CS325

Guangyu Zhang

1.

For each job j_i, it has deadline d_i and penalty p_i. Since we want the minimum of penalty, we can sort jobs by its penalty from high to low. Then, processing jobs on decreasing order. And since every jobs takes 1 minute, if we assign a job, the t decrease by 1. Next, if we find the lastest d_i between 1 and t, we can assign j_i with the max penalty value, and set the flag from true to false which means the current job is done. last, the final penalty is the total penalty minus the sum of penalty that we already assigned. t is the total time we have, and f is penalty while finished.

```
Arr = [[j1, d1, p1, flag1], [j2, d2, p2, flag2], … ,[ji, di, pi, flagi]]

// flag is a Boolean value to show the job is done or not

Mergesort(arr[i][2])                //here we assume the mergesort can sort it from high to low

For i in range(len(arr))

    For j in range(min(t, arr[i][1]), 1)

        If arr[j][3]=true

            p += max(arr[j][2])     // get the p that already finished

            arr[j][3]=false

f = sum(arr[i][2]) – p             // total penalty decrease the job without penalty
```

the sort part has O(nlgn), and the scheduling part has O(n^2), so the total complexity is O(n^2)

2.

For {a1, a2, …, an}, the scheduling algorithm only takes start time and end time as its input. In other words, the orders of scheduling does not impact the final result of the scheduling, the total algorithm will return the same results as first to finish algorithm. Therefore, since the first to finish is a greedy algorithm, if we pick the last as start, it's a reverse version of the first to finish, the last to start is greedy, too.

When we select the last activity to schedule, it will be compared to all other activities, so it will be the best choice in every interval, and since if some activities have time conflict in interval, the last to start can avoid it because if we pick one as last, the activities have conflict with it will not be considered in the next pick. Therefore, it is an optimal algorithm.

3.

The first step of last-to-start activity scheduling algorithm is sorting the activity starting time in decreasing order. Here, I use sorted function in Python, which has the O(nlgn) complexity (https://wiki.python.org/moin/TimeComplexity). Then, using the time information to determine a maximum set of activities that do not conflict with each other. Here, choose the latest one as the first activity, and use it start time to determine the next activity. Next, use L2S[] to store the activity we pick.

```
Arr=[[NO., start, end], … ]

sort(arr[i][1])

L2S=[]
```

```
j=1

for j in range(len(arr)):

      if arr[i][1] >= arr[j][2]:

        L2S.append(arr[j])

        i = j
```

since I use the sort function in Python which complexity is O(nlgn), and other codes have O(n), the total complexity is O(nlgn)