadline, add the current job to the result.
          Else ignore the current job.
```

Implementation:

C++ Java

```java
// Java program to find the maximum profit
// job sequence from a given array
// of jobs with deadlines and profits
import java.util.Arrays;

// A class to represent a job
class Job implements Comparable<Job>
{
    char id; // Job Id
    int dead; // Deadline of job

    // Profit if job is over
```

```
    // before or on deadline
    int profit;

    Job(char id, int dead, int profit) {
        this.id = id;
        this.dead = dead;
        this.profit = profit;
    }
    // This function is used for sorting all
    // jobs according to decreasing order of profit
    @Override
    public int compareTo(Job o) {
        if(this.profit < o.profit)
            return 1;
        return -1;
    }
};

class GFG
{
    // Returns minimum number of platforms reqquired
    static void printJobScheduling(Job arr[], int n)
    {
        // Sort all jobs according to
        // decreasing order of prfit
        Arrays.sort(arr);

        // To store result (Sequence of jobs)
        int result[] = new int[n];
        // To keep track of free time slots
        boolean slot[] = new boolean[n];

        // Initialize all slots to be free
        for (int i=0; i<n; i++)
                slot[i] = false;

        // Iterate through all given jobs
        for (int i=0; i<n; i++)
        {
            // Find a free slot for this job
            // (Note that we start
```

```
                // from the last possible slot)
                for (int j=Math.min(n, arr[i].dead)-1; j>=0; j--)
                {
                    // Free slot found
                    if (slot[j]==false)
                    {
                        result[j] = i; // Add this job to result
                        slot[j] = true; // Make this slot occupied
                        break;
                    }
                }
            }

            // Print the result
            for (int i=0; i<n; i++)
                if (slot[i])
                    System.out.print(arr[result[i]].id + " ");
        }

        // Driver Code
        public static void main(String args[])
        {
            Job arr[] = {new Job('a', 2, 100),
                         new Job('b', 1, 19),
                         new Job('c', 2, 27),
                         new Job('d', 1, 25),
                         new Job('e', 3, 15)};

            int n = arr.length;

            System.out.print("Following is maximum profit"
                             +" sequence of job : ");

            printJobScheduling(arr, n);
        }
    }
```

Output:

```
Following is maximum profit sequence of job : c a e
```

Time Complexity of the above solution is $O(n^2)$. It can be optimized using Disjoint Set Data Structure. Please refer below post for details.