CSC3100 Assignment 1 Report

Boshu Dou 122090091

October 6, 2023

1 Problem 1

1.1 Possible solution:

1. Solution 1:

We use double traversal to determine the disorder pairs in the input array. Basic idea is to count the number of element that is smaller than the current element.

2. Solution 2:

We use merge sort to do the job. Since merge sort has automated comparison process, all we need to do is to add a counter to that algorithm.

1.2 My solution

1. Ideas and thinking process:

First we need to use basic split method to divide the input into distinct numbers and add them into an array. Then we perform divide and conquer principle to separate the array into small arrays and so on so forth.

The core difference between pure merge sort and my algorithm is the counter to count the disorder pairs. The comparison process only occurs during the merge process. Due to the

characteristic of the merge sort, we can guarantee that every two arrays that are to be merged are well sorted. In other words, every two merging arrays are at ascending order. We define the total array at each recursion step as lst. Hence, when we discover one pair of disorder pairs at the left array index i, we will have mid (len of lst // 2) - i more disorder pairs.

For example (even length):

```
lst = [1,5,6,2,3,7]
left_arr = [1,5,6]
right_arr = [2,3,7]
mid = len(lst) // 2
```

mid == 3

When the program found the first left array element that is bigger than the right array, in this case 5 (i == 1) in the left, then it will add 3-1-2 disorder pairs (52,53).

Another example (odd length):

```
lst = [1,5,6,2,3,7,8]
left_arr = [1,5,6]
right_arr = [2,3,7,8]
mid = len(lst) // 2
```

mid == 3

When the program found the first left array element that is

bigger than the right array, in this case 5 (i == 1) in the left, then it will add 3-1-2 disorder pairs (52,53).

2. Why it is better

The time complexity of my method is O(nlogn), which is much better than the original O(n^2) method.

2 Problem 2

2.1 Possible solutions:

1. Solution 1:

We use recursion to calculate fn one by one, which will be extremely time consuming when the data is really big (O(2^n)).

2. Solution 2:

We use quick matrix multiplication to quickly calculate the fn, and we mod them during each calculation.

2.2 My solution:

1. **Ideas and thinking process:**

The main idea is to find the matrix representative of this formula, which is really easy to find. Then we need to calculate matrix multiplication quickly. Since we are only dealing with matrix to some power, I use square power to calculate this multiplication. Each time I multiply the n by 2 and calculate the square of the existing matrix. And in the end I multiply the matrix by the

remainder. This can greatly reduce the time complexity.

2. Why it is better:

My time complexity is O(logn), which is better than the original method.