CSE 321 – Introduction To Algorithm Analysis Homework #5

Gökçe Nur Erer – 171044079

Question 1:

The algorithm designed for this question is created by the observation that the optimal moving plan either ends and New York or San Francisco. If the optimal moving plan ends at New York that means you have to pay:

1) Cost of operating at n^{th} month which is N_n + Cost of the optimal moving plan at $n\text{-}1^{th}$ month which ends at New York
2) Cost of operating at n^{th} month which is N_n + Cost of the optimal moving plan at $n\text{-}1^{th}$ month which ends at San Francisco + Moving Cost

Same logic applies for the optimal moving plan ending at San Francisco as well. The optimal moving plan O_{NY} (n) will denote the minimum costed moving plan which ends at New York from month 1 to n. The optimal moving plan O_{SF}(n) will denote the minimum costed moving plan which ends at San Francisco from month 1 to n. As a result:

$$O_{NY}(n) = N_n + \min (O_{NY}(n-1), O_{SF}(n-1) + M)$$

$$O_{SF}(n) = S_n + \min (O_{SF}(n-1), O_{NY}(n-1) + M)$$

In the worst case this algorithm will run n times, n denoting the month count. Other operations take constant time. Therefore, overall time complexity is O(n).

Question 2:

The algorithm designed for this question is created by the observation that if the student chooses the session which finishes the fastest, he/she can attend to the maximum sessions possible. So the algorithm first gets a list of sessions with their names, starting times and finishing times. Then the algorithm sorts the session list by their finishing times. Then the algorithm checks for each session if its starting time overlaps with any of the other session's finishing time. And to do that whenever a suitable session is added to the resulting list, the finishing time compared to the starting time is changed with the last session's finishing time added to the resulting list.

In the worst case and all the other cases this algorithm will run n times which results in the time complexity O(n).

Question 3:

Question 4:

To find an alignment with dynamic programming we must create a dynamic programming table to keep track of the costs. Each side of the table represents the two strings which will be aligned. Rows represent the second string's characters and columns represent the first string's characters. The 0th row and column represents a gap. To arrive a cell there are 3 options:

1) From the cell above which means the character represented by that cell for second string is aligned with a gap
2) From the cell to the left which means the character represented by that cell for first string is aligned with a gap
3) From the cell to the above-left which means the characters represented by both first string and second string are aligned together which might result in a match or a mismatch.

To initialize this dynamic programming table first the algorithm initializes the 0th row and column with given gap score and summing up the score at each cell with the previous one to the left. Because going right in the 0th row means that first string's characters are getting aligned with gaps. And each time it is moved to right gap score increases. Same applies for the 0th column. Because going down in the 0th column means that the second string's characters are getting aligned with gaps. And each time it is moved to down the gap score increases.

After the initialization, the dynamic programming table has to be filled up to find the cost. So to find each cell's cost value the algorithm uses the approach mentioned above with 3 options. If the maximum of the left cell value, above cell value and above-left cell value results in:

1) Left cell value: Gap score is added to the left cell value since now there is a gap in the alignment. And then the calculated value is the new value of the current cell.
2) Above cell value: Gap score is added to the above cell value since now there is a gap in the alignment. And then the calculated value is the new value of the current cell.
3) Above-left cell value:
    a. If the characters represented with the current cell by both first string and second string have a match, match score is added to the above-left value and then the calculated value is the new value of the current cell.
    b. If the characters represented with the current cell by both first string and second string have a mismatch, mismatch score is added to the above-left value and then the calculated value is the new value of the current cell.

The cell at the rightmost and downmost will be the overall cost after filling in the dynamic programming table. So algorithm returns that value ones it fills up all the cells in the table.

The worst case complexity for this algorithm is O(n*m), n being the first string's length and m being the second string's length since the dynamic programming table will have n*m elements and the algorithm will run for each cell in the dynamic programming table. If we consider that the length of the two strings are the same then the overall complexity can be rewritten as $O(n^2)$.

Question 5:

The algorithm designed for this question is created by the observation that if the sum is calculated by choosing the smallest 2 costs, meaning for example in [5,2,4,9] first calculation will be done with 2 and 4 since they are have the smallest costs will be 6 in total. For the next addition the smallest two costs should be added to the operation count. Which means 5 and 6 (being 2 + 4) and so on. The time complexity for this question is $O(n^2)$ since for n times we find the minimum cost pair. Finding the minimum is O(n) itself since it requires traversing the whole list. So overall $O(n*n) = O(n^2)$.