Introduction to Algorithm Design
Hamza Yoğurtcuoğlu
171044086 - Hw5

Question 1 :

First, two distinct list defined to indicate the optimum costs for month in NY and SF. Then set first month cost as a min cost for NY and SF to these lists. For each month costs 1 to i, below formulas are used to get optimum

costs: `optimumNewYork[i]= NY[i]+ min(optimumNewYork[i-1],M+optimumSanFrancisco[i-1])`
`optimumSanFrancisco[i] = SF[i] + min(optimumSanFrancisco[i-1],M+optimumNewYork[i-1])`

Worst Case :

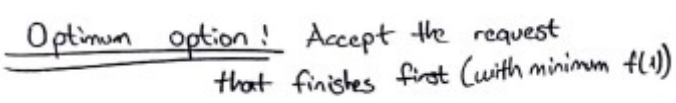
Check the sum of move cost with the cost of city is greater than cost of current city. If the sum of move cost with the cost of other city is greater than the cost of current city, stay in the current city. Then calculate all months minimum cost by this way. Time complexity of this solution is **O(n)** where n is number of months.

$$f(x)=\sum_{i=0}^{n} 1 \quad \textbf{= 1+1+1…+1 = n} \qquad f \in O(n)$$

```
------------------------------------------------
Switch city cost :  10
New York :  [50, 20, 2, 4]
San Francisco :  [1, 3, 20, 30]
Optimal Plan with minimum cost :  ['SF', 'SF', 'NY', 'NY']
Cost of optimal plan :  20
------------------------------------------------
------------------------------------------------
Switch city cost :  10
New York :  [20, 3, 4, 69, 99, 6, 19]
San Francisco :  [47, 45, 26, 2, 3, 99, 67]
Optimal Plan with minimum cost :  ['NY', 'NY', 'NY', 'SF', 'SF', 'NY', 'NY']
Cost of optimal plan :  77
------------------------------------------------
```

Question 2 :

I sorted following session according to finish time.



I have taken above note from lesson notebook.

```
start = [0 , 0 , 6 , 2 , 6 , 5]
finish = [2 , 3 , 10 , 5 , 7 , 60]
```

We can attend maximum 3 sessions. The maximum set of sessions that can be attended are like following output :

```
We are sorting according to finish of sessions
Start :   [0, 0, 2, 6, 6, 5]
Finish :  [2, 3, 5, 7, 10, 60]
The following sessions are selected
Session No : 0  start :  0  finish :  2
Session No : 2  start :  2  finish :  5
Session No : 3  start :  6  finish :  7
```

The greedy algorithm select next session which earliest finish time session. So, I sort the arrays according to finishing time.

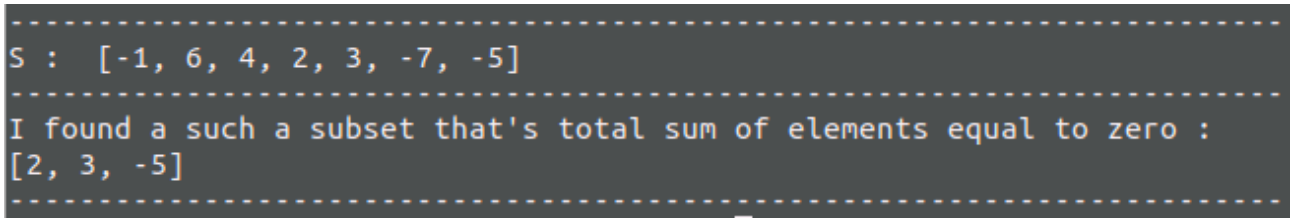
Worst Case :

If our sessions are sorted according to finish time. It will take **O(n)** runnning time. Because we are looking finished time. I forward one by one in array. In worst case scenario. This times can be unsorted according to finish time. So, it will take **O(nlong)**.

Question 3:

I create two subset of array. Then I create its combination, I divide to sub problems for I used it later. Before solving the in-hand sub-problem, dynamic algorithm will try to examine the results of the previously solved sub-problems. The solutions of sub-problems are combined in order to subset that's total sum of elements equal to zero.

So I am using to before subarray to finding other subarray. So, I am not create sub-array through dynamic programming in which sub-problem solutions are Memorized rather than computed again and again. So recursive call helps for combination of subarray until finding to subset that's total sum of elements equal to zero.



If I found elements of that subset and terminate the algorithm.

Worst Case :

The time complexity of above solution is **O(n x sum)** . Auxiliary space used by the program is also **O(n x sum)**
Because each time is we are looking to subset and we are adding to list for using later. We can use Hashmap for this but that works with list.

$$f \in O(NxSum)$$

Question 4 :
Let M =size of SequenceA
Let N= size of SequenceB
the computation is arranged into an (N+1) × (M+1) array where entry (j,i) contains similarity between SequenceA[1.....j] and SequenceB[1.....i]. The algorithm computes the value for entry(j,i) by looking at just three previous entries:

- (j-1,i-1) Diagonal Cell to entry (j,i)
- (j-1,i) Above Cell to entry (j,i)
- (j,i-1) Left Cell to entry (j,i)
- j-1,i
- j,i
- j-1,i-1
- j,i-1

p(j,i)= +2 if SequenceA[j]=SequenceB[i] (match Score) and p(j,i)= -2 if SequenceA[j]!=SequenceB[i].

The maximum value of the score of the alignment located in the cell (N-1,M-1) and the algorithm will trace back from this cell to the first entry cell (1,1) to produce the resulting alignment . If the value of the cell (j,i) has been computed using the value of the diagonal cell, the alignment will contain the SequenceA[j] and SequenceB[i]. If the value has been computed using the above cell, the alignment will contain SequenceB[j] and a Gap ('-') in SequenceB[i]. If the value has been computed using the left cell, the alignment will contain SequenceA[i] and a Gap ('-') in SequenceB[j]. The

resulting alignment will produce completely by traversing the cell (N-1,M-1) back towards the initial entry of the cell (1,1).

```
------------------------------------------------------------
Sequence A =  ALIGNMENT , Sequence B =  SLIME
match_score =  2 , mistmatch_score =  -2 , gap_score =  -1
matchCount =  4 , misMatchCount =  1 , gapCount =  2
Alignment of sequence A and B :
ALIGNMENT
SLI--ME
Cost =  4
------------------------------------------------------------
------------------------------------------------------------
Sequence A =  HAMZAAY , Sequence B =  HEZYAAY
match_score =  2 , mistmatch_score =  -2 , gap_score =  -1
matchCount =  5 , misMatchCount =  2 , gapCount =  1
Alignment of sequence A and B :
HAMZ-AAY
H-EZYAAY
Cost =  5
------------------------------------------------------------
```

Worst Case :

Worst Case is depends on two length of strings. If they have same length.

$$f \in \Theta(n^2)$$

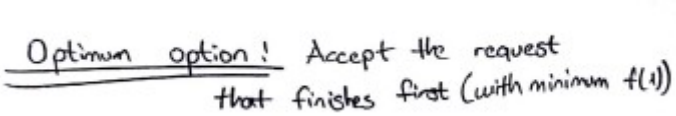
But they could be different length of strings.
(m : Length of SequenceA, n : Length of SequenceB)

$$f \in \Theta(n \, x \, m)$$

Question 5:

If we want to perform minimum number of operations.
Step 1 : We need to find 2 minimum 2 elements(A and B). So, Each time we look array 2 minimum element and removed from array. Then we append summution of A and B. Sum of A and B is appended to array again.
Step 2 : If size of array is 1. We found the sum of elements with minimum operations. If size of array is not 1. We get back to **Step 1** again.

I used following option in Greedy algorithm.



I have taken above note from lesson notebook.

```
You are given array :  [10, 20, 25, 26, 30]
A =  10 , B =  20 , A+B =  30
A =  25 , B =  26 , A+B =  51
A =  30 , B =  30 , A+B =  60
A =  51 , B =  60 , A+B =  111

Sum of array's elements =  111
Total Operation :  252
-------------------------------------
-------------------------------------
You are given array :  [5, 4, 3, 2, 1]
A =  1 , B =  2 , A+B =  3
A =  3 , B =  3 , A+B =  6
A =  4 , B =  5 , A+B =  9
A =  6 , B =  9 , A+B =  15

Sum of array's elements =  15
Total Operation :  33
-------------------------------------
```

Worst Case :

The Complexity of Algorithm depends on either sorted or unsorted array. Because we are looking 2 minimum elements in terms. If array is sorted complexity

$$f(x)=\sum_{i=0}^{n} 2 \quad \text{= 2+2+2...+2 = 2n} \qquad f \in O(n)$$

If array is unsorted , we are looking all array each term.

$$f(x)=\sum_{i=0}^{n} n-1 \quad \text{= (n-1) + (n-2) . . . + 1 = n(n-1)/2}$$

$$f \in \Theta(n^2)$$