CSE321 HW5

Q1

The designed algorithm splits the given array of words into two subarrays then solves them and returns the longest common substring of them.

The best case would be having no common substring in given words which would have n-1 comparisons for n number of words where n = 2^k for some positive integer k. So, the best-case time complexity is linear.

The worst case would be having only the same word repeated n times. Assuming the repeated word is length of m characters, the total number of comparisons would be m * (n – 1) which is

$$O(nm)$$

Q2

a)

I can’t analyze the explained algorithm because I couldn’t design one.

b)

A non-recursive linear algorithm would be starting from the first element and check until the last element since that is the direction that the merchant is allowed to buy and sell.

The algorithm basically always keeps the cheapest day index until the current checked day, whenever the checked day price is less than the cheapest day price it is updated.

If checked day does not have a price less than the cheapest day price this means it is possible for it to be a better sell day. So, it’s checked if the current max profit is less than buying at cheapest day and selling at current day. If so buying and selling days are updated.

Since the whole array is iterated no matter what the prices are the best and worst time complexities are both linear.

c)

I can’t compare the algorithms because I couldn’t design the first one. But if I could it probably wouldn’t have a better time complexity than the one designed for the b section.

Q3

The designed algorithm iterates the given array and stores the result of the already iterated subparts of it in an array so following subparts can be computed using them. The values that are stored are the lengths of the consecutively increasing subparts that include the value in given array at the same index. After every subpart length is computed the maximum of them is returned. In total there are n comparisons to compute subpart lengths and another n comparisons to get the maximum length. The worst case time complexity is linear time.

Q4

a)

The algorithm designed with dynamic programming algorithm design in mind starts with the last cell and computes the total points that can be collected for every cell in the last row and column. This is done for every row and column whose cells are computed in reverse order, as in from the side of the ending cell and to the starting cell, since the cells of prior, meaning one after towards the ending cell, rows and columns are computed they can also be computed comparing their east and south cells.

The worst time complexity is quadratic time since for every cell their south and east cells are compared. In theory the last row and column only has linear comparison count but for the sake of simplicity the algorithm was implemented with no special cases for rows and columns.

b)

The algorithm designed with greedy programming algorithm design in mind assumes that the best route always goes through the next cell that has the greater point available. So, the algorithm takes only one path always following the next greatest point available. This makes it a linear algorithm.

But this assumption is not always correct and with greater sizes the accuracy is also greater. If given maps are generated randomly it can be assumed that the algorithm has an accuracy rate that is less than but maybe proportional to the dynamic algorithm.

But there are edge cases that are not random resulting in absolute failures such as

1	0	$\alpha_{0,2}$	$\cdots$	$\alpha_{1,m}$
1	0	$\alpha_{1,2}$	$\cdots$	$\alpha_{2,m}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
1	0	$\vdots$	$\ddots$	$\alpha_{n-1,m}$
1	$\alpha_{n,1}$	$\cdots$	$\alpha_{n,m-1}$	$\alpha_{n,m}$

Here the algorithm simply skips every value whenever column index is larger than 1 and row index is not n.

c)

In the previous assignment's report, I had said that the recursive route-finding algorithm had O(nm) time complexity. Which is wrong, the correct equation that shows the number of comparisons for that algorithm is actually

$$T(n,m) = T(n-1,m) + T(n,m-1) + 1$$

$$T(0,0) = 0$$

Which is hard to solve. After some tests it turns out its time complexity is exponential. I still don't know what its time complexity is exactly, but it is far worse than the other two methods.

Between the dynamic and greedy algorithms greedy one is faster but does not always find the correct result. And it probably performs worse in real life data, than it does for random data. So even though the dynamic algorithm is not the fastest one it's the most preferable one, due to its 100% correctness and being only quadratic.

To put how much better the dynamic algorithm is than the brute-force one into perspective, it takes same amount of time for better one to solve 1250x1250 sized grid while worse one solves 13x13 grid.