JOB SEQUENCING WITH DEADLINES

I Problem Definition

- There are 'n' jobs to be processed on a machine.
- Each job 'i' has a deadline $d_i \geq 0$ and profit $P_i \geq 0$.
- P_i is earned if the job is completed by its deadline.
- The job is completed if it is processed on a machine for unit time.
- Only one machine is available for processing jobs.
- Only one job is processed at a time on the machine.
- A feasible solution is a subset of jobs J such that each job is completely by its deadline.
- An optimal solution is a feasible solution with maximum profit value.

II Algorithm (Variation 1)

Input: (i) $D(i) \geq 1$, $1 \leq i \leq n$ are deadlines, $n \geq 1$.

(ii) $P(i) \geq 0$, $1 \leq i \leq n$ are profit, $n \geq 1$

Output: J → optimal solution

$J(i)$ is the i^{th} job in the optimal solution $1 \leq i \leq k$.

Constraints: Jobs are ordered such that $P_1 \geq P_2 \geq \cdots P_n$

Algorithm:

JobSequencing (D, J, n) — D: deadline array, J: optimal solution, n: Total No. of jobs

```
{
1.  D(0) = J(0) = 0          # Initialize J(0) is fictious job
2.  k = 1 ; J(1) = 1         # Job one is inserted
3.  for i = 2 to n do        # Jobs are in non decreasing order of Pi.
    {
4.      r = k                # r and k are indices of existing job in J
5.      while ( D(J(r)) > D(i) and      # Job r can be processed after i
                D(J(r)) ≠ r )           # deadline of r is not exactly r
        {
6.          r = r - 1
        }                    # Consider whether r-1 can be processed after i.
#       if ( D(J(r)) ≤ D(i) and D(i) > r
        {                    # new job i can come after existing job r
            insert i into J at pos r+1
8.          for q = k to r+1 by -1 do      (q >= r+1)
            {
9.              J(q+1) = J(q)
            }
90.         J(r+1) = i ; k = k+1
        }
    }
}
```

TSEC
ENGINEERING COLLEGE

Analysis :

Complexity of Prim's depends on
(1) n - Total number of jobs
(2) s - Total number of jobs in optimal solution.

Line 3 , for loop will be executed (n-1) times.
Line 5, while loop will be executed at most 'k' times.

If conditions in line 7 is true, then for inserting internal for loop will run (k-r) time to insert job i.

∴ Total running time = Θ (n * total no. of jobs in optimal soln)

= Θ (ns)

In worst case, when all job is to be executed then s = n

∴ Running time = $\Theta(n^2)$.

Example ①

let n = 5

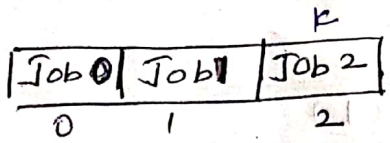
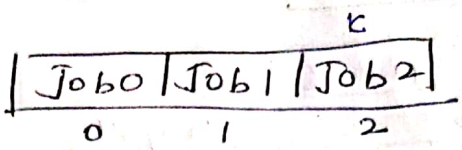
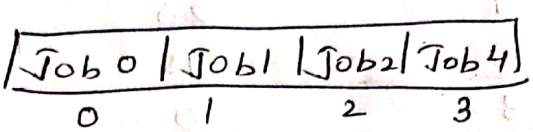
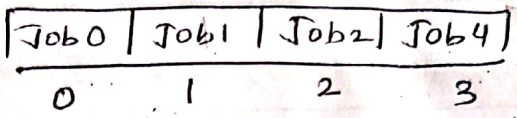
Jobs	Profit	Deadline
1	20	2
2	15	2
3	10	1
4	5	3
5	1	3

Step 1: Sorting based on profit in non decreasing order.

Jobs	Profit	Deadline
1	20	2
2	15	2
3	10	1
4	5	3
5	1	3

Step 2: Constructing optimal solution

Iteration	Job being considered	J	action
0	—	Job 0 (slot 0)	—
1	Job 1	Job 0 | Job 1 (slots 0, 1)	assigned to slot [0, 1]

2	Job 2		assigned slot [1, 2]
3.	Job 3		cannot fit. Reject Job 3.
4.	Job 4		assigned to slot [2, 3]
5.	Job 5		cannot fit reject Job 5

∴ Total profit = 20 + 15 + 5
= 40

The optimal solution is $J = \{1, 2, 4\}$

Set of feasible solutions is
{2, 2, 5}, {3, 2, 4}, {3, 2, 5}
{3, 1, 4}, {3, 1, 5} etc..

Example 2

Job	Profit	Deadline
1	100	2
2	10	1
3	15	2
4	27	1

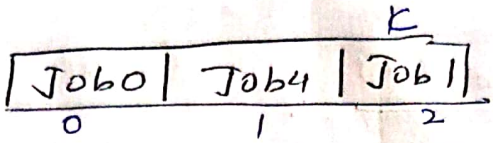
Step 1, Sorting based on profit in non-decreasing order.

Jobs	Profit	Deadline
1	100	2
4	27	1
3	15	2
2	10	1

Step 2: Constructing Optimal Solution.

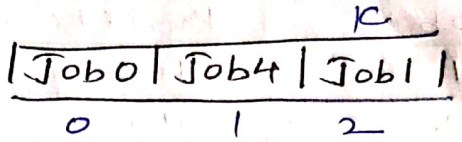
Iteration	Job being considered	J	action.
0	–	Job 0 (0 K)	–
1	Job 1	Job 0 | Job 1 (0 1)	assigned to slot [0,1]
2.	Job 4	Job 0 | Job 4 | Job 1 (0 1 2)	moved Job 1 to slot [1,2] and assigned slot [0,1] to Job 4

3. Job 3



cannot fit reject job1

4. Job 2



cannot fit Job 2, reject job 2

∴ Total profit = 100 + 27
= 127

The optimal solution is $J = \{4, 1\}$

Set of feasible solutions is,

$\{2, 1\}$. $\{1, 3\}$, $\{3, 1\}$, $\{4, 3\}$
$\{2, 3\}$.

STE

TSEC
ENGINEERING COLLEGE

Algorithm : (Variation 2)

Input, output and constraint are same as variation 1.

Jobset [1....n] will be initialized to zero. It contains jobs included in an optimal solution.

slot [1...n] will be initialize to false.

```
JobSequencing ( d, p, n)
{
1.  for  i = 1  to  n  do.
    {
2.      for  j = d [i]  to  0  by  -1
        {
3.          if (slot[j] == false)
            {
4.              slot[j] = true
5               Jobset[j] = i
6.              break
            }
        }
    }
7.  return  Jobset
```

Complexity is $O(n^2)$