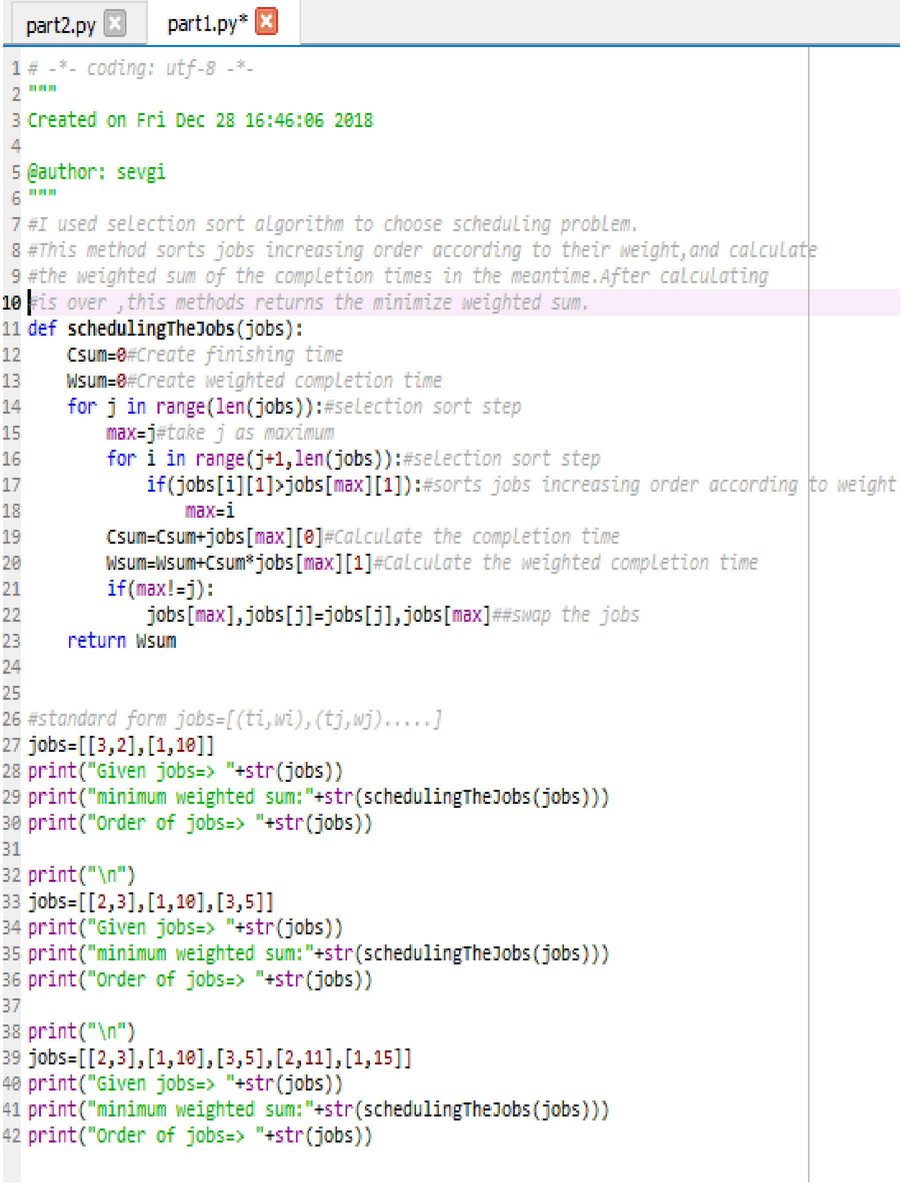
CSE 321 Homework 5

SEVGİ BAYANSALDUZ – 151044076

- ## PART1



```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 28 16:46:06 2018

@author: sevgi
"""
#I used selection sort algorithm to choose scheduling problem.
#This method sorts jobs increasing order according to their weight,and calculate
#the weighted sum of the completion times in the meantime.After calculating
#is over ,this methods returns the minimize weighted sum.
def schedulingTheJobs(jobs):
    Csum=0#Create finishing time
    Wsum=0#Create weighted completion time
    for j in range(len(jobs)):#selection sort step
        max=j#take j as maximum
        for i in range(j+1,len(jobs)):#selection sort step
            if(jobs[i][1]>jobs[max][1]):#sorts jobs increasing order according to weight
                max=i
        Csum=Csum+jobs[max][0]#Calculate the completion time
        Wsum=Wsum+Csum*jobs[max][1]#Calculate the weighted completion time
        if(max!=j):
            jobs[max],jobs[j]=jobs[j],jobs[max]##swap the jobs
    return Wsum


#standard form jobs=[(ti,wi),(tj,wj).....]
jobs=[[3,2],[1,10]]
print("Given jobs=> "+str(jobs))
print("minimum weighted sum:"+str(schedulingTheJobs(jobs)))
print("Order of jobs=> "+str(jobs))

print("\n")
jobs=[[2,3],[1,10],[3,5]]
print("Given jobs=> "+str(jobs))
print("minimum weighted sum:"+str(schedulingTheJobs(jobs)))
print("Order of jobs=> "+str(jobs))

print("\n")
jobs=[[2,3],[1,10],[3,5],[2,11],[1,15]]
print("Given jobs=> "+str(jobs))
print("minimum weighted sum:"+str(schedulingTheJobs(jobs)))
print("Order of jobs=> "+str(jobs))
```

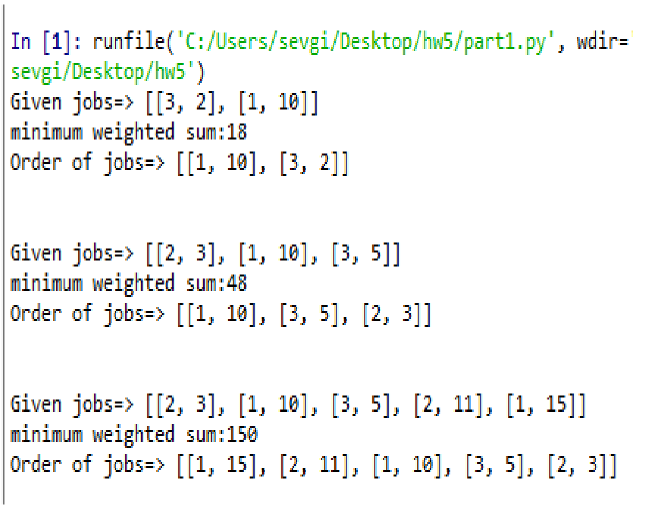
I used selection sort algorithm to choose scheduling pr increasing order according to their weight and calculat times in the meantime. After calculating is over, this m sum.

COMPLEXİTY:

It has the same time complexity as the selection sortin

$$\sum_{j=0}^{n} \sum_{i=i+1}^{n} 1 = \theta(n^2)$$

OUTPUT:



```
In [1]: runfile('C:/Users/sevgi/Desktop/hw5/part1.py', wdir=
sevgi/Desktop/hw5')
Given jobs=> [[3, 2], [1, 10]]
minimum weighted sum:18
Order of jobs=> [[1, 10], [3, 2]]


Given jobs=> [[2, 3], [1, 10], [3, 5]]
minimum weighted sum:48
Order of jobs=> [[1, 10], [3, 5], [2, 3]]


Given jobs=> [[2, 3], [1, 10], [3, 5], [2, 11], [1, 15]]
minimum weighted sum:150
Order of jobs=> [[1, 15], [2, 11], [1, 10], [3, 5], [2, 3]]
```

- ## PART2

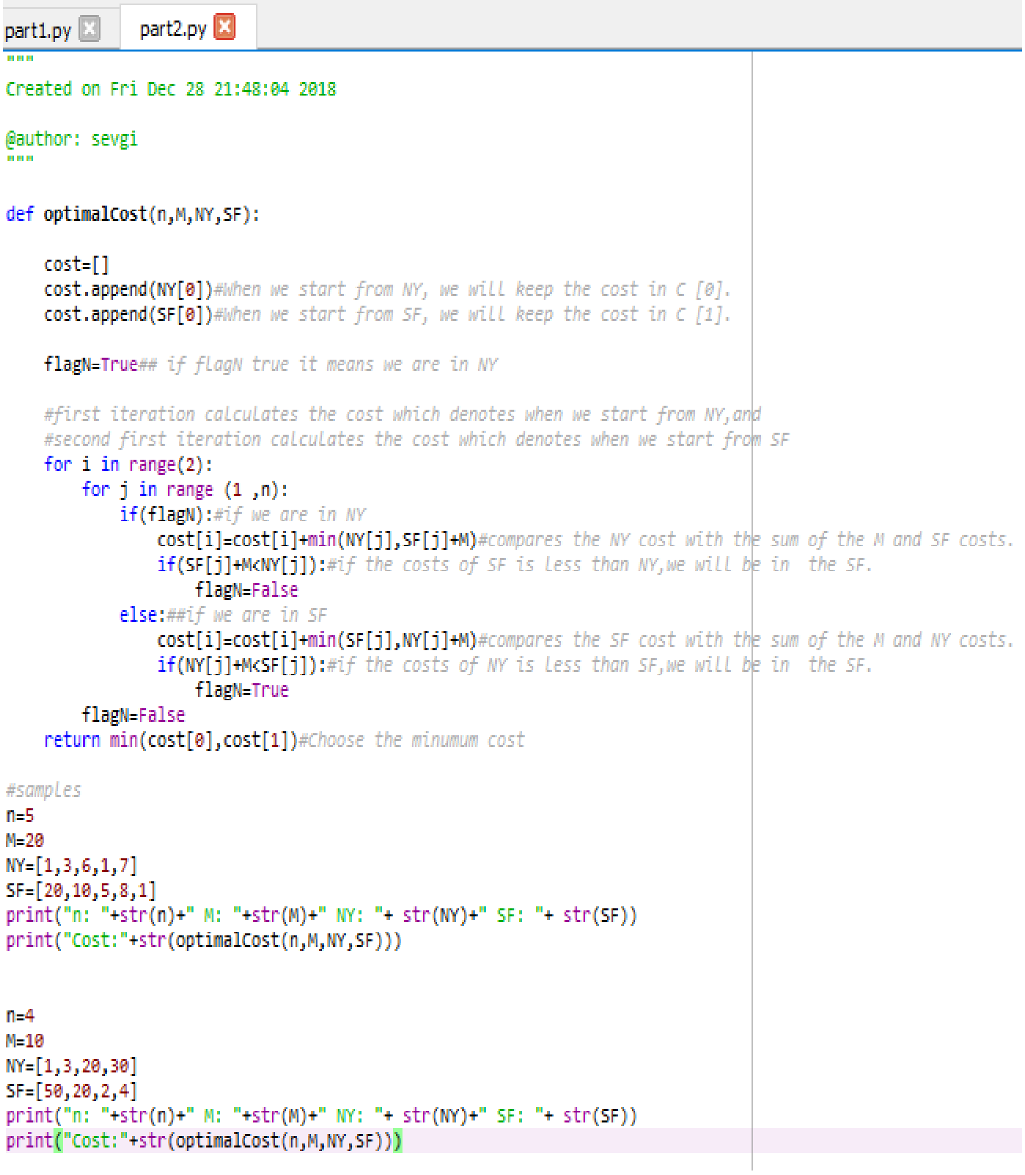
a)

N=5,M=20

City	Month1	Month2	Month3	Month4	Month5
NY	1	3	6	1	7
SF	20	10	5	8	1

According the given algorithm cost will be 71. Plan=[NY,NY,SF,NY,SF]
But optimal plan is =[NY,NY, NY,NY, NY] and optimal cost is 18.

b)



```
"""
Created on Fri Dec 28 21:48:04 2018

@author: sevgi
"""

def optimalCost(n,M,NY,SF):

    cost=[]
    cost.append(NY[0])#When we start from NY, we will keep the cost in C [0].
    cost.append(SF[0])#When we start from SF, we will keep the cost in C [1].

    flagN=True## if flagN true it means we are in NY

    #first iteration calculates the cost which denotes when we start from NY,and
    #second first iteration calculates the cost which denotes when we start from SF
    for i in range(2):
        for j in range (1 ,n):
            if(flagN):#if we are in NY
                cost[i]=cost[i]+min(NY[j],SF[j]+M)#compares the NY cost with the sum of the M and SF costs.
                if(SF[j]+M<NY[j]):#if the costs of SF is less than NY,we will be in  the SF.
                    flagN=False
            else:##if we are in SF
                cost[i]=cost[i]+min(SF[j],NY[j]+M)#compares the SF cost with the sum of the M and NY costs.
                if(NY[j]+M<SF[j]):#if the costs of NY is less than SF,we will be in  the SF.
                    flagN=True
        flagN=False
    return min(cost[0],cost[1])#Choose the minumum cost

#samples
n=5
M=20
NY=[1,3,6,1,7]
SF=[20,10,5,8,1]
print("n: "+str(n)+" M: "+str(M)+" NY: "+ str(NY)+" SF: "+ str(SF))
print("Cost:"+str(optimalCost(n,M,NY,SF)))


n=4
M=10
NY=[1,3,20,30]
SF=[50,20,2,4]
print("n: "+str(n)+" M: "+str(M)+" NY: "+ str(NY)+" SF: "+ str(SF))
print("Cost:"+str(optimalCost(n,M,NY,SF)))
```

First iteration of outer loop we assume that I start from NY and I choose the next city according to comparing the cost of NY[j] with the sum of the M and SF[j] `s cost. If NY[j]`s cost is less than SF[j]+M, I will stay in the NY and increase the cost[0] with NY[j] and I will choose the next city according to this rule. Otherwise I will go to the SF and increase the cost[0] with SF[j]+M and I will choose the next city according to this rule: min(SF[j],NY[j]+M). These comparisons will continue until j equals n.

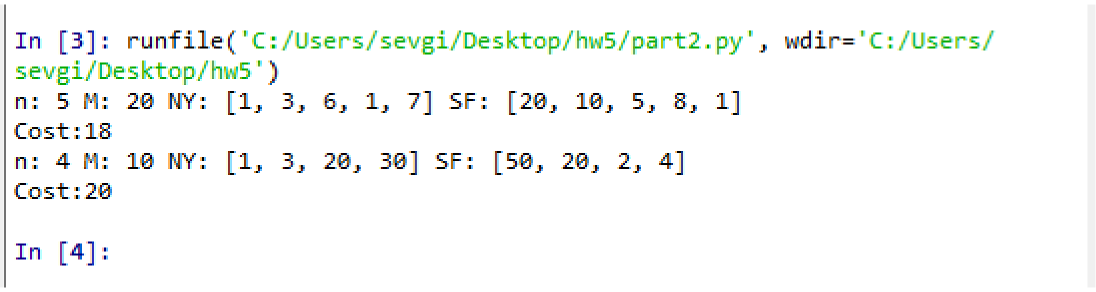
In the iteration of outer loop we assume that I start from SF and I choose the next city according to comparing the cost of SF[j] with the sum of the M and NY[j] `s cost. If SF[j]`s cost is less than NY[j]+M, I will stay in the SF and increase the cost[1] with SF[j] and I will choose the next city according to this rule. Otherwise I will go to the NY and increase the cost[1] with NY[j]+M and I will choose the next city according to this rule: min(NY[j],SF[j]+M).These comparisons will continue until j equals n.

After the outer loop is end,the minimum cost will be chosen between cost[0] and cost[1].

Complexity:

$$\sum_{j=0}^{2} \sum_{i=1}^{n} n = \theta(2n) = \theta(n)$$

Outputs:



```
In [3]: runfile('C:/Users/sevgi/Desktop/hw5/part2.py', wdir='C:/Users/
sevgi/Desktop/hw5')
n: 5 M: 20 NY: [1, 3, 6, 1, 7] SF: [20, 10, 5, 8, 1]
Cost:18
n: 4 M: 10 NY: [1, 3, 20, 30] SF: [50, 20, 2, 4]
Cost:20

In [4]:
```