HOMEWORK 4

Yunus Emre Geyik / 1801042635

All algorithms are also commented on the code line by line.

QUESTION 1

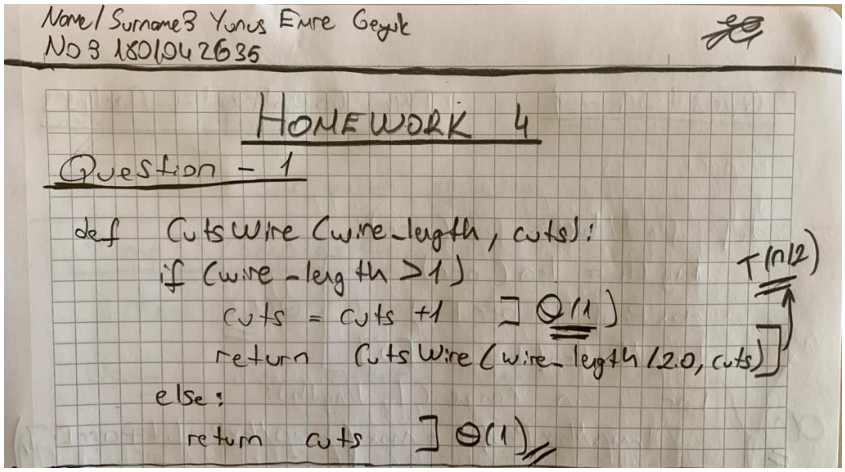
I used the decrease and conquer method to solve the problem. In this, I reduced the steel wire length with a constant factor.

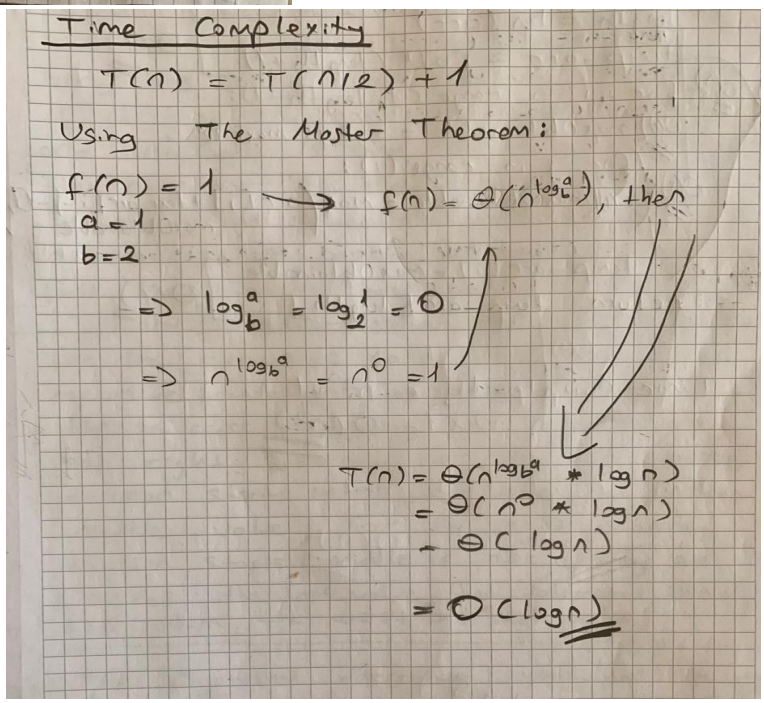
The algorithm first checks if the length of the steel wire is greater than 1. If it is greater than 1, the cuts counter (initially 0) representing the minimum number of cuts is increased by one and the steel wire length is divided by 2.0 and the reduction is done by a constant factor. Then the function is called recursively and in each call, the cuts counter is incremented by one because the wire is reduced by a constant factor, as if a cutting operation is done, and this process continues until the steel wire length is less than 1. The reason why the steel wire is reduced by dividing by 2.0 here, I wrote it as a float value to give correct results even when the length of the steel wire is a single number.

If the length of the steel wire is less than 1, the cuts counter is returned directly.

In the main function, I showed these operations with an odd and even length wire and printed them on the screen for better understanding.

I tested the code I wrote in main with 2 different bar lengths.





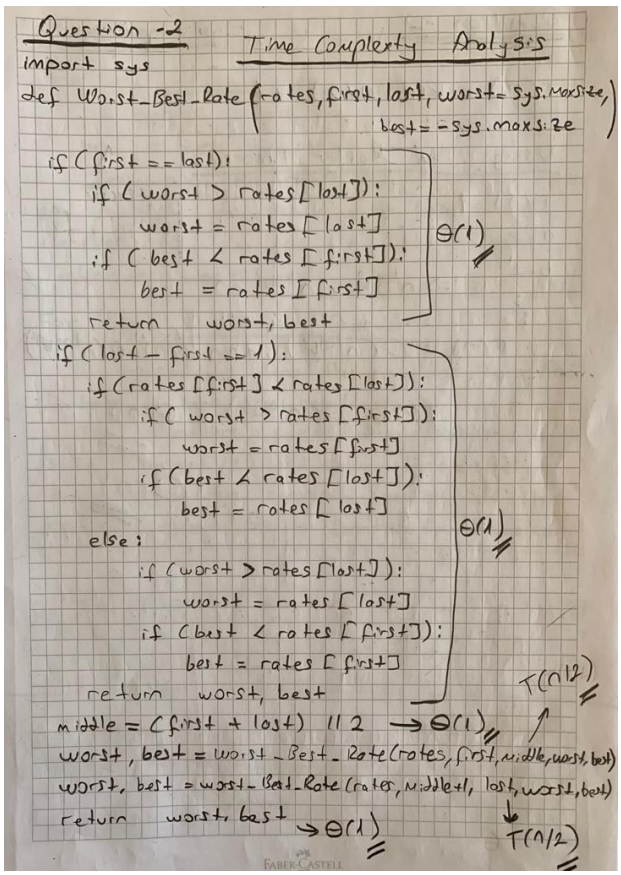
QUESTION 2

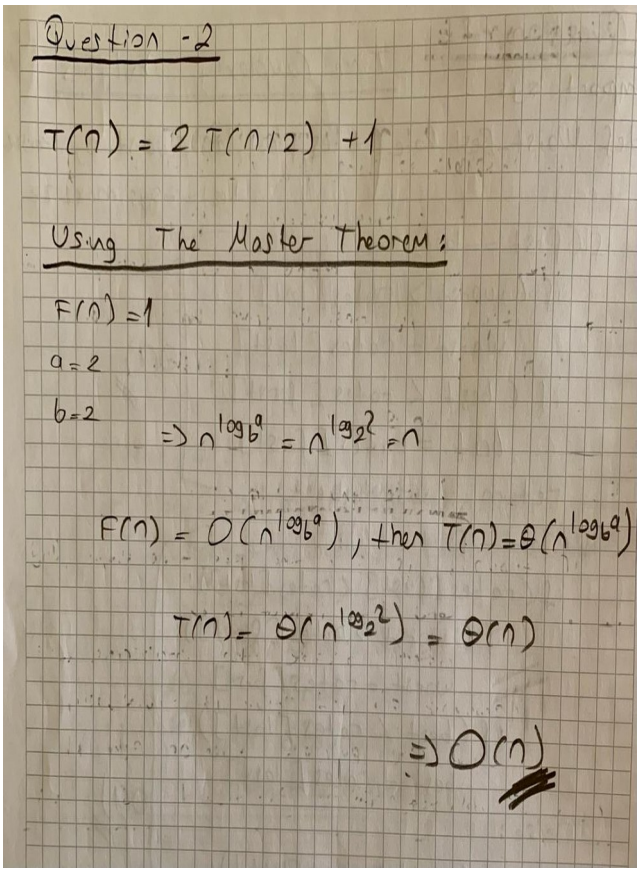
To solve the problem, I recursively divided the given array into two sub-arrays using the divide and conquer method and examined these sub-arrays.

The worst parameter given as the parameter of the function is the maximum number of positive integers and the best parameter is the maximum number of negative integers.

These two parameters are defined in this way to be used in comparisons within the function. Because in order for a value in the success rates array to be worst, it must be definitely less than the worst value we have determined, and must be greater than the best value we have determined to be the best. In the function, it is checked whether the given success rates array has a single element or not. If it has only one element, worst and best success rate are the same and returned. Afterwards, it is checked whether the given success rates array has 2 elements. If it has 2 elements, after some comparison operations, worst and best values are found and retrun.

If the number of elements of the array is more than 2, the index giving the middle element of the array is determined. And the array is divide into 2 sub-arrays. For the left sub-directory, the first index, which comes as a parameter to the function first index, is selected, and the last index is determined as the middle index. For the right sub-directory, the first index is determined as the index of the middle + 1 index, and the last index is the last index that comes as a parameter to the function. And recursively, the function is called repeatedly, the success rates array is divided into sub-arrays, these sub-arrays are checked, the worst and best values are found and these values are returned.



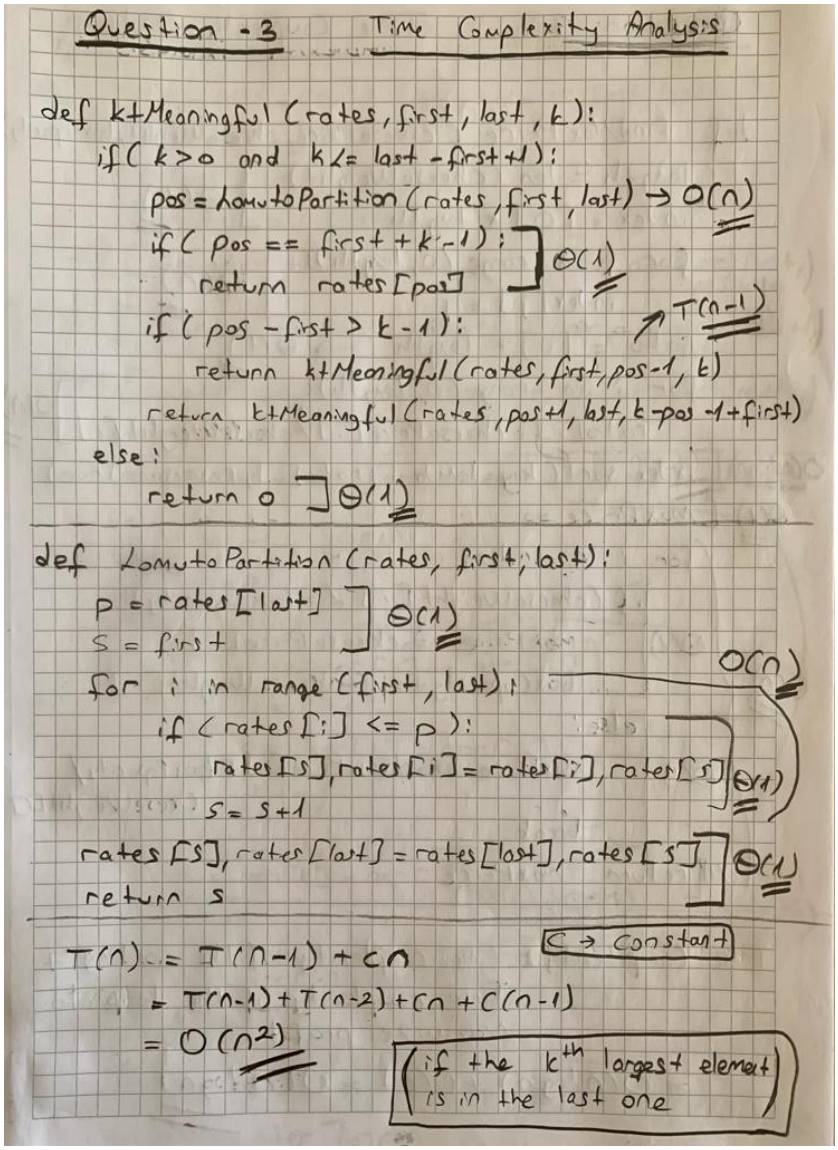


QUESTION 3

In the algorithm, I applied the decrease and conquer method using Lomuto Partition and tried to find the k'th meaningful rate in the success rate array without sorting.

I set the last element of the given array as the pivot in the LomutoPartition function. And I gave the starting index of the array to a variable s and set it as a pointer that navigates through the array. This variable will then determine the exact location of the pivot in the array.
Later, I traveled from the beginning to the end of the array with a loop and placed the elements of the array smaller than the value of the pivot I had determined, on the left of the pivot, and the larger ones on the right of the pivot, and increased the variable s by one. Finally, I swapped the pivot with the value in the index held by the variable s and placed the pivot in its correct position.

In the kthMeaningful function, I first checked whether the desired k'th index is in the array. I then determined the location of the pivot using the LomutoPartition function. If the index where the pivot is located corresponds to the desired k'th index, I returned the pivot. If the index where the pivot is located is greater than the desired k'th index, I checked the array elements to the left of the pivot, and if the index where the pivot is located is less than the desired k'th index, I performed a check again on the array elements to the right of the pivot. So recursively, I tried to find the k'th meaningful rate on the right and left parts of the pivot. When I finally found it, I returned it.

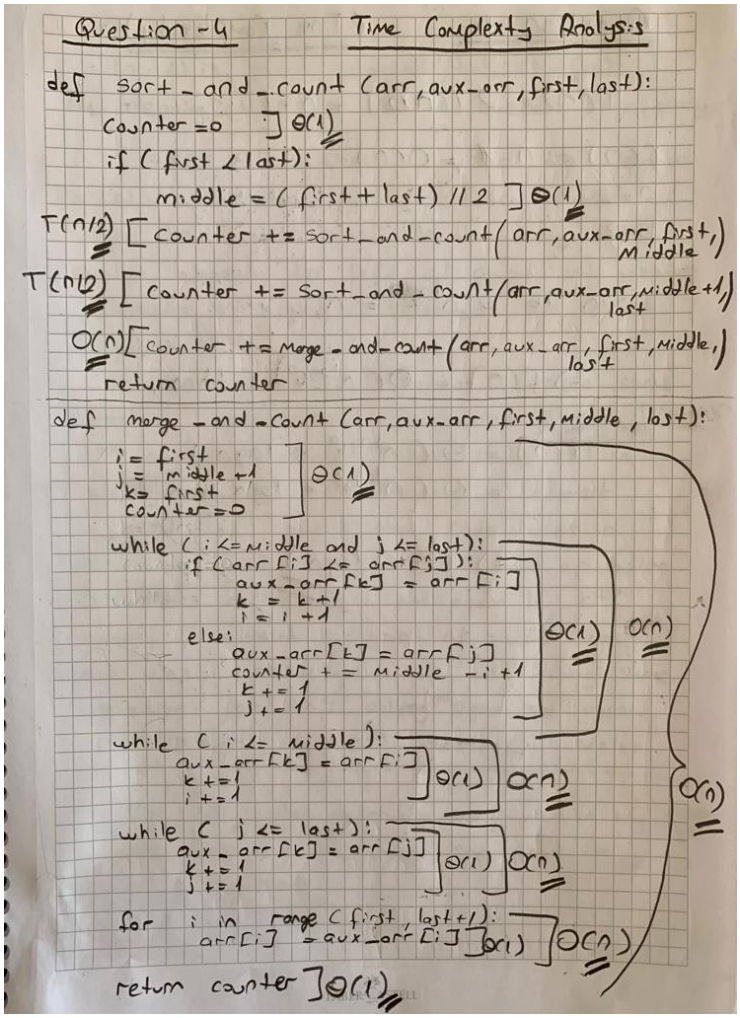


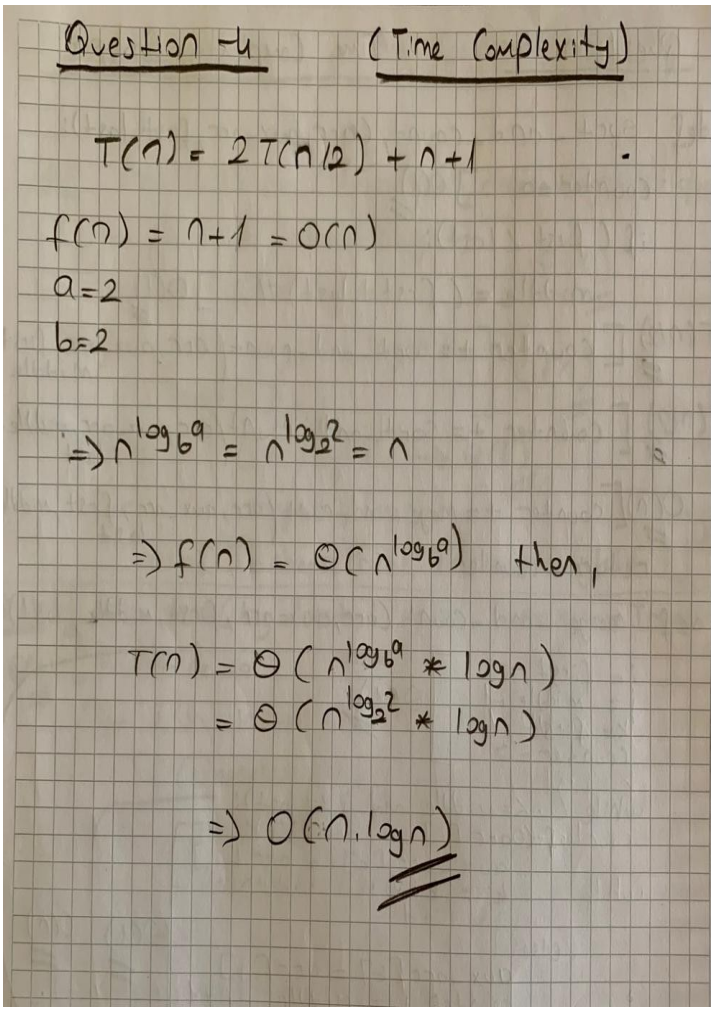
QUESTION 4

To solve the problem, I recursively divided the given array into two sub-arrays using the divide and conquer method and examined these sub-arrays and their the merging part.

In the algorithm, I divide the given array into two sub-arrays using the sort_and_count function. And I calculated the number of reverse ordered pairs of each sub-arrays and then sorted these sub-arrays in incresing order. Later, using the helper function merge_and_count, I made a comparison between these two ordered arrays and found reverse ordered pairs, combined these two arrays into the array I was using temporarily, and finally copied this temporary array to the original array. I then returned my counter, which holds the total number of reverse ordered pairs for these three cases.

The most important parts in this algorithm are when merging two ordered subarrays in the merge_and_count function and comparing the number of reverse ordered pairs, if the element of the left subarray is greater than the element of the right subarray, there is a reverse ordered pair here and there is no need to look at the remaining element of the left subarray. We reduce the complexity of the algorithm. However, if the element of the right subarray is greater than the element of the left subarray, there is no reverse ordered case here. After these comparisons, the smaller value is written into the temporary array and if the element of any sub-array ends, the elements of the other are placed directly into the temporary array. These comparisons are provided by the variables determined to navigate the indexes of the left and right subdirectories.





QUESTION 5

In the part where exponent processing is performed using the Brute-Force algorithm, a result variable is initially set as 1. If exponent is 0, the result is 1. If base is 0, the result is 0. then, in a loop, in accordance with the brute-force algorithm, from 0 to exponent is multiplied by the result base and the result is written to the result again, and the result is returned at the end.

In the part where exponent operation is performed using the Divide-Conquer algorithm, a half variable is initially set to 0. If the exponent is 0, the result is 1. If base is 0, the result is 0. The function is called repeatedly so that the given exponent value is divided by 2, and the result is written to the half value. Thus, the problem is divided into sub-parts in accordance with the divide and conquer method. Later, whether the exponent value is even or odd is checked. If even, the square of the half is returned. If odd, the square of the half is multiplied by the given base value and the result is returned.