ASSIGNMENT 1 - Exercise 2

Consider the following algorithm *ALGO-X(A)* that takes an arrays A of numbers, possibly negative, sorted in non-decreasing order.

```python
def algox(A):
    """ ALGO-X(A) algorithm provided by exercise 2 pseudo-code.

    :param A: list[int] incresingly ordered list of numbers in Z set
    :return: ordered list
    """
    B = []
    n = len(A)

    for i in range(n):
        B.append(A[i]*A[i])

    for i in range(1, n):
        j = i - 1
        v = B[i]
        while j >= 0 and B[j] > v:
            B[j + 1] = B[j]
            j = j - 1
        B[j + 1] = v

    return B
```

Question 1

Explain what Algo-X does. Do not simply paraphrase the code. Instead, explain the high level semantics, independent of the code.

Assumptions are: a given array A with non-decreasing positive and negative numbers.

An empty array goes through a for-loop that fills it with the squares of all the numbers from array A.

Then, another for-loop sorts the array by incrementally swapping elements in increasing order, i.e. starting with sorting the first couple of elements and increasing the sorting pool until reaching all the array.

Finally, it returns the new sorted array B, so overall the functionality was to make the values with small squares which have a preceding bigger square, to go swap it to the very first index available; hence the array will still remain sorted because it was a starting assumption.

Question 2

Analyze the complexity of Algo-X in the best and worst case. Justify your answer by clearly describing a best and worst-case input of size n, as well as the behavior of the algorithm in each case.

The best case is to never go through the while loop, hence always having a B[j] > v, hence having already an array with only positive ordered numbers, so all the squares all already kept sorted. Complexity is O(n).

An example of best case input with 4 elements: [1, 2, 3, 4]

The worst case is to have only negative numbers because it will be resulting in having to swap all the squares back to the first positions because the input array will be sorted in decreasing order while we want it in increasing order and to do that we use a swap function that will always trigger for every number. Complexity is $O(n^2)$; because we loop through the array length ' n ' and for swap an element smaller than another, we swap (possibly) ' n ' times, so $n * n$.

An example of worst case input with 4 elements: [-4, -3, -2, -1]

Question 3

Write an algorithm called Better-Algo-X that does exactly the same thing as Algo-X in O(n) time. Analyze the complexity of Better-Algo-X.

DONE. Linear time implemented from an inspiration derived from merge sort implementation.

```python
def better_algox(A: list) -> int:
    """ Sort the square of an already incresingly sorted list of numbers in Z

    :param A: list[int] incresingly ordered list of numbers in Z set
    :return: ordered list
    """
    # get squared elements in list -> O(n)
    length = len(A)
    for i in range(length):
        A[i] = A[i] ** 2

    # calculate where's the minimum element and its index -> O(n)
    orderedlist = []
    k = 1
    min_index = 0
    min_el = A[min_index]
    while k < length:
        if A[k] < min_el:
            min_el = A[k]
            min_index = k
        k += 1

    # append all elements in incresing order -> O(n)
    tot_el = 0
    k = min_index
    j = min_index + 1
    while tot_el < length:
        if A[k] < A[j]:
            orderedlist.append(A[k])
            k = (k-1) % length
        else:
            orderedlist.append(A[j])
            j = (j+1) % length
        tot_el += 1

    return orderedlist
```