CSE321 Introduction to Algorithm Design Homework5 Report

Part1

To solve this problem, I implemented 2 methods. **optimalPlan**, and **optimalPlanDynamicAlgorithm**. optimalPlan calls optimalPlanDynamicAlgorithm methods if there are no errors. optimalPlanDynamicAlgorithm methods creates 2 arrays OPT_NY and OPT_SF to solve problem with dynamic algorithm. Then in a loop, with range size of input arrays which are the parameter of the method, I filled OPT_NY and OPT_SF.

OPT_NY array filled with sum of NY and minimum of the one before OPT_NY and sum of M (cost of move) and one before SF. Looking one before index is for understanding the movement is reasonable or not.

OPT_SF array filled with sum of SF and minimum of the one before OPY_SF and sum of M (cost of move) and one before NY. Looking one before index is for understanding the movement is reasonable or not.

Worst-Case Scenario:

The algorithm has size of arrays iteration. And each iteration takes constant time O(1). So that the total running time of **worst case is O(n).**

Part2

To solve this problem, I implemented one method. It is **optimalSession** method. Parameter of the method is 2D array of activities. [[StartTime,EndTime],[StartTime,EndTime]]. First of all, I sorted to 2D array based on the second parameter to make a greedy approach. After the sort operation, I added the first activity to my returnValueArray. Then in a loop with range 1 to size of 2D array, I checked the start and end time of the next activity with old activity. If new activity starts after the old activity's end, then I added the activity into the returnValueArray.

Worst-Case Scenario:

Sorting has O(n*log(n)) time complexity. Then the algorithm has size of array iteration. And each takes constant time O(1). It is O(n*log(n) + n). After throwing lower terms, the total running time of **worst case is O(n*log(n)).**

Part3

To solve this problem, I implemented **totalSumEqualsToZero** and **totalSumEqualsToZeroDP** methods. totalSumEqualToZero method, created 2 times 2D array. With row count is size of array, column count is MAX_SUM. Then it created an empty array to store subset with sum of items equal to zero. Then it called totalSumEqualsToZeroDP method with, 0“(index)”, 0“(sum)”, array, len(array), table_of_dp“(2D array)”, is_checked“(2D array)”, sum_equal_zero_array“(empty array)”. The best case of the totalSumEqualsToZeroDP method is value of index equals to value of len of array. In this statement, if sum equals to zero there is a subset with sum of item equals to zero. But it can be empty subset. So that I check sum_equal_zero_array’s size. If it is different from zero, the subset is not empty. Then I printed the subset and terminated the program. Otherwise I returned False to continue. The other case(not being best case), firstly I checked is_checked array. If is_checked array is True then I return the value of table_of_dp. This 2D array holds True or False value. True means for is_check, it is solved. True means for table_of_dp, it has zero sum. In each case I append the item into the sum_equal_zero_array to print it. Before recursive call I check the table_of_dp is solved, if it is not solved, I recall the method with index + 1, sum + array[index] and other parameters. Meaning of sum + array[index] is sum the items. Otherwise return True to terminate program. After recursive call I deleted the last item from sum_equal_zero_sum_array. Meaning of it, problem did not solve so remove items and to add new items and control if its sum equals to zero. Then I made a new recursive call to search new subsets with sum of items equal to zero.

Worst-Case Scenario:

There is a 2D array(table_of_dp) array with size n time MAX_SUM. Recursive method searches each index. So that the time complexity is O(n * MAX_SUM). n means that the size of array. But in the worst-case MAX_SUM goes infinity. In the time complexity we a looking the size in the infinity so n equals to infinity and SUM_MAX equals to infinity. So that total running time of **worst case is O(n*MAX_SUM) = O(n^2).**

Part4

To solve this problem, I implemented **printAlignment** and **findAlignment** methods. findAlignment methods takes 2 arrays. These are string1 and string2. First of all, I create a 2D table to make solution dynamic programming. This table has len(string1) + len(string2) + 1 row and len(string1) + len(string2) + 1 column. I filled the first crow and first column with gap_score. It is “-1”. The in nested for loop with range 1 to len_string1 + 1 and other loop with range 1 to len_string2 + 1. There are some important cases to fill table. If strings are matches fill the table[i][j] with table[i - 1][j - 1]. Otherwise there are 3 cases. Firstly, they could mismatch, secondly, string1 could be underscore, thirdly, string2 could be underscore. In this case I have to find max number of these cases. The reason finding max, gap_score"-1" and mismatch_score"-2" has negative values. Filling table operation is done. Then I create 2 array with size len_string1 + len_string2 + 1 to fill new strings. Then in a loop I checked sum controls. Loop starts from the right bottom of the table.

The loop is an infinity loop. There are 4 cases.

1) Strings could be matches. In this case, I filled the strings result with their value, decreased both index value of strings, increased both temp length of the strings and match count.

2)Strings could be mismatch. In this case, I filled the strings result with their value, decreased both index value of strings, increased both temp length of the strings and mis_match count.

3)The string1 could be underscore"_". In this case, I filled the string1Result with "_", filled string2Result with its value, increased both index value of strings. I increased the temp_length2 value and I increased the temp_length1 value and gap_score if it is range of string1's size values.

4)The string2 could be underscore"_". In this case I filled string2Result with "_", filled string2Result with its value, decreased both index of strings. I increased the temp_length1 value and I increased the temp_length2 value and gap_score if it is range of string2's size value.

If "i" or "j" becomes 0 the condition of infinity loop becomes False and Terminate the loop. There are 2 loops. These are filled string1Result and string2Result if above while loop terminated before the end string1 or string2.

I assumed the answer size of len_string1 + len_string2 + 1. So that there could be non-necessary underscore"_". So that in a loop I found the index of first character(no underscore) starts. Then I called the printAlignment methods and print the alignment.

Finally, I calculate the $\boldsymbol{Cost = N * match\_score + M * mismatch\_score + K * gap\_score}$ and returned the cots.

Worst-Case Scenario:

There is a nested for loop with range len_string1 and len_string2 to fill table. This nested loop works len_string1 * len_string2 times. Infinity while loop works len_string1 + len_string2 time in the worst case. And other loops work in the worst case len_string1 or len_strign2 times. We can throw the lower terms, so that total running time of **worst case is O(len_string1 * len_string2) = O(m * n). (If length of strings are equal then it is O(n^2)).**

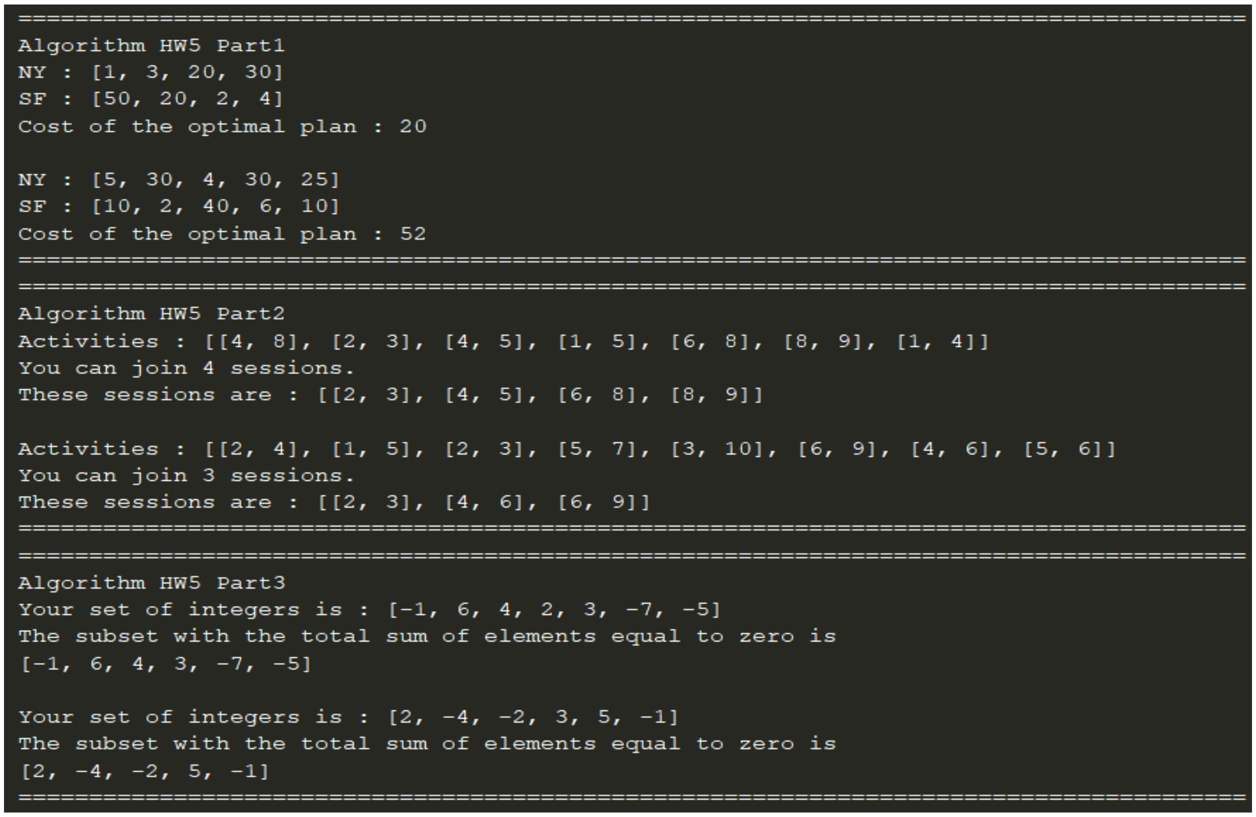
Part5

To solve this problem, I implemented **makeAllPositive** and **findMinimumNumberOfOperations** methods. In findMinimumNumberOfOperations method, firstly I called makeAllPositive method. makeAllPositive method creates a 2D array and mapped array of integers. If item is negative mapped it with False, otherwise mapped it with True and returns the mapped 2D array. After mapping, I sorted the mapped array based on the first item. The reason of sorting mapped array is making minimum operation to sum all item. Then I a loop with range size of mapped array I made sum operation and found operation count. If loop count is 1 than operation count is sum of values of index 0 and index 1. Otherwise operation count is total sum plus index “i” item. Important thing is that, if sum value is smaller than zero make it positive and calculate the operation count. Otherwise do nothing just add. Finally, I returned the sum of item and minimum operation count.

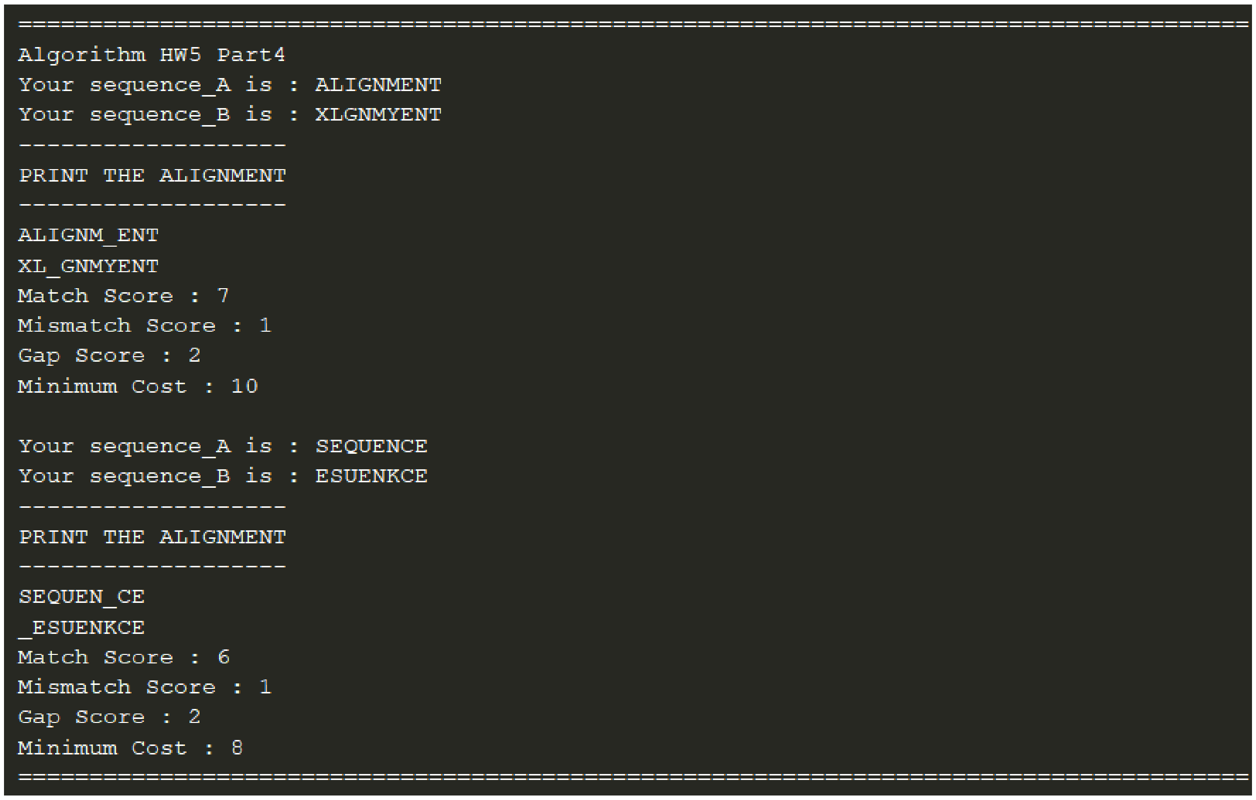
Worst-Case Scenario:

Mapped item has O(n) time complexity. Sorting has O(n*log(n)) time complexity. Then the algorithm has size of 2D array iteration. And each takes constant time O(1). It is O(n + n*log(n) + n). After throwing lower terms, the total running time of **worst case is O(n*log(n)).**

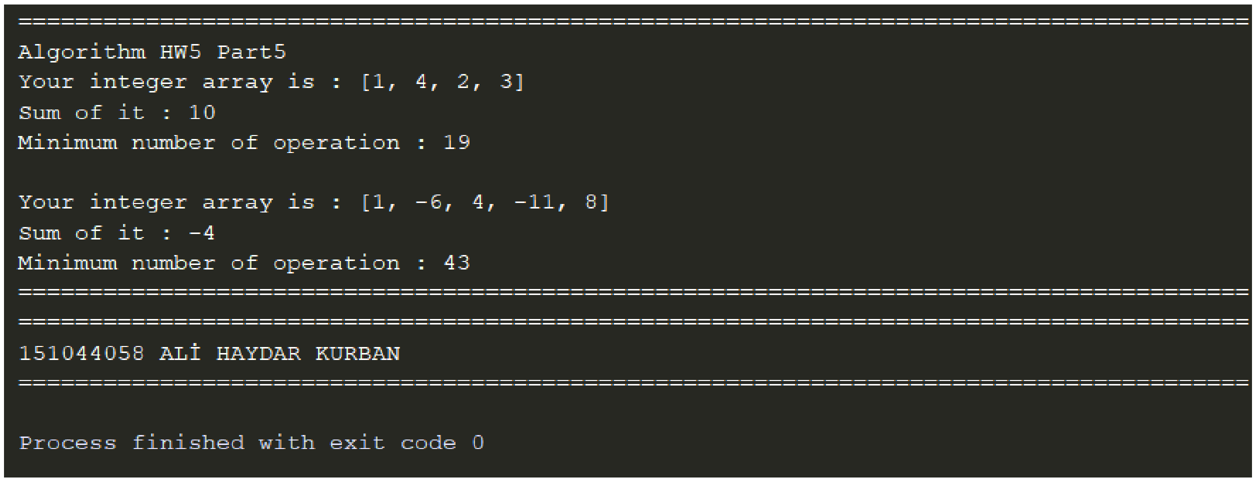
Expected Outputs of Given Driver Methods



```
==========================================================================================
Algorithm HW5 Part1
NY : [1, 3, 20, 30]
SF : [50, 20, 2, 4]
Cost of the optimal plan : 20

NY : [5, 30, 4, 30, 25]
SF : [10, 2, 40, 6, 10]
Cost of the optimal plan : 52
==========================================================================================
==========================================================================================
Algorithm HW5 Part2
Activities : [[4, 8], [2, 3], [4, 5], [1, 5], [6, 8], [8, 9], [1, 4]]
You can join 4 sessions.
These sessions are : [[2, 3], [4, 5], [6, 8], [8, 9]]

Activities : [[2, 4], [1, 5], [2, 3], [5, 7], [3, 10], [6, 9], [4, 6], [5, 6]]
You can join 3 sessions.
These sessions are : [[2, 3], [4, 6], [6, 9]]
==========================================================================================
==========================================================================================
Algorithm HW5 Part3
Your set of integers is : [-1, 6, 4, 2, 3, -7, -5]
The subset with the total sum of elements equal to zero is
[-1, 6, 4, 3, -7, -5]

Your set of integers is : [2, -4, -2, 3, 5, -1]
The subset with the total sum of elements equal to zero is
[2, -4, -2, 5, -1]
==========================================================================================
```



```
=====================================================================================
Algorithm HW5 Part4
Your sequence_A is : ALIGNMENT
Your sequence_B is : XLGNMYENT
-------------------
PRINT THE ALIGNMENT
-------------------
ALIGNM_ENT
XL_GNMYENT
Match Score : 7
Mismatch Score : 1
Gap Score : 2
Minimum Cost : 10

Your sequence_A is : SEQUENCE
Your sequence_B is : ESUENKCE
-------------------
PRINT THE ALIGNMENT
-------------------
SEQUEN_CE
_ESUENKCE
Match Score : 6
Mismatch Score : 1
Gap Score : 2
Minimum Cost : 8
=====================================================================================
```



```
=====================================================================================
Algorithm HW5 Part5
Your integer array is : [1, 4, 2, 3]
Sum of it : 10
Minimum number of operation : 19

Your integer array is : [1, -6, 4, -11, 8]
Sum of it : -4
Minimum number of operation : 43
=====================================================================================
=====================================================================================
151044058 ALİ HAYDAR KURBAN
=====================================================================================

Process finished with exit code 0
```

ALİ HAYDAR KURBAN 151044058