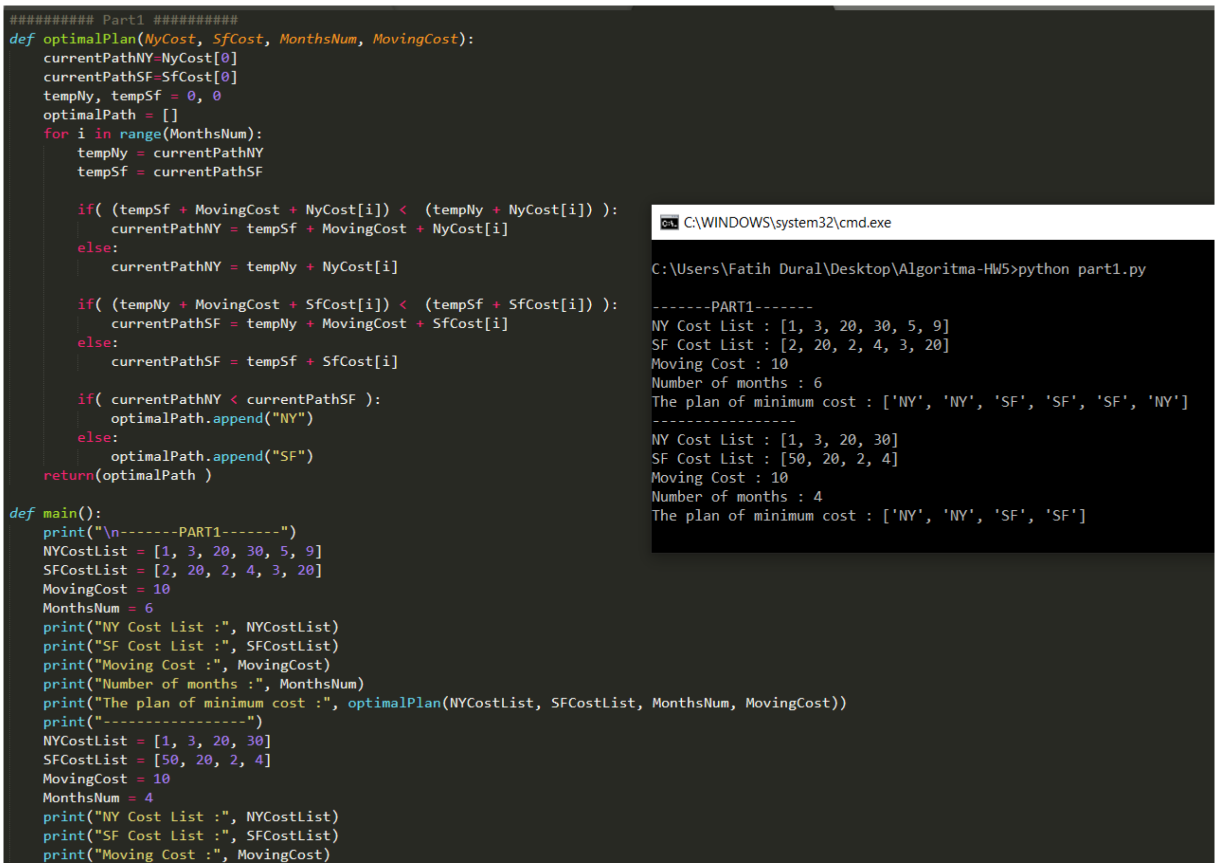
CSE 321 Homework 5

Fatih Dural

151044041

Report

Part1



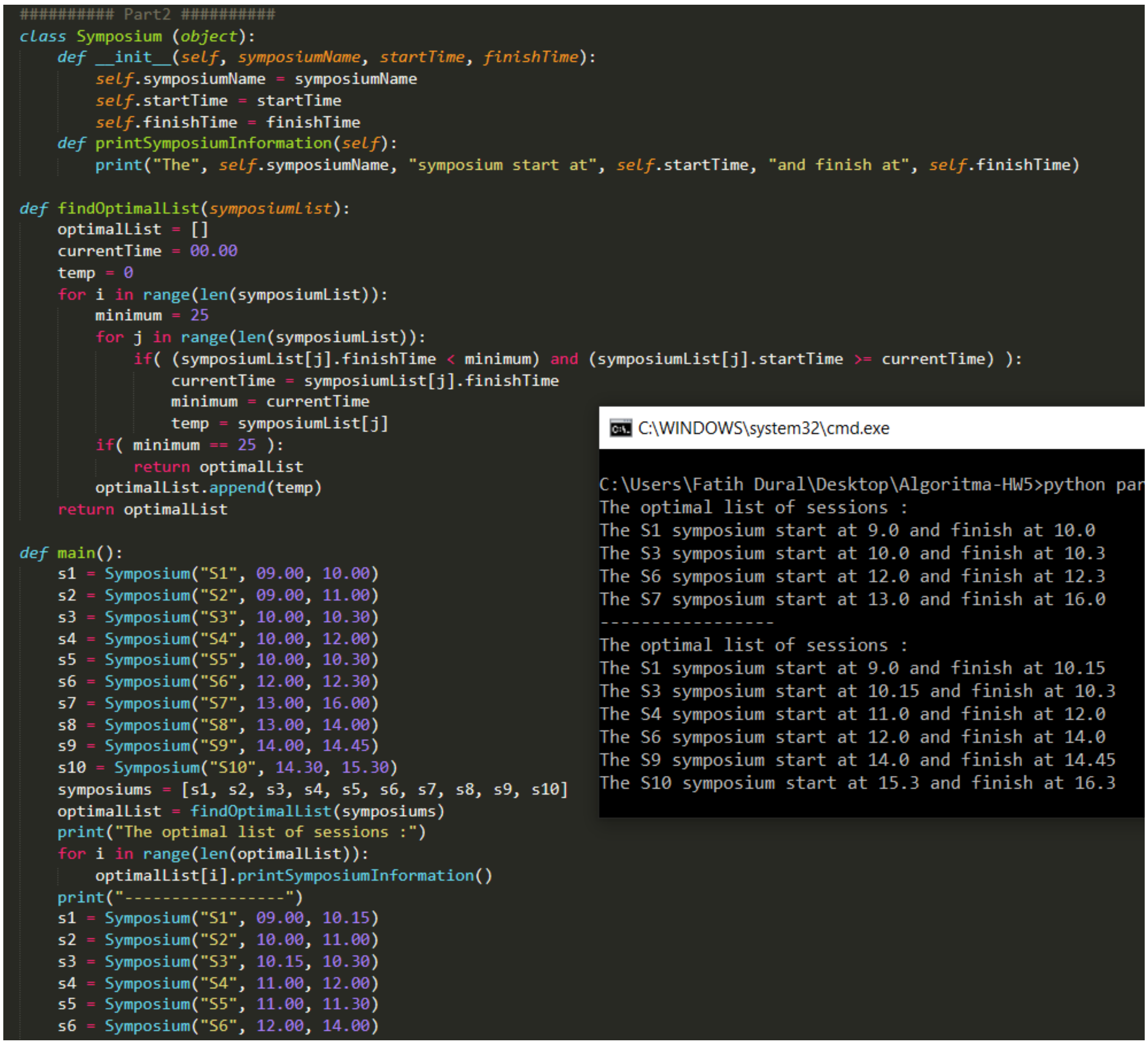
```
########## Part1 ##########
def optimalPlan(NyCost, SfCost, MonthsNum, MovingCost):
    currentPathNY=NyCost[0]
    currentPathSF=SfCost[0]
    tempNy, tempSf = 0, 0
    optimalPath = []
    for i in range(MonthsNum):
        tempNy = currentPathNY
        tempSf = currentPathSF

        if( (tempSf + MovingCost + NyCost[i]) <  (tempNy + NyCost[i]) ):
            currentPathNY = tempSf + MovingCost + NyCost[i]
        else:
            currentPathNY = tempNy + NyCost[i]

        if( (tempNy + MovingCost + SfCost[i]) <  (tempSf + SfCost[i]) ):
            currentPathSF = tempNy + MovingCost + SfCost[i]
        else:
            currentPathSF = tempSf + SfCost[i]

        if( currentPathNY < currentPathSF ):
            optimalPath.append("NY")
        else:
            optimalPath.append("SF")
    return(optimalPath )

def main():
    print("\n-------PART1-------")
    NYCostList = [1, 3, 20, 30, 5, 9]
    SFCostList = [2, 20, 2, 4, 3, 20]
    MovingCost = 10
    MonthsNum = 6
    print("NY Cost List :", NYCostList)
    print("SF Cost List :", SFCostList)
    print("Moving Cost :", MovingCost)
    print("Number of months :", MonthsNum)
    print("The plan of minimum cost :", optimalPlan(NYCostList, SFCostList, MonthsNum, MovingCost))
    print("----------------")
    NYCostList = [1, 3, 20, 30]
    SFCostList = [50, 20, 2, 4]
    MovingCost = 10
    MonthsNum = 4
    print("NY Cost List :", NYCostList)
    print("SF Cost List :", SFCostList)
    print("Moving Cost :", MovingCost)
```

C:\WINDOWS\system32\cmd.exe

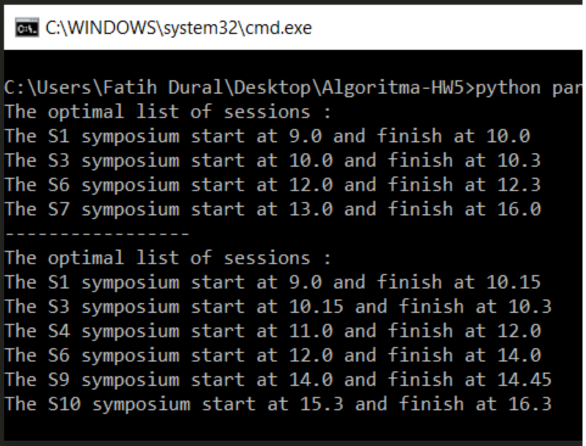
```
C:\Users\Fatih Dural\Desktop\Algoritma-HW5>python part1.py

-------PART1-------
NY Cost List : [1, 3, 20, 30, 5, 9]
SF Cost List : [2, 20, 2, 4, 3, 20]
Moving Cost : 10
Number of months : 6
The plan of minimum cost : ['NY', 'NY', 'SF', 'SF', 'SF', 'NY']
----------------
NY Cost List : [1, 3, 20, 30]
SF Cost List : [50, 20, 2, 4]
Moving Cost : 10
Number of months : 4
The plan of minimum cost : ['NY', 'NY', 'SF', 'SF']
```

This section is looking for the most appropriate way. Monthly fees, displacement fees and number of months are determined. We will find an optimal plan where we can carry out the works at the most reasonable price requested from us. This algorithm will be established as a dynamic programming algorithm. The approach that records the results obtained in the previous steps and uses these records to obtain new results is called dynamic programming. In the program, the information for the two cities is given to the function. The function examines possible situations for the current month. It examines the previous paths and the current path, makes it more logical whether the displacement remains or remains the most appropriate. Returns OptimalPath. The time complexity of the algorithm is n time and the worst-case is the same.

Part2



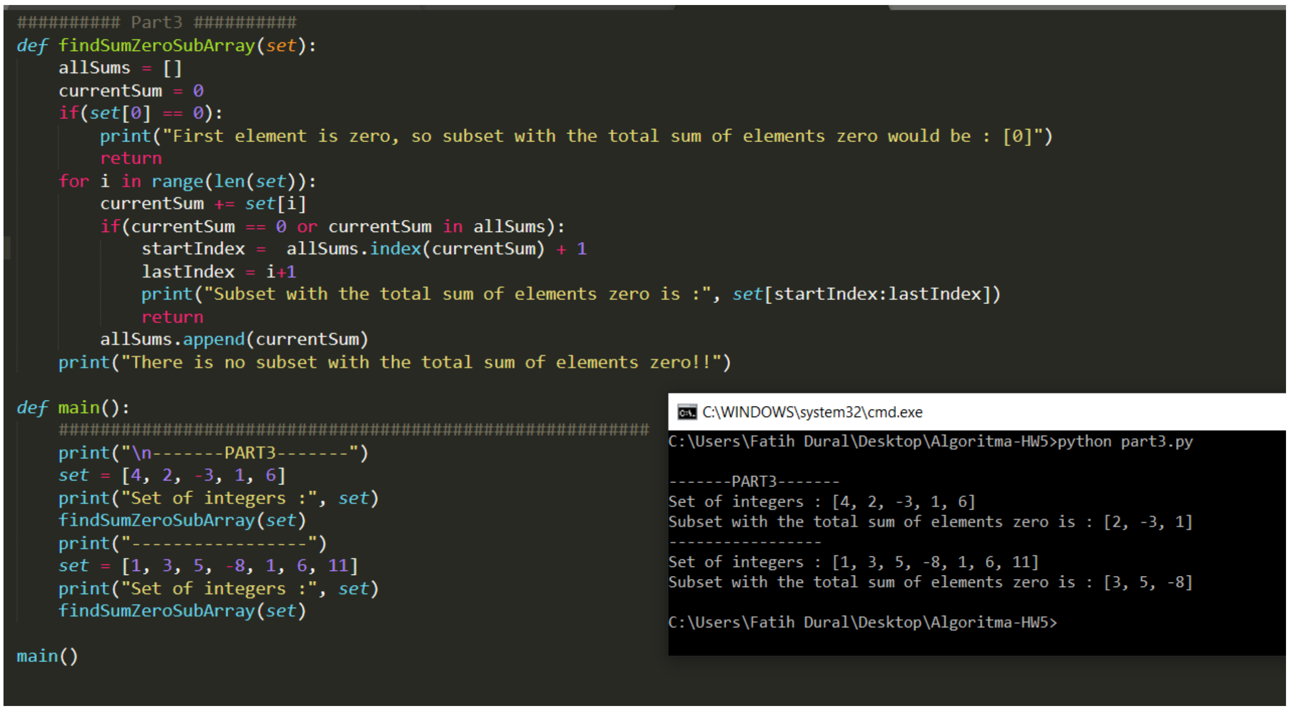
```
########## Part2 ##########
class Symposium (object):
    def __init__(self, symposiumName, startTime, finishTime):
        self.symposiumName = symposiumName
        self.startTime = startTime
        self.finishTime = finishTime
    def printSymposiumInformation(self):
        print("The", self.symposiumName, "symposium start at", self.startTime, "and finish at", self.finishTime)

def findOptimalList(symposiumList):
    optimalList = []
    currentTime = 00.00
    temp = 0
    for i in range(len(symposiumList)):
        minimum = 25
        for j in range(len(symposiumList)):
            if( (symposiumList[j].finishTime < minimum) and (symposiumList[j].startTime >= currentTime) ):
                currentTime = symposiumList[j].finishTime
                minimum = currentTime
                temp = symposiumList[j]
        if( minimum == 25 ):
            return optimalList
        optimalList.append(temp)
    return optimalList

def main():
    s1 = Symposium("S1", 09.00, 10.00)
    s2 = Symposium("S2", 09.00, 11.00)
    s3 = Symposium("S3", 10.00, 10.30)
    s4 = Symposium("S4", 10.00, 12.00)
    s5 = Symposium("S5", 10.00, 10.30)
    s6 = Symposium("S6", 12.00, 12.30)
    s7 = Symposium("S7", 13.00, 16.00)
    s8 = Symposium("S8", 13.00, 14.00)
    s9 = Symposium("S9", 14.00, 14.45)
    s10 = Symposium("S10", 14.30, 15.30)
    symposiums = [s1, s2, s3, s4, s5, s6, s7, s8, s9, s10]
    optimalList = findOptimalList(symposiums)
    print("The optimal list of sessions :")
    for i in range(len(optimalList)):
        optimalList[i].printSymposiumInformation()
    print("-----------------")
    s1 = Symposium("S1", 09.00, 10.15)
    s2 = Symposium("S2", 10.00, 11.00)
    s3 = Symposium("S3", 10.15, 10.30)
    s4 = Symposium("S4", 11.00, 12.00)
    s5 = Symposium("S5", 11.00, 11.30)
    s6 = Symposium("S6", 12.00, 14.00)
```



```
C:\Users\Fatih Dural\Desktop\Algoritma-HW5>python par
The optimal list of sessions :
The S1 symposium start at 9.0 and finish at 10.0
The S3 symposium start at 10.0 and finish at 10.3
The S6 symposium start at 12.0 and finish at 12.3
The S7 symposium start at 13.0 and finish at 16.0
-----------------
The optimal list of sessions :
The S1 symposium start at 9.0 and finish at 10.15
The S3 symposium start at 10.15 and finish at 10.3
The S4 symposium start at 11.0 and finish at 12.0
The S6 symposium start at 12.0 and finish at 14.0
The S9 symposium start at 14.0 and finish at 14.45
The S10 symposium start at 15.3 and finish at 16.3
```

In Part 2, there are many symposiums with different start and end times. Symposiums should be entered on time and stopped until the end time. We are asked for the Greedy Algorithm that will allow us to participate in the symposium in the greatest amount. Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So the problems where choosing locally optimal also leads to global solution are best fit for Greedy. A class named Symposium was created. This class holds the name, start and end time of the symposium. The findOptimalList function finds the most appropriate solution in the list of symposia given to it as a parameter. Timing was performed over 24 hours and represented by float numbers. The function finds the symposiums that are not delayed and have the lowest end time according to the current time. One cycle rotates by the number of elements, the other cycle finds the optimal solution each time. The worst timing would also be the time complexity n ^ 2.

Part3



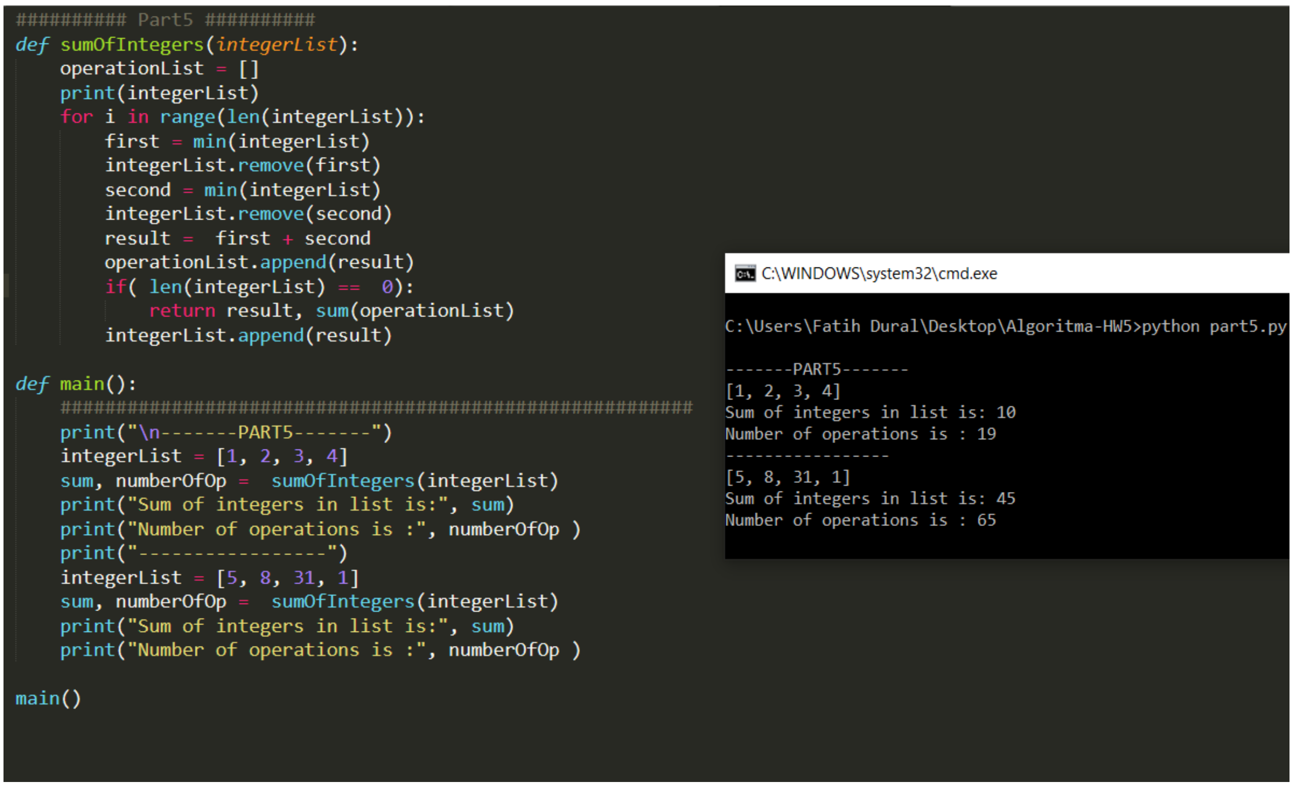
```python
########## Part3 ##########
def findSumZeroSubArray(set):
    allSums = []
    currentSum = 0
    if(set[0] == 0):
        print("First element is zero, so subset with the total sum of elements zero would be : [0]")
        return
    for i in range(len(set)):
        currentSum += set[i]
        if(currentSum == 0 or currentSum in allSums):
            startIndex =  allSums.index(currentSum) + 1
            lastIndex = i+1
            print("Subset with the total sum of elements zero is :", set[startIndex:lastIndex])
            return
        allSums.append(currentSum)
    print("There is no subset with the total sum of elements zero!!")

def main():
    ##############################################################
    print("\n-------PART3-------")
    set = [4, 2, -3, 1, 6]
    print("Set of integers :", set)
    findSumZeroSubArray(set)
    print("----------------")
    set = [1, 3, 5, -8, 1, 6, 11]
    print("Set of integers :", set)
    findSumZeroSubArray(set)

main()
```

C:\WINDOWS\system32\cmd.exe

```
C:\Users\Fatih Dural\Desktop\Algoritma-HW5>python part3.py

-------PART3-------
Set of integers : [4, 2, -3, 1, 6]
Subset with the total sum of elements zero is : [2, -3, 1]
----------------
Set of integers : [1, 3, 5, -8, 1, 6, 11]
Subset with the total sum of elements zero is : [3, 5, -8]

C:\Users\Fatih Dural\Desktop\Algoritma-HW5>
```

In this section, there is a set of integers. A subset of elements with zero sum is requested. When the subset is found, the program must terminate and print the subset. And the algorithm must be the dynamic programming algorithm. The function takes the integer list as a parameter. There is a loop of the number of elements. The elements in the loop are incrementally collected. Each total is kept in a list named allSum. If the current total is zero or if the current total has already occurred, there is the desired result. If the current sum is the logic of occurrence before, if the two results are the same, then there is a result with the sum of increments or decreases equal to zero. When the condition is satisfied, the current state is the last element of the subset and the first element in that allSum list. Once the start and end indexes have been specified, the function returns this result. The complexity of time is the number of elements of the set in each case up and n time.

Part5



```
########## Part5 ##########
def sumOfIntegers(integerList):
    operationList = []
    print(integerList)
    for i in range(len(integerList)):
        first = min(integerList)
        integerList.remove(first)
        second = min(integerList)
        integerList.remove(second)
        result =  first + second
        operationList.append(result)
        if( len(integerList) ==  0):
            return result, sum(operationList)
        integerList.append(result)

def main():
    ##########################################################
    print("\n-------PART5-------")
    integerList = [1, 2, 3, 4]
    sum, numberOfOp =  sumOfIntegers(integerList)
    print("Sum of integers in list is:", sum)
    print("Number of operations is :", numberOfOp )
    print("-----------------")
    integerList = [5, 8, 31, 1]
    sum, numberOfOp =  sumOfIntegers(integerList)
    print("Sum of integers in list is:", sum)
    print("Number of operations is :", numberOfOp )

main()
```

C:\WINDOWS\system32\cmd.exe

```
C:\Users\Fatih Dural\Desktop\Algoritma-HW5>python part5.py

-------PART5-------
[1, 2, 3, 4]
Sum of integers in list is: 10
Number of operations is : 19
-----------------
[5, 8, 31, 1]
Sum of integers in list is: 45
Number of operations is : 65
```

At the last part, there's an old computer system. You have an old computer system where you need to do A + B to add A + B numbers. For example, 13 operations are done for 5 + 8. We are asked to design a greedy algorithm to calculate the sum of the sequence with the minimum number of operations. Integer lists are created and given to the sumOfInteger function as parameters. The most appropriate greedy algorithm in this regard would be to collect the smallest ones. The loop rotates by the number of elements and the smallest two elements are taken. These two elements are added to a result list. In addition, the result must be added to the main list again and these two elements must be deleted from the main list. Operations continue until the last element. The last total is the sum of the number of elements and the result list holds the number of operations. The sum of the elements of the result list is the number of all operations. This information is returned from the function and printed on the screen. The time complexity of the algorithm is n time. and the worst time is the same

Fatih DURAL

151044041