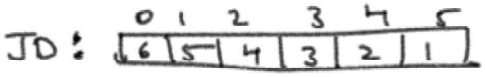
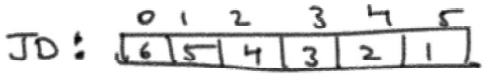
21 September 2023 11:27 AM

Job Scheduling



days = 2 → number of cuts = ? = 1

⇒ diff = 6 diff = 1 ⇒ ans = 7

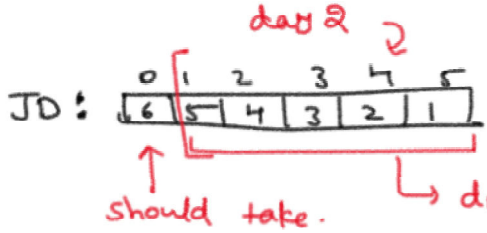


days = 3 → cuts = 2

day 1	day 2	day 3	Total.
6	5	4 3 2 1	15
6	5 4	3 2 1	14
6	5 4 3	2 1	13
6	5 4 3 2	1	12
6 5	4	3 2 1	13
6 5	4 3	2 1	12
6 5	4 3 2	1	11
6 5 4	3	2 1	11
6 5 4	3 2	1	10

return

⑥ 5 4 3 ② ①



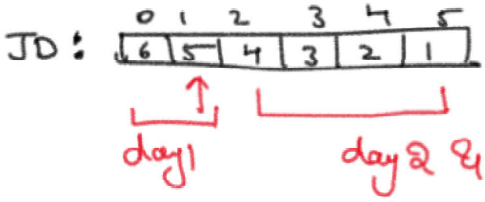
day 1 day 2 day 3.

6

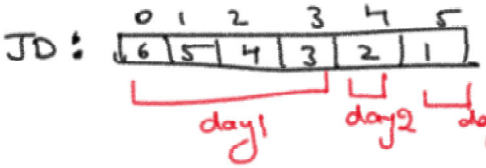
Starting from day 0.

daywise Max

Next Iteration :



to



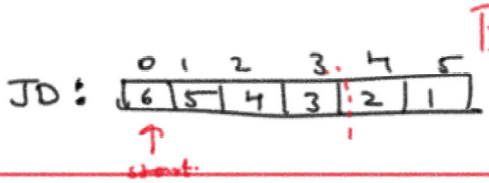

```
for (i = 0; i ≤ n-d ; i++)
{     // process
}
```

track max of that day also

```
daywiseMAX[day] = max( )
```

---

process :




```
function solve ( JD, days, curr day ) {
                       left   start

    for (int start ; start ≤ n-d , start++ ) {
        daywiseMAX[currday] = max ( same, JD[curr])
```

Tracks max.

```
        daywiseMAX[currday] = max( same, JD[curr] )
        // assign for next day.
        solve( JD, dayleft-1, curr_day+1, start+1 );
        // populates next day.
        sum = sum.(daywiseMAX)
        ans = min(sum, ans)
    }
    return ans
}
```

```
f( days_left, curr_day, start_index )

for(int start ; start ≤ n-d , start++ ) {
    daywiseMAX[currday] = max( same, JD[curr] )
    // assign for next day.
    solve( JD, dayleft-1, curr_day+1, start+1 );
    // populates next day.
    sum = sum.(daywiseMAX)
    ans = min(sum, ans)
}
```

days = 3

JD:

0	1	2	3	4	5
6	5	4	3	2	1

f(3,1,0)

0	1	2	3
M	~~X~~	M	M

6 5

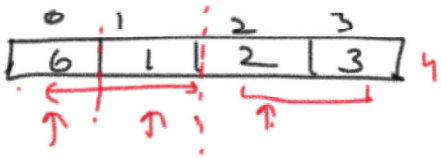
~~Start~~ = 0 :

f(2, 2, 1)

0	1	2	3
6	1	2	3

days = 3

day 0	day 1	day 2	Total (min)
(6)	(1)	2 (3)	10 ⟶ min
(6)	1 (2)	(3)	11
(6) 1	(2)	(3)	11



days = 3

f(3, 0, 0)
days left, curr day, pos

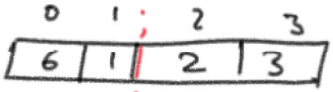
f(2, 1, 1)

f(1, 2, 2) — all

f(1, 2, 3)

```
for(int start ; start ≤ n-d , start++ ) {
    daywiseMAX[currday] = max ( same, JD[curr])
    // assign for next day.
    solve ( JD, dayleft -1 , curr day+1 , start+1 );
    // populates next day.
    Sum = Sum.(daywiseMAX)
    ans = min(sum, ans)
}
```

NEW ALGO :



start n = 4

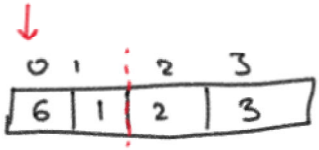
days = 3

① start ≤ n-d. [can be inside for loop]

② if(d == 1) { right max } return.

Maintain a 2D array: dp[ind][day]

return dp[0][3]



d = 3
n = 4

```
for( ; start ≤ n-d ; start++) {
    mx = max(mx, v[i]) ;
    // then can or cannot break.
```

dp[start][day] = min(dp[start][day], mx + solve(start+1, day-1))