CSE 321 - Homework 5

Due date: 08/01/2023, 23:59

Çağla Şahin

1. **Solution:**

Explanation of the code:

This code is designed to find the longest common substring among a list of strings.

The longest_substring() function is a recursive function that uses *divide and conquer* to find the longest common substring. It takes in the list of strings, strings, and the indices of the current sublist being considered, low and high.

The base case of the recursion is when low == high, in which case the function returns the only string in the sublist. Otherwise, it finds the midpoint of the sublist and recursively calls itself on the left and right halves of the sublist. It then calls the helper_func() function on the results of these recursive calls to find the longest common substring between them.

The helper_func() function compares the characters at each index of its two string arguments, first and second, and builds up a string of the common characters. It returns this string when either a non-matching character is encountered or the end of one of the strings is reached.

The driver code prompts the user to input a number of strings and stores them in a list. It then calls the longest_substring() function on this list and prints the result.

2.a Solution:

Explanation of algorithm:

To design a divide-and-conquer algorithm to solve this problem, we can follow these steps:

1. Divide the input array into two halves.
2. Recursively find the maximum profit that can be obtained from the left and right halves.

3. Find the maximum profit that can be obtained by buying in the left half and selling in the right half.
4. Return the maximum of the profits obtained in step 2 and 3.

The maximum profit that can be obtained by buying in the left half and selling in the right half can be found by considering three cases:

1. The maximum profit is obtained by buying the lowest price in the left half and selling the highest price in the right half.
2. The maximum profit is obtained by buying the second lowest price in the left half and selling the highest price in the right half.
3. The maximum profit is obtained by buying the third lowest price in the left half and selling the highest price.

2.b Solution:

Explanation of algorithm:

To design an algorithm that is not based on the divide-and-conquer approach to solve this problem in linear time, we can use a single pass through the input array to find the maximum profit.

This code is a function that takes in a list of prices prices and returns the maximum profit that can be obtained by buying and selling the goods at the given prices.

The function starts by initializing the variables max_profit and min_price to 0 and the first element of the prices array, respectively.

It then iterates through the prices array, starting from the second element, and updates the min_price and max_profit variables at each step. The min_price variable is updated to the minimum of the current min_price and the current element of the prices array. The max_profit variable is updated to the maximum of the current max_profit and the difference between the current element of the prices array and the min_price.

Finally, the function returns the max_profit variable.

2.c Solution:

Worst-Case Time Complexities of functions

The worst case complexity of first function is O(n * log(n)), as it involves dividing the input array into halves and recursively finding the maximum profit.

In the worst case, the input array is not already sorted and the maximum and minimum values are at the ends of the array. This would require the function to recursively divide the array into halves and find the maximum and minimum values at each step, leading to a time complexity of O(n * log(n)).

The worst case time complexity of second function is O(n), as it involves a single pass through the input array to find the maximum profit.

In the worst case, the input array is not already sorted and the maximum and minimum values are at the ends of the array. This would require the function to compare the minimum price encountered so far with each element in the array and update the maximum profit at each step, leading to a time complexity of O(n).

3. Solution:

This function, longest_increasing_subarray, that takes in an array arr and returns the length of the longest increasing sub-array. It does this using a dynamic programming approach.

The function first initializes the lengths array to store the lengths of the longest increasing sub-array ending at each index. It then loops through the input array and, for each element, looks at all the elements before it to see if it can extend the current sub-array. If it can, it updates the lengths array accordingly. After the loop, the function returns the maximum value in the lengths array, which is the size of the longest increasing sub-array.