CSE 321 - Homework 5
Sena Özbelen - 1901042601

1-) This algorithm is based on binary search. It has start and end parameters to call the halves. Unlike binary search, it calls both halves since this is divide and conquer algorithm. When input size is 1, it returns the word itself since the longest substring is the word again. When it is more than 1, it gets substrings from both halves and compare them to find their longest common substring.
Time Complexity: T(n): 2T(n/2) + k where k is the smallest substring's length
: O(n) as approx.

2-)

a. This is again based on binary search. If input size is 1, then the only price is the current price so it returns its index and the max profit is 0. If the size is more than 1, it gets the index of minimum price and maximum price and indices that gives us the max profit in the corresponding half. Now, we have to compare 3 profits, profits from the first and second halves and the new profit calculated by using the minimum price of the first half and the maximum price of the second half. The max profit's indices are returned.
Time Complexity: T(n): 2T(n/2) + c
: O(n) as approx.

b. The second part is based on linear scanning. Since we have to go through the list once, we should keep the max profit's max and min prices and a temporary min value to calculate the current profit with the current value using as max value. If current profit is higher than max profit, then update the max profit with the current profit and also update min and max prices.
Time Complexity: O(n)

c. In terms of time complexity, they have the same complexity. The second one should traverse the list. The first one uses divide and conquer algorithm but the complexity of F(n) does not depend on the size of the input so this does not give us n complexity. Therefore, from Master Theorem, 2 > 1 (a > b^d) gives us n^(loga (base = b)) == n

3-) Recurrence relation of the question : T(n) = T(n-1)+1, T(1) = 1
We can decrease the size by one and when the size is 1, it returns 1 since there is only one number and this is already an increasing array. For other cases, if the previous number is lower than the current number, increase the

counter by one, otherwise, reset it to 1 again. For DP solution, we can traverse the array and increase the counter if it is still increasing. In every step, we can store the counter in an array for every number and we can easily find the biggest subarray and its location.
Time Complexity: O(n) since we should traverse the array.

4-)

a. This algorithm stores the max points for each position in a dynamic table array. The final position stores the max point for our path so that we can easily find out the max point. To fill the dynamic table, the boundary cells (first row and first column) have only one route so we find their max points by going through this route. For other cells, there are two options, upper cell or left cell. We choose the max one to get the max point.
Time Complexity: O(m*n) as approx.

b. Since Greedy Algorithms tend to choose the most feasible way at that time, we choose the cell having higher point between two options. If there is only one way to go (boundary cells), then it should keep going through this route.
Time Complexity: O(m+n-1) as approx.

c. Both brute force and dynamic programming are more accurate than greedy one because they try all routes in order to find the max route but greedy algorithms choose the feasible one without considering the whole. However, Greedy gives the fastest way to find a route since it does not try to find all combinations. Dynamic Programming is slower than greedy algorithm but faster than brute force since we do not need to go through every route again and again while searching for the max route. We can easily fill the table by using for loops but brute force algorithm checks every single combination by recalculating the points so this makes it the slowest one.