Hw05 -- İslam Göktan SELÇUK – 141044071

PART 1: T(n) = O(n) + O(nlog(n)) + O(n) = O(nlog(n)).

```
44 # Harcanan zamana ve işlerin agirligina gore optimum
45 # cost'u bulur.
46 def scheduling(times, weights):
47     ratios = [] # times/weights oranini tutar.
48     totalCost = 0
49     currentTime = 0 # Her cost hesabından sonra zaman guncellenir.
50
51     # Listelerdeki tum oranlar ve bunlarin indeksleri
52     # iki boyutlu bir diziye atanir.
53     for i in range(0, len(times)):
54         x = []
55         x.append(times[i] / weights[i])
56         x.append(i)
57         ratios.append(x)
58
59     # Oranlarin ve indekslerin tutuldugu iki boyutlu liste
60     # mergesort ile siralanir.
61     mergeSort(ratios)
62
63     # Artan sirayla siralanan listeden toplam cost bulunur.
64     for x in ratios:
65         i = x[1] # Ratio listesinden degerin indeksi bulunur.
66         # Her defasinda zaman guncellenerek isleme dahil edilir.
67         currentTime += times[i]
68         totalCost += currentTime * weights[i]
69         print("time", i+1, " = ", currentTime,
70               ", weight", i+1, " = ", weights[i],
71               ", cost", i+1, " = ", currentTime * weights[i], sep="")
72
73     print("Total cost: ", totalCost)
74
75     return totalCost
```

O(n) zaman alır. Oranlardan n büyüklüğünde bir liste oluşturulur.

O(nlog(n)) zaman alır. Oranlardan oluşan listeyi mergesort ile sıralar.

O(n) zaman alır. Oran listesindeki her bir eleman için cost hesabı yapar.

Test:

```
4 # Test inputs
5 times = [1, 3, 5, 2]
6 weights = [10, 2, 4, 7]
7
```

Sonuç:

```
time1 = 1, weight1 = 10, cost1 = 10
time4 = 3, weight4 = 7, cost4 = 21
time3 = 8, weight3 = 4, cost3 = 32
time2 = 11, weight2 = 2, cost2 = 22
Total cost:  85
```

PART 2:

a-) Eğer New York için [3, 5, 2], San Francisco için [4, 1, 1] ve taşınma maliyeti için 20 verirsek aşağıdaki algoritma [NY, SF, SF] sonucunu verecektir. Ancak taşınma maliyetinden dolayı doğru sonucun [NY, NY, NY] olması gerekir. Bu yüzden bu algoritma yanlış sonuç üretir.

a) Show that the following algorithm does not correctly solve this problem by giving an instance which it does not return the correct answer.

for i= 1 to n

  if $N_i < S_i$ then

    Output "NY in Month i"

  else

    Output "SF in Month i"

b-) T(n) = O(n)

```python
# Iki sehirdeki ofislerde calisarak ve sehirler arasinda
# gecis yapılarak harcanan optimum cost'u bulur.
def costOfOptimalPlan(ny, sf, movingCost):
    n = len(ny)
    totalCost = 0
    if n > 0:
        # Ilk aydaki sehir direkt karsilastirma ile bulunur.
        if sf[0] > ny[0]:
            print("NY in Month 1")
        else: print("SF in Month 1")

        # Her ay icin bulunulan sehir'in cost'u ile
        # diger sehir'e gecis cost'u ve sehir'in cost'u toplanarak
        # karsilastirilir.
        for i in range(1, n):
            ny[i] += min(ny[i-1], movingCost + sf[i-1])
            sf[i] += min(sf[i-1], movingCost + ny[i-1])
            # O ay icin optimum cost'un saglandigi sehir print edilir.
            if sf[i] > ny[i]:
                print("NY in Month", i+1)
            else: print("SF in Month", i+1)

        # Son indeksteki minimum cost secilir.
        totalCost = min(sf[n-1], ny[n-1])
        print("Total cost:", totalCost)

    return totalCost
```

O(n) zaman alır. İkinci aydan itibaren her ay için cost hesabı yapılır.

Test:

```python
# Test inputs
ny = [1,3,20,30] # New York icin harcanan cost'lar
sf = [50,20,2,4] # San Francisco ofisinde harcanan cost'lar
movingCost = 10  # Sehir degistirildiginde harcanan cost
```

Sonuç:

```
NY in Month 1
NY in Month 2
SF in Month 3
SF in Month 4
Total cost: 20
```