GTU
Department of Computer Engineering
CSE 321 - Fall 2022
Homework - 3 Report

Asuman Sare ERGÜT
1901042657

- Question 1 -

Before specifically focusing on which technique to apply, fundamental components should be provided to the algorithm.

First we need to construct our graph, deciding on will it be constructed using adjacency list or matrix format. Due to list implementation is more stable than matrix implementation, in terms of time efficiency change, I prefer adjacency list implementation. And to explain this as Python structure, I used key-value pairs (dictionary to represent a graph). From dictionary, also a list is created, to represent adjacency list.

Since we're at the beginning of the implementation, all nodes are unvisited. So all nodes should be marked as "unvisited", by traversing the graph and assigning 0 to them.

Now, it is time to decide the technique to be sort.

a) DFS

Reasoning Behind the Algorithm:

We need to apply depth first search logic to perform topological ordering. This only can be used on DAG's.

For each unvisited nodes of the graph, we need to go it's "deep". Once a node is selected, mark it as visited and traverse it's related nodes.

Worst Case Time Complexity:

Due to we're using adjacency list as graph structure, the time complexity of creating such a list is O(m), if **m** is the number of edges in a graph. However, in the worst scenario of a complete graph with $\binom{n}{2}$ edges, the time complexity is $O(n^2)$.

b) Non-DFS

Reasoning Behind the Algorithm:

Another method, that also brings solution to the graphs that is not a DAG is Kahn's algorithm. Main approach is that: A vertex with an in-degree of zero and no incoming edges is always the first vertex in topological sorting.

The term in-degree: number of incoming edges
For each vertex in the graph, calculate the number of incoming edges. Again, all nodes should be marked as unvisited. Select every vertex with in-degree of 0, add it to a queue, and then remove it from the queue. If the in-degree of nearby nodes is decreased to zero, add it to the queue and keep doing so until it is empty.

The topological sort is not possible for the provided graph if the number of visited nodes does not equal the number of nodes in the graph.

Worst Case Time Complexity:
V represent the vertices amount and E represents the edge amount. Time complexity will mi at most O(V+E).

- Question 2 -

Reasoning Behind the Algorithm:

When a problem is hard to solve, or has the big time complexity, dividing that problem into small parts is one of the most applied and useful methods. Exponentiation is one of those problems. As a solution to optimize time complexity from linear time to logarithmic time, It is possible to divide multiplication into smaller parts, each of which is a multiple of 2.

But why 2? Any number can be constructed from powers of two. (I think that's because 2 is the smallest prime number). Once we know the number n, instead of multiplying n times a, we can express n in a different way like binary and instead of multiplying-only, we need to use both multiplication and summation. By doing this, we can get O(logn) complexity.

Worst Case Time Complexity:

As the other “divide by 2” algorithm’s results, our algoritm has the time complexity of O(logn), where n is the exponent.

- Question 3 -

Reasoning Behind the Algorithm:

How we’re, as human, solving a sudoku? First take a look at to all rows and columns, starting from 0th…8th (9 row and 9 column), and decide is there any ready-to-fill cells exist. If yes, fill it. If not, think the possibilities, which number I should write to that cell.

Sometimes auto-generated or human-written sudoku games may be unsolvable. Before start to solving the game, we first need to control whether it is solvable or not. After that fill the ready-to-fill cells if exists. After that by making possibility calculations, fill the cells.

By looking rows and columns as human eye is easy, from top to bottom or left to right. It’s equivalent on computer is “enumarating” the rows and columns.

Worst Case Time Complexity:

Algorithm contains lots of recursive steps, but most of them are inside “a row” or “a column” or “a 3x3 mini board”. But overall, it takes:

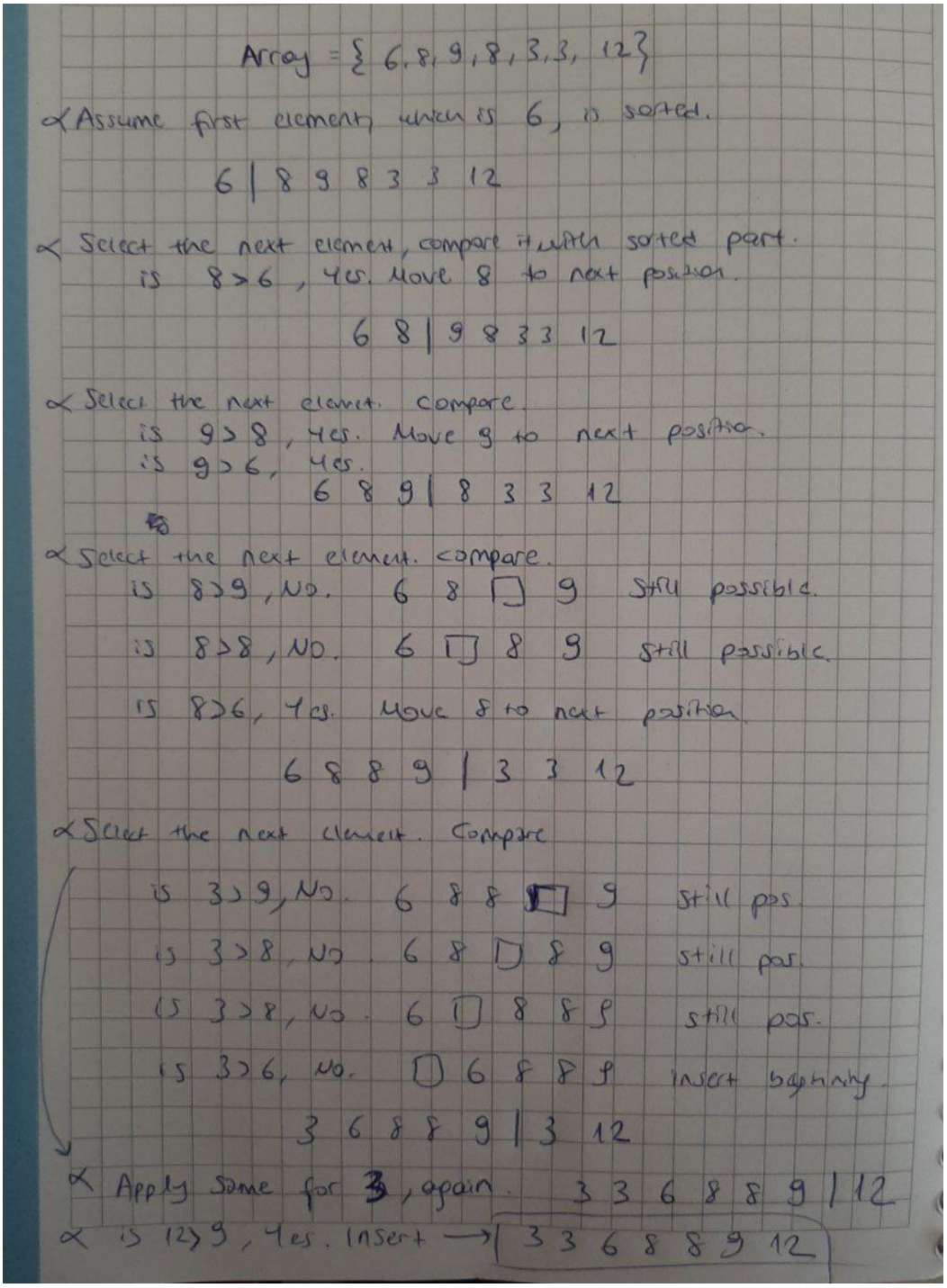
$$T(n) = T(n-1) + \log n \Rightarrow O(n \log n)$$

where logn comes from we’re just looking almost half of the board(3x3 part of the 9x9 board)

- Question 4 -

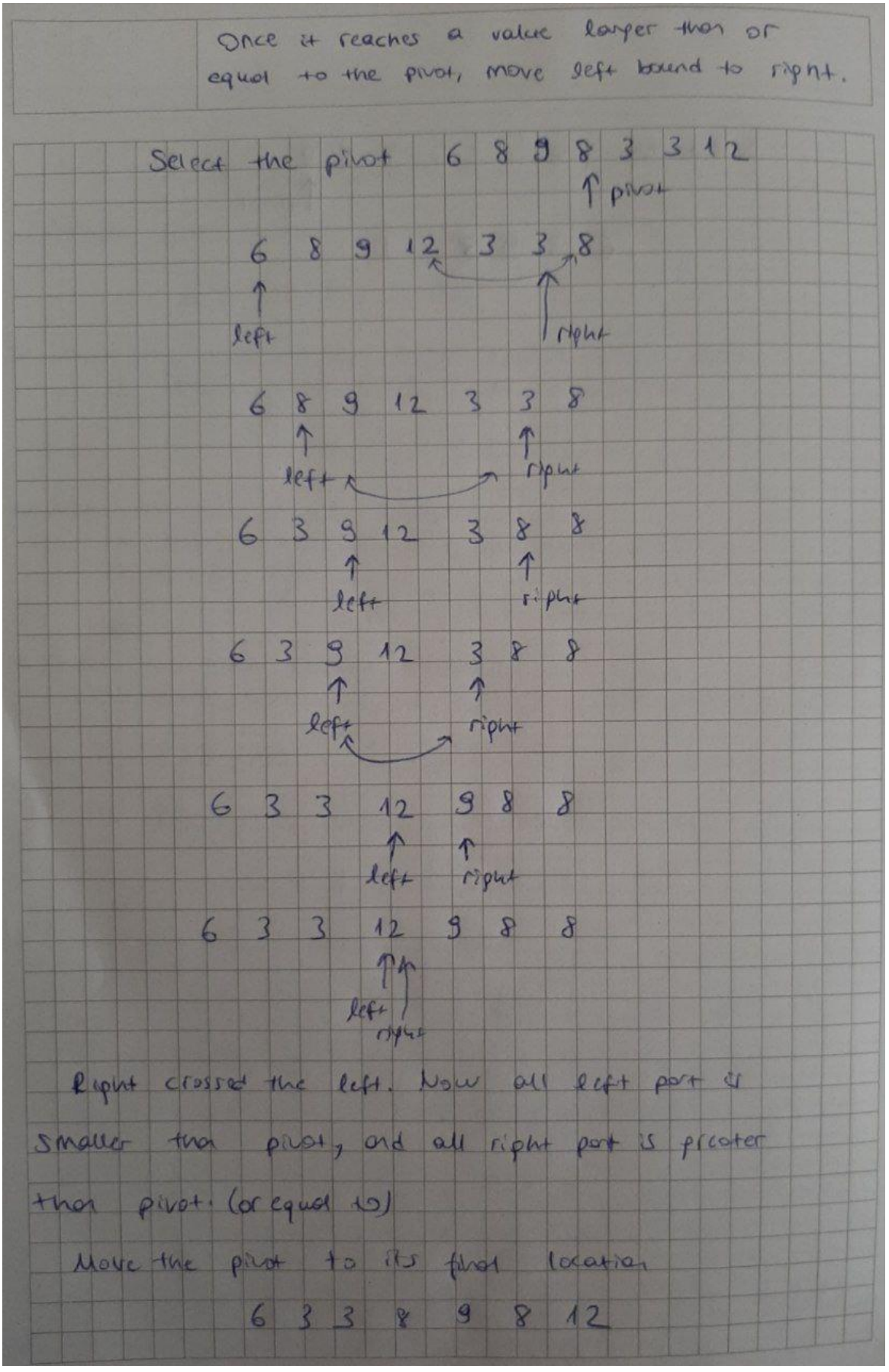
Sort the array {6, 8, 9, 8, 3, 3, 12} in ascending order

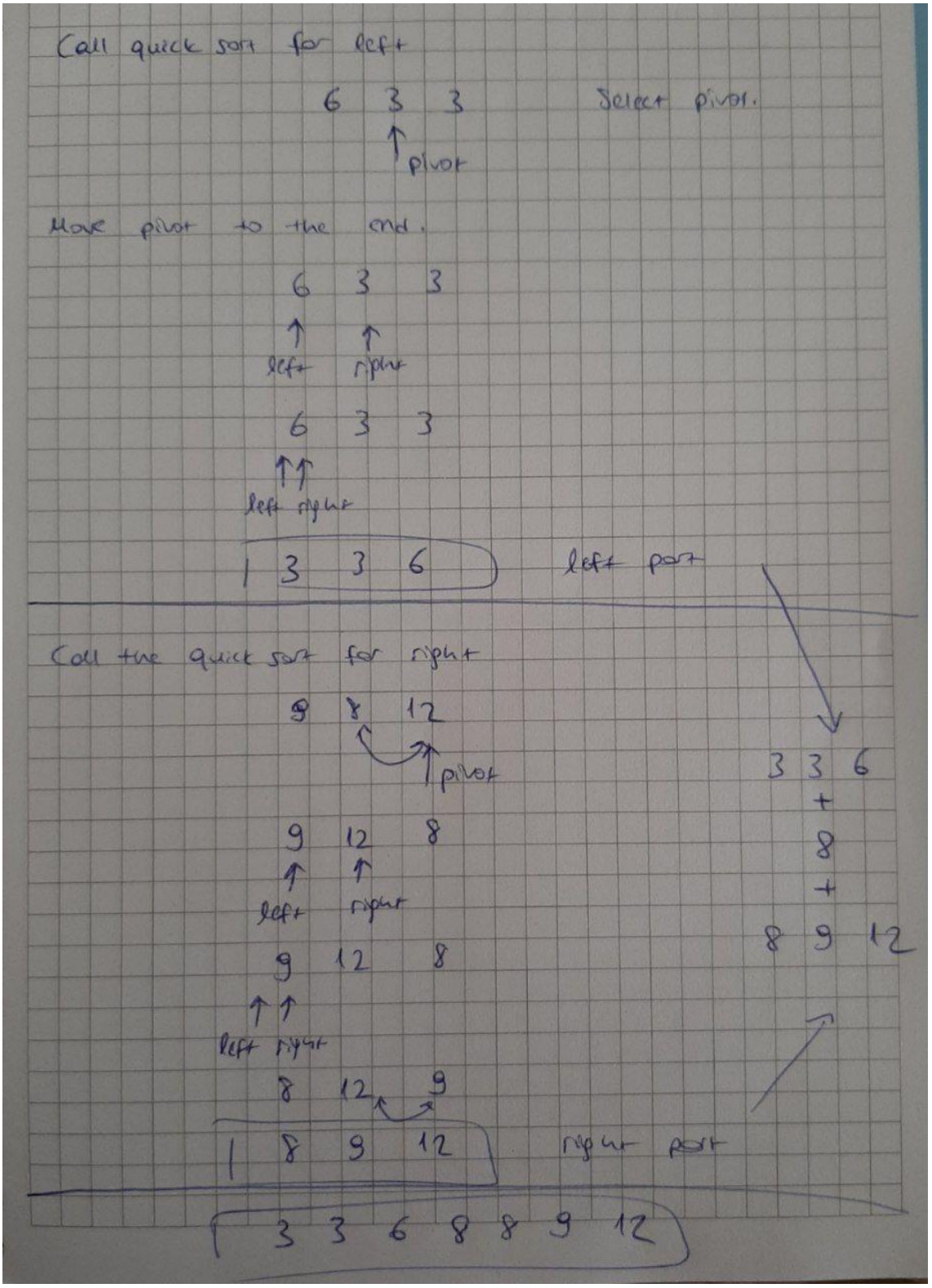
Insertion Sort:



Does insertion sort stable? Yes. Relative order is preserved due to if element is not bigger than compared one (even in equal), it's place will be same.

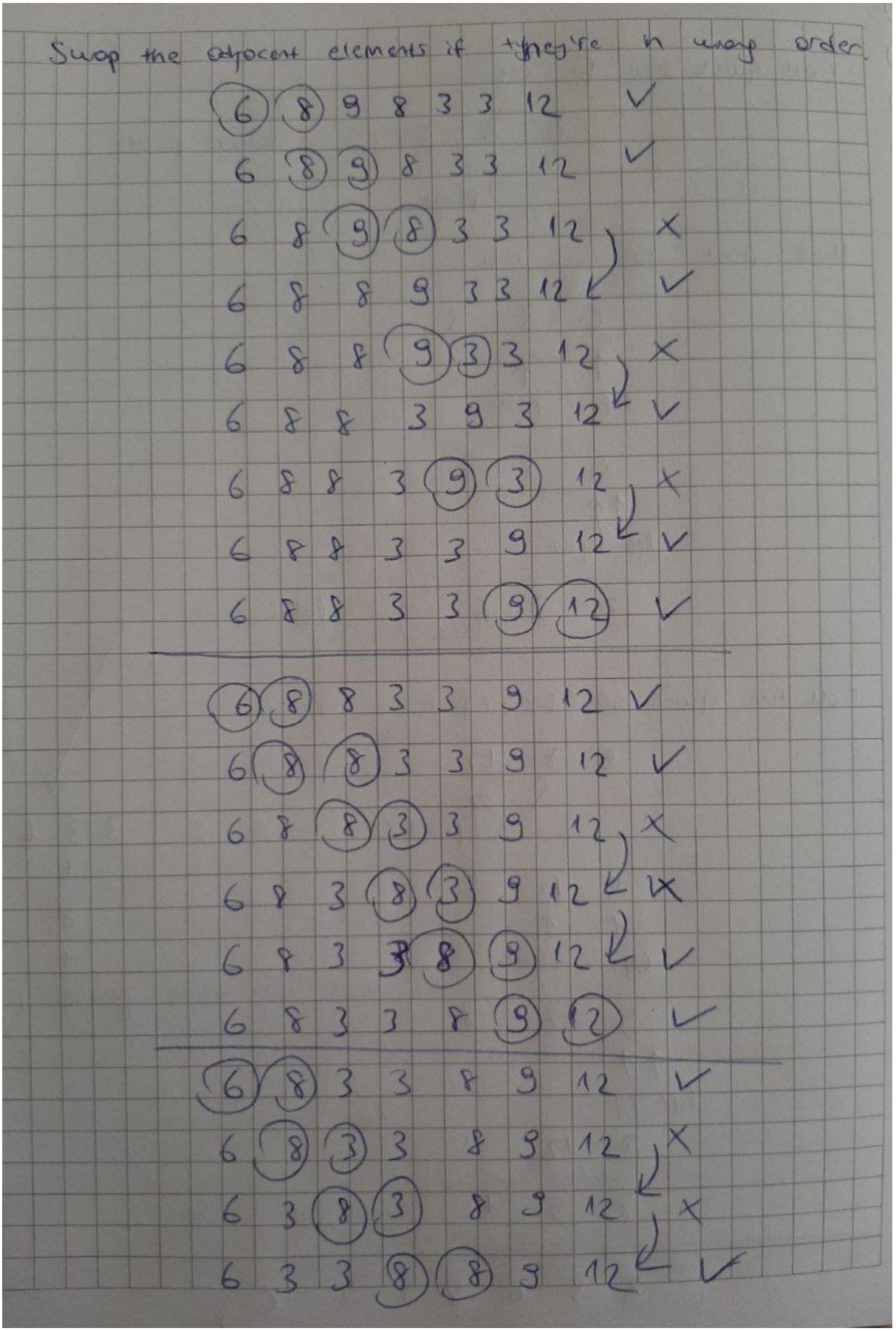
Quick Sort:

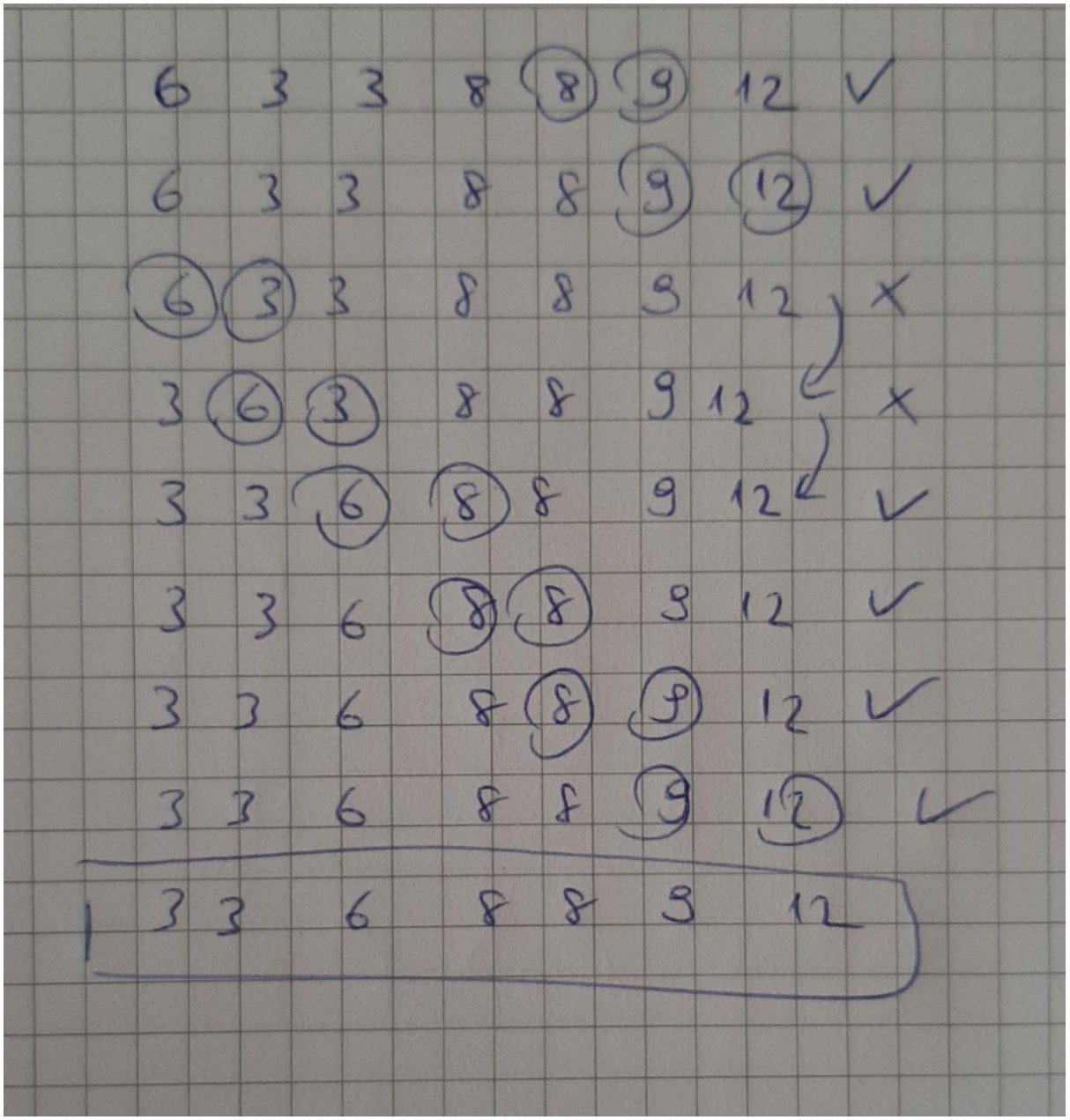

Once it reaches a value larger than or equal to the pivot, move left bound to right.
Select the pivot 6 8 9 8 3 3 12
pivot
6 8 9 12 3 3 8
left
right
6 8 9 12 3 3 8
left
right
6 3 9 12 3 8 8
left
right
6 3 9 12 3 8 8
left
right
6 3 3 12 9 8 8
left
right
6 3 3 12 9 8 8
left
right
Right crossed the left. Now all left part is smaller than pivot, and all right part is greater than pivot. (or equal to)
Move the pivot to its final location
6 3 3 8 9 8 12



Does quick sort stable? No. According to pivot and it's place, every element's position can be change. We couldn't guarantee not to change.

Bubble Sort:


Swap the adjacent elements if they're in wrong order.
6 8 9 8 3 3 12 ✓
6 8 9 8 3 3 12 ✓
6 8 9 8 3 3 12 ✗
6 8 8 9 3 3 12 ✓
6 8 8 9 3 3 12 ✗
6 8 8 3 9 3 12 ✓
6 8 8 3 9 3 12 ✗
6 8 8 3 3 9 12 ✓
6 8 8 3 3 9 12 ✓
6 8 8 3 3 9 12 ✓
6 8 8 3 3 9 12 ✗
6 8 3 8 3 9 12 ✗
6 8 3 3 8 9 12 ✓
6 8 3 3 8 9 12 ✓
6 8 3 3 8 9 12 ✓
6 3 3 8 8 9 12 ✓



Does bubble sort stable? Yes. Relative order is preserved due to if element is not bigger than compared one (even in equal), it's place will be same.

- Question 5 -

a) Relation between brute force and exhaustive search

- Both are searching algorithms.
- The practice of looking for every conceivable answer is one of both algorithms' shared characteristics.
- Since brute force is an algorithm that is used when a problem has a finite size, it differs from exhaustive search in this regard. Exhaustive search, on the other hand, is a method of dealing with complex issues.
- In terms of time-consuming, exhaustive search has an advantage on brute force search.

b) Caesar's Cipher and AES

Caesar's Cipher:

- The main idea behind Caesar's Cipher is using a "fixed key" to encrypt the text (or anything). Fixed key is applied on a specific pattern, i.e. a letter with a predetermined number of places is used to replace each letter in a given text.

```
Text : ATTACKATONCE
Shift: 4
Cipher: EXXEGOEXSRGI
```

Is it vulnerable to brute force attacks?

- Not much. because when this fixed key is find, all text will be decrypted easily. Finding this fixed key is not hard, also. After a few trying, a reasonable pattern can be obtained.

AES:

- AES stands for Advanced Encryption Standard, is an example of symmetric block cipher, with having certain dissimilarities. Although it is hard to implement, it is a much stronger than DES type encryptings. Performs operations on bytes of data rather than in bits. In encryption, think of each block as a 16-byte (4 by 4 bytes = 128) grid set up in a column-major configuration.

```
[ b0 | b4 | b8 | b12 |
| b1 | b5 | b9 | b13 |
| b2 | b6 | b10| b14 |
| b3 | b7 | b11| b15 ]
```

Is it vulnerable to brute force attacks?

- Working on bytes mades it much stronger. Even with today's technology, it is impossible to break the AES algorithm, despite the fact that it has been around for 20 years.

c) Why does the naive solution to primality testing (for input n, checking if $x \in$ {2, 3, ..., n − 1} divides n) grow exponentially?

Because the number of divider is also growing, while the number is growing. Bigger numbers are constructed from multiplication of prime numbers, with a growing rate. For example,

2^1 x 3^1 x 4^1 = 9

When we make 1-> 2

2^2 x 3^2 x 4^2 = 29

NOTE: All driver functions are working. If there occurs a problem in custom input, you can try to edit the “driver” parts and run it. It may be because inputting differences.