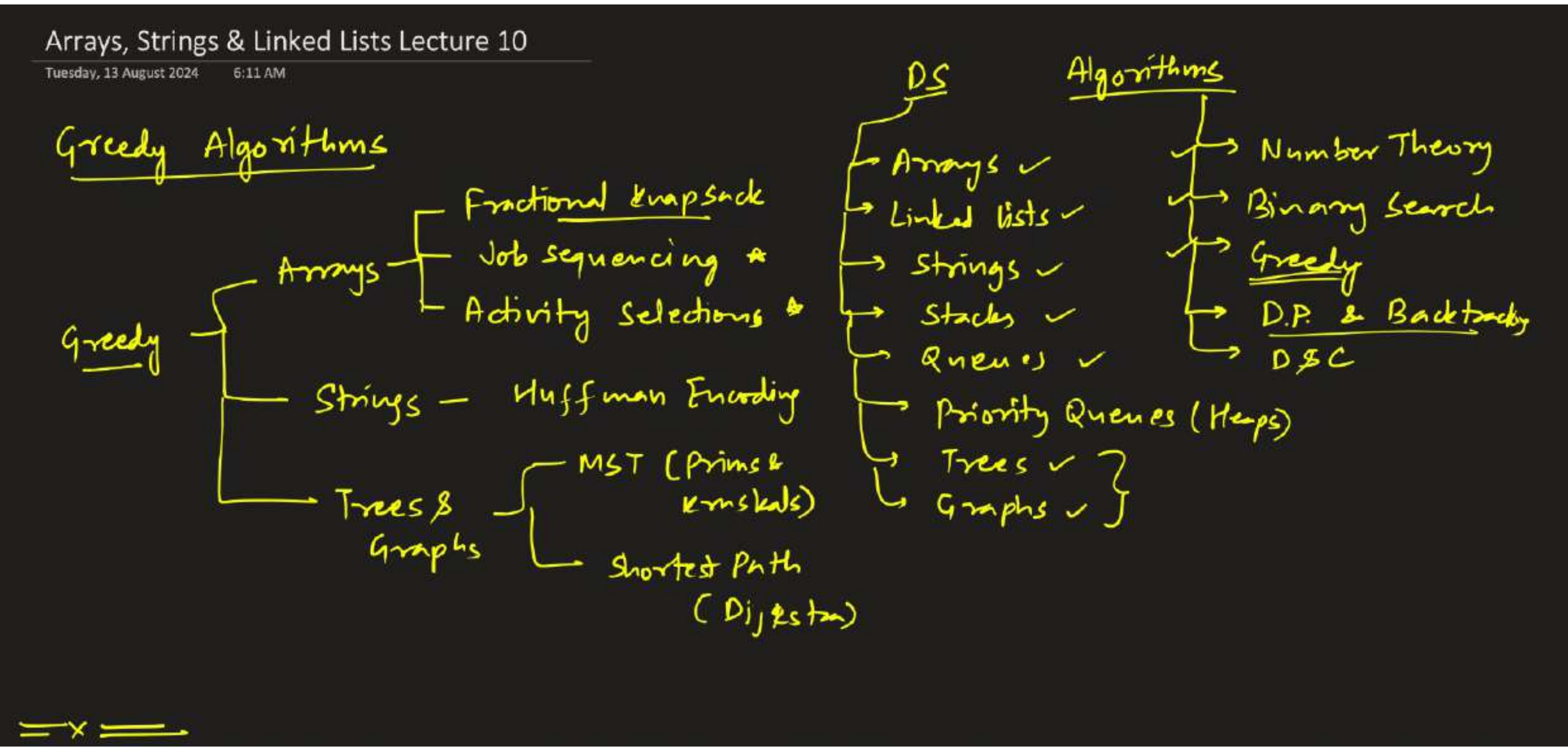


Arrays, Strings & Linked Lists Lecture 10

Tuesday, 13 August 2024 6:11 AM

Greedy Algorithms

Greedy
- Arrays
  - Fractional knapsack
  - Job sequencing *
  - Activity Selections *
- Strings — Huffman Encoding
- Trees & Graphs
  - MST (Prims & Kruskals)
  - Shortest Path (Dijkstra)

DS
- Arrays ✓
- Linked lists ✓
- Strings ✓
- Stacks ✓
- Queues ✓
- Priority Queues (Heaps)
- Trees ✓
- Graphs ✓

Algorithms
- ✓ Number Theory
- ✓ Binary Search
- ✓ Greedy
- D.P. & Backtracking
- D&C

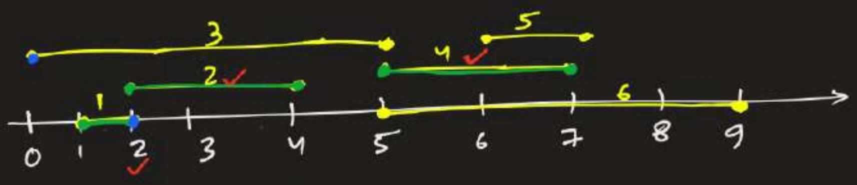
=x=

=x==

Activity Selection Problem

n Activities with start & end time. Select max. number of activities that can be done by a single person if the person can work on a single activity at a time.

eg: [1, 2], [2, 4], [0, 5], [5, 7], [6, 7], [5, 9]



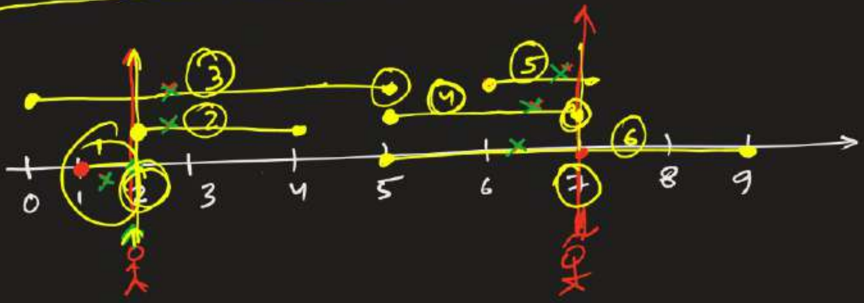
[1,2] [2,4] [5,7]
1 2 4

Greedy choice:- Pick the next activity with least finish time among remaining activities & start time ≥ finish time of current activity.

https://leetcode.com/problems/minimum-number-of-arrows-to-burst-balloons/

a[1] < b[1]

eg: [[1, 2], [2, 4], [0, 5], [5, 7], [6, 7], [5, 9]]

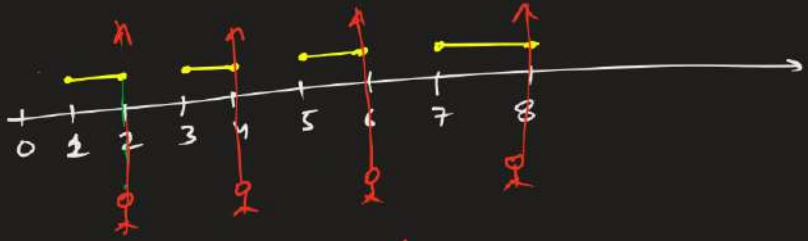


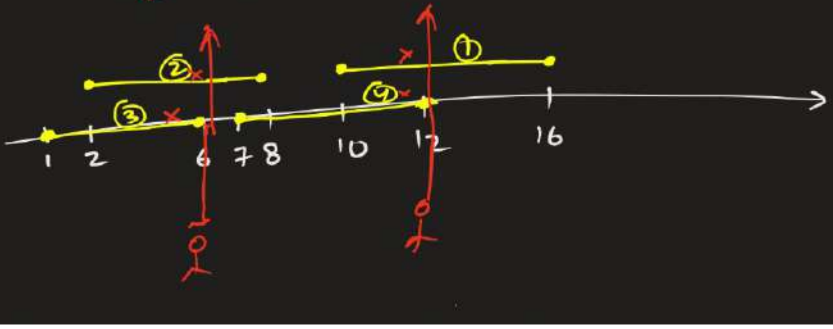
min arrows = 2

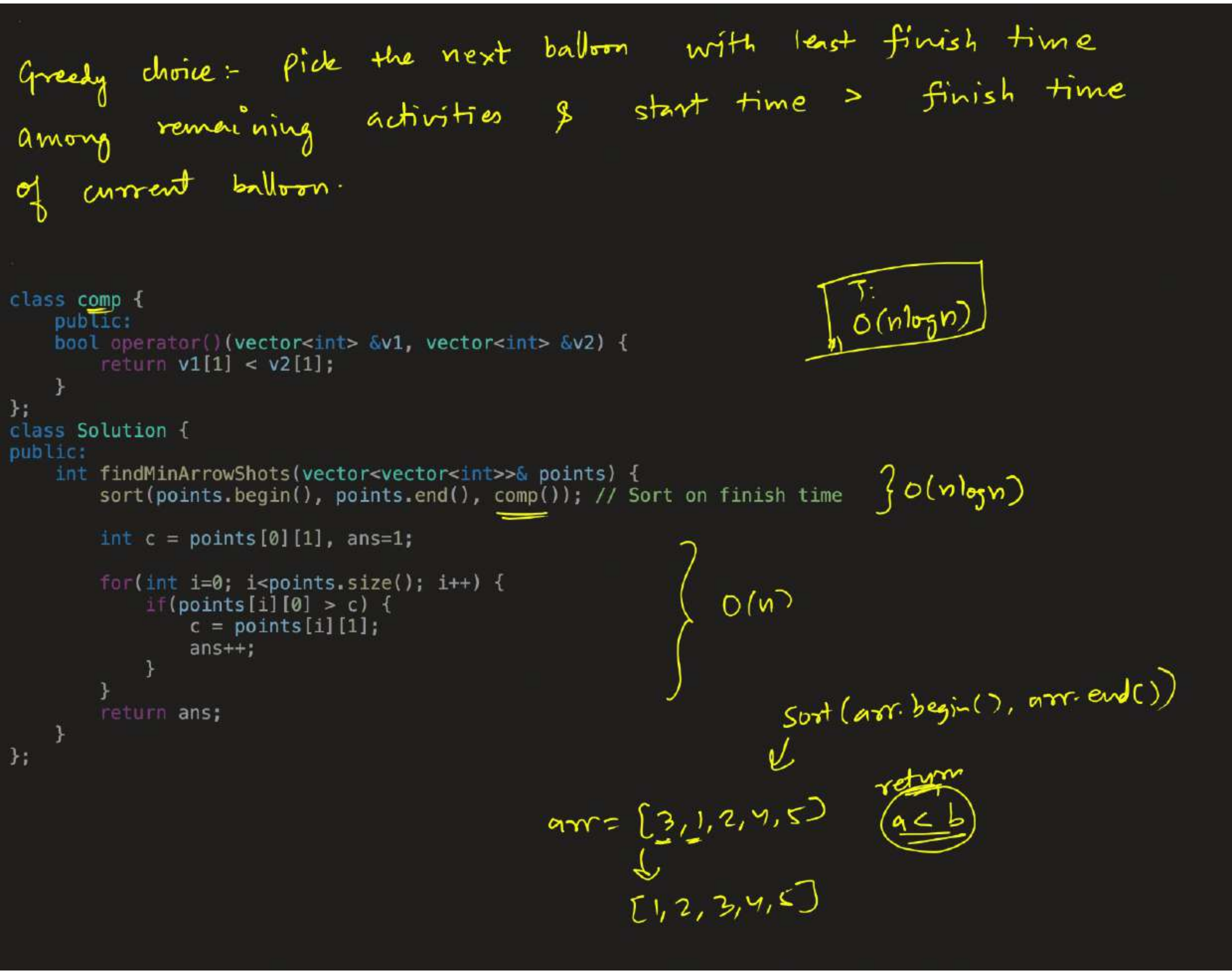
<2, 7>

<2, 6>

min arrows= 4







Greedy choice :- Pick the next balloon with least finish time among remaining activities & start time > finish time of current balloon.

```
class comp {
    public:
    bool operator()(vector<int> &v1, vector<int> &v2) {
        return v1[1] < v2[1];
    }
};
class Solution {
public:
    int findMinArrowShots(vector<vector<int>>& points) {
        sort(points.begin(), points.end(), comp()); // Sort on finish time

        int c = points[0][1], ans=1;

        for(int i=0; i<points.size(); i++) {
            if(points[i][0] > c) {
                c = points[i][1];
                ans++;
            }
        }
        return ans;
    }
};
```

T: O(nlogn)

} O(nlogn)

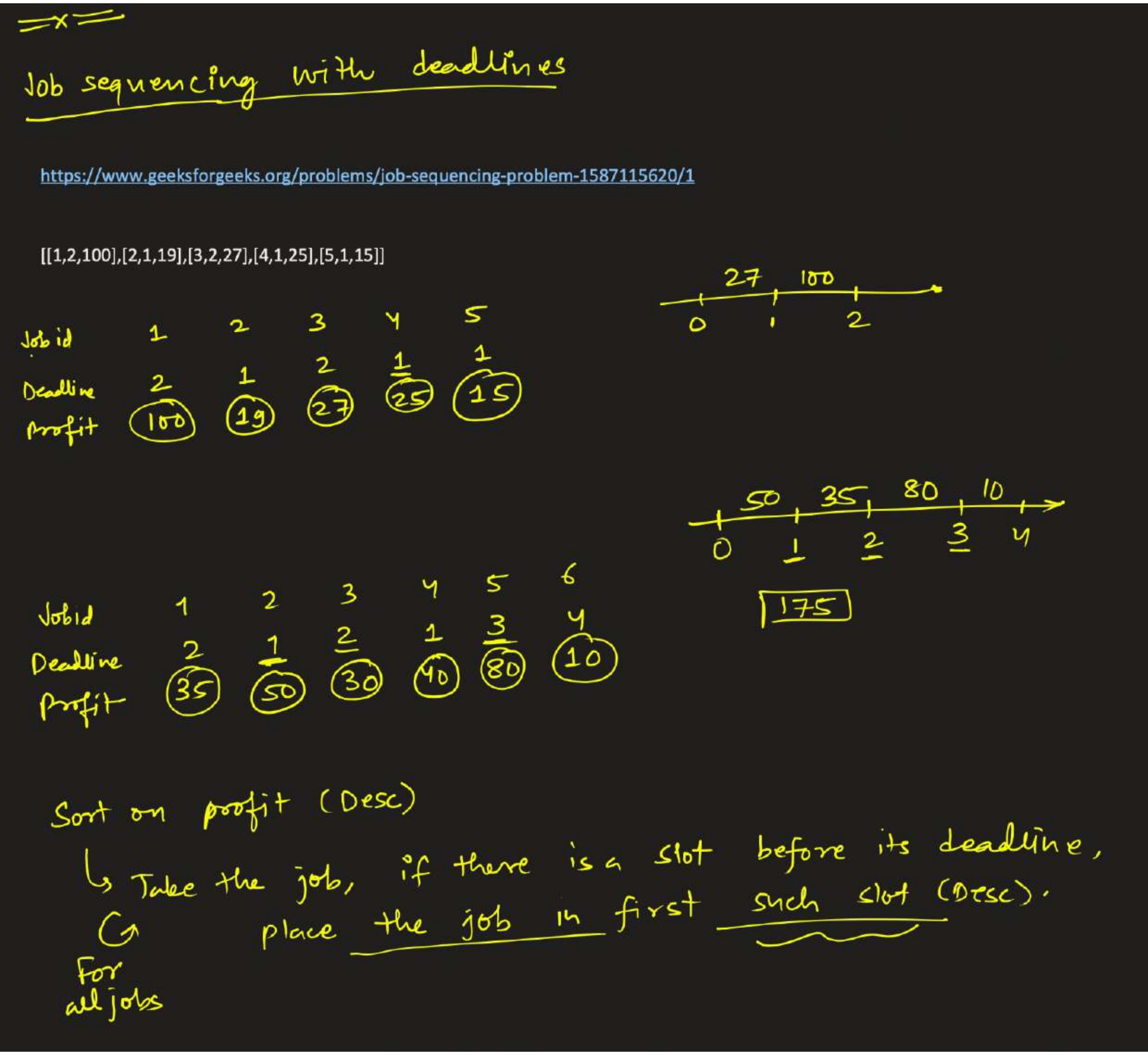
} O(n)

sort(arr.begin(), arr.end())

arr= [3,1,2,4,5]

[1,2,3,4,5]

return a < b



=x=

Job sequencing with deadlines

https://www.geeksforgeeks.org/problems/job-sequencing-problem-1587115620/1

[[1,2,100],[2,1,19],[3,2,27],[4,1,25],[5,1,15]]

Job id	1	2	3	4	5
Deadline	2	1	2	1	1
Profit	100	19	27	25	15

175

Jobid	1	2	3	4	5	6
Deadline	2	1	2	1	3	4
Profit	35	50	30	40	80	10

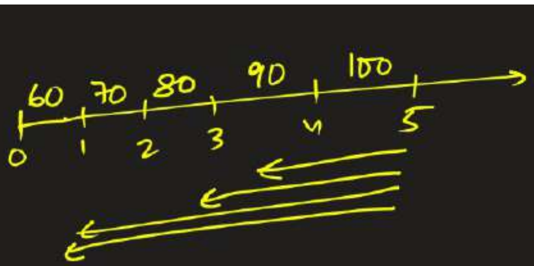
Sort on profit (Desc)

↳ Take the job, if there is a slot before its deadline,

↳ place the job in first such slot (Desc).

For all jobs

job id	1	2	3	4	5
Deadline	5	5	5	5	5
profit	100	90	80	70	60



$O(n^2)$

Jobid	1	2	3	4	5	6
Deadline	2	1	2	1	3	4
Profit	35	50	30	40	80	10

Sort on Deadline (Desc)

Jobid	6	5	1	3	2	4	
Deadline	4	3	2	2	1	1	0
profit	10	80	35	30	50	40	

$\boxed{n \log n}$

10+ 80+35+ 50

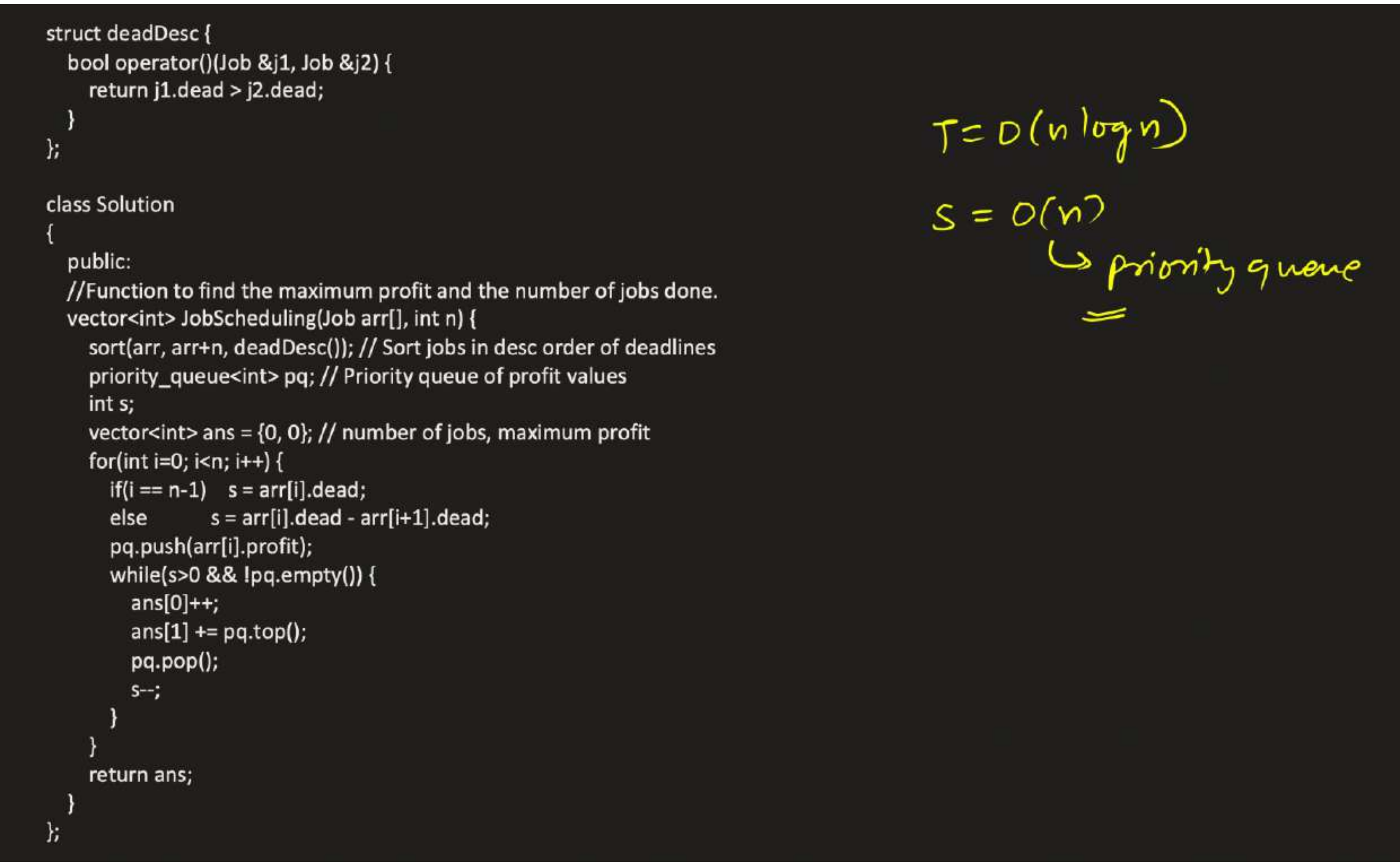
slots available = ~~2~~ ~~0~~ 1

< 2,1,50 >

~~<2,1,50>~~

3, 2, 30

< 4,1,40 >



```
struct deadDesc {
   bool operator()(Job &j1, Job &j2) {
      return j1.dead > j2.dead;
   }
};

class Solution
{
   public:
   //Function to find the maximum profit and the number of jobs done.
   vector<int> JobScheduling(Job arr[], int n) {
      sort(arr, arr+n, deadDesc()); // Sort jobs in desc order of deadlines
      priority_queue<int> pq; // Priority queue of profit values
      int s;
      vector<int> ans = {0, 0}; // number of jobs, maximum profit
      for(int i=0; i<n; i++) {
         if(i == n-1)    s = arr[i].dead;
         else          s = arr[i].dead - arr[i+1].dead;
         pq.push(arr[i].profit);
         while(s>0 && !pq.empty()) {
            ans[0]++;
            ans[1] += pq.top();
            pq.pop();
            s--;
         }
      }
      return ans;
   }
};
```

$T = O(n \log n)$

$S = O(n)$

↳ $\underline{\underline{\text{priority queue}}}$