Job Sequencing with deadlines :-

Jobs to be done on a single machines

→ Job i takes unit time

→ has a deadline d_i, $d_i \geq 0$

→ has a profit P_i, $P_i \geq 0$

P_i is paid only if i is completed within its deadline d_i {if deadline is n, the job should be completed by n^{th} second}

Machine can only do one job at time

Select a feasible subset of jobs to maximize profit.

Ex :-

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

All feasible solutions

Solution	{1,2}	{1,3}	{1,4}	{2,3}	{3,4}	{1}	{2}	{3}	{4}
Order of Processing	2,1	1,3 or 3,1	4,1	2,3	4,3	1	2	3	4
Value	110	115	127	25	42	100	10	15	27
		×	optimal						

Greedy Strategy : Order by decreasing profit

{1, 4, 3, 2}

For each job that we try out, we should verify that by adding that job we are preserving feasibility (ie deadlines are met).

High level algorithm for the greedy solution

Assume that : $P_1 \geq P_2 \geq P_3 \cdots \geq P_n$, $d_i > 0$

```
Algorithm GreedyJob(d, J, n)
// d = sequence of deadlines
// n = number of jobs
// J = feasible subset of solutions that we
// are going to pick and return.
{
    J := {1}; // very first job is always feasible
              // because all jobs can be scheduled
              // in the 1st second.
    for i := 2 to n do
    {
        if all deadlines in J ∪ {i} are feasible
        then J := J ∪ {i}; // how to pick
                           // job?
    }
    return J;
}
```

The question is given a set of jobs J, when is J feasible?

we can try brute force method, try all permutations and check the feasibility but it is an highly inefficient method.

∴ we make an <u>observation</u> that

J can be scheduled in increasing order of deadlines.

Theorem: if $J = \{1, 2, \ldots k\}$, with the assumption that $d_1 \leq d_2 \leq d_3 \leq \cdots \leq d_k$, then J is feasible iff $1, 2, 3, \ldots k$ is a feasible order.

Using the above fact

To check if J is feasible

i) Sort it by deadlines

ii) check that the deadline of job i, $d_i \geq i$

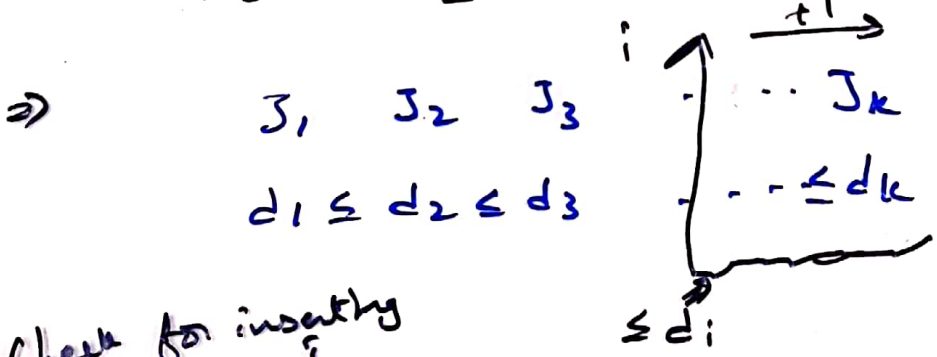
(Checking that each job meets its deadline, because job i would be scheduled at i^{th} second).

Using the above theorem to solve the problem of extending J to $J \cup \{i\}$

i.e $J \longrightarrow J \cup \{i\}$ (current solution)

assuming $J_1, J_2, J_3 \ldots J_k$ are arranged in order of deadlines

i.e. $d_1 \leq d_2 \leq d_3 \ldots \leq d_k$



Now extend this to include i which comes in with deadline d_i

Check for inserting i

(1) i is feasible

$d_i \geq$ insert position

(2) Every thing to its right is now going to be shifted by 1, all these deadlines were feasible when in their current positions, are they still feasible when they are shifted by 1 position to their right.

If the above conditions hold, then i can be added to the job list J, otherwise, i.e if any one of the conditions fail then job i is discarded and go to the next job.

Detailed solution to the job scheduling problem using greedy strategy :-

```
Algorithm JS(d, J, n)
{   d[0] := J[0] := 0 ;  // dummy value for inserting.
    J[1] := 1 ; // containing job 1
    k := 1 ; // size of J   i.e. J = J0, J1, J2 ... Jk
                                 (J0 = dummy job; k ↑ r)
k ← for i := 2 to n do
    {
        r := k ;  // checks the deadline of new job is smaller than the deadline we are looking
        while( d[J[r]] > d[i] and
               d[J[r]] != r ) do      ] k times
                  // checking if inserting it is not effecting the feasibility of other jobs in the list already.
            r := r - 1 ;   // we walk left with r := r-1
        if( d[J[r]] <= d[i] and
                d[i] > r ) then
        ↓ checks that job i is feasible hence can be included into set J.
        {   for q := k to (r+1) step -1 do
                J[q+1] := J[q] ;          ] k-r
            J[r+1] := i ;
            k := k+1 ;
        }
    }
    return J ;
}
```

if S is the final value of k, i.e. S is no. of jobs in final solution ∴ JS is $\Theta(Sn)$

Since $S \le n$, worst case $\Theta(n^2)$

Deducted to $\Theta(n)$ by disjoint set union and find alg.

①

Ex:

New Job no	I	II	III	IV	V	VI
Job no:	6	3	4	2	5	1
Profit	99	67	45	34	23	10
Deadline	2	3	1	4	5	3

① Job I is allotted ~~slot~~ slot [0, 1]

new Job no:	I					
ordered by deadlines	2					
Job no.	6					

② Job II is allotted ~~slot~~ slot [1, 2], the deadlines are in increasing length.

New Job no.:	I	II				
Ordered by deadlines	2	3				
Job no.	6	3				

③ Job III is being considered. Deadline is 1, So we have to shift jobs I and II upward

New Job no.	I →	II →				
Ordered by deadlines	2	3				
Job no.:	6	3				

New Job no.	III	I	II			
Ordered by deadlines	1	2	3			
Job no.	4	6	3			

The deadlines are in increasing order

Job IV has a deadline of 4, so it can be allotted slot [3, 4]

New Job no.	III	I	II	IV		
Ordered by deadlines	1	2	3	4		
Job no.	4	6	3	2		

Job V has a deadline of 5, so it can be allotted slot [4, 5]

New Job no.	III	I	II	IV	V	
Ordered by deadlines	1	2	3	4	5	
Job no.	4	6	3	2	5	

Job VI has a deadline of 3 but we cannot shift the array to the left, so we reject job VI

∴ The above is a the schedule.