① For dynamic programming, we need to keep the optimal arrays in which the cities of NY and SF are kept at each iteration cost. Displacement cost should be taken into account when adding elements to these arrays. Also, whichever of the two cities cost less, it should be added to the arrays. For each month costs 1 to i, below formulas are used to get minimum costs.

$$optN[i] = NY[i] + \min(optN[i-1], m + optS[i-1])$$
$$optS[i] = SF[i] + \min(optS[i-1], m + optN[i-1])$$

Checked the sum of move cost with cost of city is greater than rest of current city. If the sum of move cost with the cost of other city is greater than the cost of current city. Then we calculate all months min cost by this way. ⟶ worst case running time.

So $T(n) = O(n)$ → Linear time complexity. We have a single for Loop. This Aogram works as the number of elements array.

② Greedy choice is always pick the next activity whose finish time is least among the remaining activities and the start time is more than or equal to the finish time of previously selected activity. We can sort the activities according to their finishing time so that we always consider the next activity as minimum finishing time activity.

We have below algorithm:

→ Firstly we sort the activities according to their finishing time

→ We select the first activity from the sorted array and print it

→ We do following for remaining activities in the sorted array. If the start time of this activity is greater than or equal to the finish time of previously selected activity then we select this activity and print it

We can clearly see that the algorithm is taking a $O(n)$ running time where n is the number of activities. Also if the array passed to the function are not sorted, we can sort them in $O(n\log n)$ time.

(3) Designed a dynamic programming algorithm to check whether there is a subset with the total sum of elements equal to zero.

The algorithm find negative or positive number total equal zero in the array. And this program return according to boolean (True or False). If this array have sum equal to 0 return True, and program print "The set has a subset sum" Otherwise the program print "The set does not have a subset"

A new subset array was created for the dynamic approach and the values were saved. At this point, dynamic programming was the most efficient approach for the algorithm. So we can say for Time complexity analyse worst case

O(n)

④ We can find best path through the matrix for this problem And we use local alignment with approach dynamic programming

Time analysis

O(mn) time complexity

O(mn) space, can be brought to O(m+n)

This problem is a sequence alignment problem

for dynamic programming Let V(i,j) be the optimal alignment score of $S_{1...i}$ and $T_{1...j}$ $(0 \le i \le n)$ $0 \le j \le m$. V has following properties base conditions

$$V(i,0) = \sum_{k=0}^{i} \sigma(S_k, -)$$

$$V(0,j) = \sum_{k=0}^{j} \sigma(-, T_k)$$

$V(i,0) = 0$
$V(0,j) = 0$

Recurrence relationship

$$V(i,j) = \max \begin{cases} 0 \\ V(i-1, j-1) + \sigma(S_i, T_j) \\ V(i-1, j) + \sigma(S_i, -) \\ V(i, j-1) + \sigma(-, T_j) \end{cases}$$

pseudo code

```
for i=0 to n do
begin
  for j=0 to m do
  begin
    calculate V(i,j) using
    V(i-1,j-1), V(i,j-1) and V(i-1,j)
  end
end
```

Time complexity O(mn) space complexity O(n+m) if only V(S,T) is required and O(mn) for the reconstruction of the alignment

firstly finded longest score call in my program. Used a matrix and Running code with equal length two strings I tried example example two strings of hw pdf but it has to be space for gap value so my two string

"ALIGNME"
"SLI ME"
  gap

(5) This program designed a greedy algorithm to calculate the sum of the array with the minimum number of operations.

Example array has negative or positive number for sum operations. Every array element sum return operation number, This algorithm is the simplest example to the greedy algorithm. But If element of array not only positive So we must think elements as absolute value. We realized example following

$8 + 5 = 13$

$(-8) + 5 = 13$

$8 + (-5) = 13$

$(-8) + (-5) = 13$

All sum operation results is same and absolute total

This algorithm time complexity analysis obtained as

$O(n)$