

## Q1

### Explaining The Algorithm

I got two list(jobProcessingTimeList and jobWeightList) and return one list(jobOrderList). Indices of the jobProcessingTimeList and jobWeightList lists are equal the job numbers. For minimizing the weighted sum of the completion times, I must choose a job firstly if its weight is more and time is less. So, I must choose firstly the job which has biggest  $w_i/t_i$  value.

So I sort the list by comparing the  $w_i/t_i$  values of the jobs. And I return the ordered job list.

An example:

Handwritten example showing the calculation of  $w_i/t_i$  values for 7 jobs and the resulting job order list.

Jobs and their  $w_i/t_i$  values:

- $J_0 \Rightarrow t_0=8, w_0=24 \Rightarrow w_0/t_0=3$
- $J_1 \Rightarrow t_1=10, w_1=40 \Rightarrow w_1/t_1=4$
- $J_2 \Rightarrow t_2=4, w_2=8 \Rightarrow w_2/t_2=2$
- $J_3 \Rightarrow t_3=6, w_3=30 \Rightarrow w_3/t_3=5$
- $J_4 \Rightarrow t_4=2, w_4=16 \Rightarrow w_4/t_4=8$
- $J_5 \Rightarrow t_5=12, w_5=24 \Rightarrow w_5/t_5=2$
- $J_6 \Rightarrow t_6=8, w_6=8 \Rightarrow w_6/t_6=1$

Tables for  $t_i$  and  $w_i$ :

$t_i$	$t_0$	$t_1$	$t_2$	$t_3$	$t_4$	$t_5$	$t_6$
$t_i$	8	10	4	6	2	12	8

$w_i$	$w_0$	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$	$w_6$
$w_i$	24	40	8	30	16	24	8

Initial  $w_i/t_i$  values:

$w_i/t_i$	$3(i=0)$	$4(i=1)$	$2(i=2)$	$5(i=3)$	$8(i=4)$	$2(i=5)$	$1(i=6)$
$w_i/t_i$	3	4	2	5	8	2	1

Sort non-increasingly (by comparing the  $w_i/t_i$  values):

$w_i/t_i$	$8(i=4)$	$5(i=3)$	$4(i=1)$	$3(i=0)$	$2(i=2)$	$2(i=5)$	$1(i=6)$
$w_i/t_i$	8	5	4	3	2	2	1

Job order list:

Job order	4	3	1	0	2	5	6
Job order	$J_4$	$J_3$	$J_1$	$J_0$	$J_2$	$J_5$	$J_6$

### Time Complexity Analyze

I use 2 for loops and one insertion sort. for loops are takes  $O(n)$  time and insertion sort takes  $O(n^2)$  time. So time complexity is  $O(n+n^2) \Rightarrow O(n^2)$ .

## Q2

a)

Let  $n=4$

$M=10$

$N=[1,1,3,1]$

$S=[12,15,1,14]$

This algorithm returns [NY,NY,SF,NY] with total cost= $1+1+1+1+10+10=24$  (it doesn't look the M value)

But optimal solution is [NY,NY,NY,NY] with total cost= $1+1+3+1=6$

b)

### Explaining The Algorithm

Our optimal plan finishes either NY or SF. If it finishes with NY, there is an exact cost NY for size  $n$  and we add to this cost  $NY[n-1]$  or  $SF[n-1]+M$  (choose the minimum one). If it finishes with SF, there is an exact cost SF for size  $n$  and we add to this cost  $SF[n-1]$  or  $NY[n-1]+M$  (choose the minimum one).

So in the for loop, dynamic programming table is filled and we use its values for calculating the early costs.

Lastly, after the for loop there are two lists and at the end of the lists (exact monthCounts size) one is store the cost of the optimal plan ended with NY the other one is store the cost of the optimal plan ended with SF we choose the minimum one and return it.

### Time Complexity Analyze

There is one for loop with size  $n$ . The other processes takes constant time. So time complexity is  $O(n)$ .