Algorithim Assignment 1

Omar Ahmed Alhozimi عمر احمد الحزيمي	443014015

Part 1:

Insertion sort:

1-

```
insertion_sort(array : list of sortable items, n : length
of list)
    i ← 1
    while i < n
        j ← i
        while j > 0 and array[j - 1] > array[j]
            swap array[j - 1] and array[j]
            j ← j - 1
        end while
        i ← i + 1
    end while
```

2-

The Insertion sort assign the value of index j for the second element, and the value of index i for the first element, then compare them with each other and swap if j is bigger than i to make the numbers in the correct position, and then it repeat the process with all of the array indexes until it all are sorted.

3- Theoretical analysis of Time complexity for insertion sort:

Best case: insertion sort in the best case takes O(n), that happens when the array is already sorted.

Average case: insertion sort in the average case takes O(n^2).

Worst case: insertion sort in the worst case takes O(n^2) and that happened when the array is in reverse order.

Merge Sort:

1-

```
public static void mergeSort(int[]
arr) {
    if (arr.length <= 1) {
        return;
    }
    int mid = arr.length / 2;
    int[] left =
Arrays.copyOfRange(arr, 0, mid);
    int[] right =
Arrays.copyOfRange(arr, mid,
arr.length);
    mergeSort(left);
    mergeSort(right);
    merge(arr, left, right);
}
```

2-

The merge sort at first divides the array of element into half which take O(1).

Then after dividing all of the array element until it reach each of size 1, it conquer the element array each at his own.

And then at the end merge sort combine these conquered element into the array to get that sorted array.

3- Theoretical analysis of Time complexity for Merge Sort:

- In the best case: the merge sort takes O(n logn) we got this because it take O(logn) for conquering them into 2 sub half n/2, and O(n) for combining all of the array elements.

- In the average case: The merge sort takes O(n logn).

- In the worst case: The merge sort takes O(n logn) also.

Insertion sort VS Merge sort(theoretically):

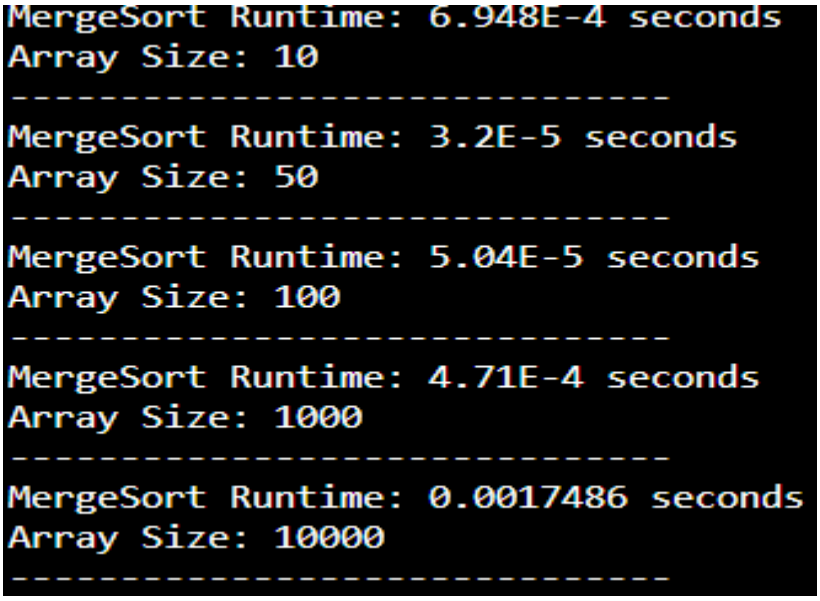
In the implementation the merge sort is more complex than insertion sort, but the insertion is better in the best case since it take O(n) and that happens when the array is already sorted, the implementation of merge sort is harder but its better than the insertion sort in the average and worst case since it takes only O(n logn).

Part 2:

The run time for the insertion sort:

```
Array Size: 10
InsertionSort Runtime: 6.959E-4 seconds
------------------------------
Array Size: 50
InsertionSort Runtime: 1.06E-5 seconds
------------------------------
Array Size: 100
InsertionSort Runtime: 3.71E-5 seconds
------------------------------
Array Size: 1000
InsertionSort Runtime: 0.0025948 seconds
------------------------------
Array Size: 10000
InsertionSort Runtime: 0.0476249 seconds
------------------------------
```

The run time for the Merge Sort:



In Conclusion :

By analyzing the result we can see that the big difference happens when we put a large input size, and we have said theoretically that the insertion sort is only better at the best case, otherwise the merge sort is better since we are getting O(nlogn), and as we implemented the code and we got the run time we have seen that our theory is right and the merge sort is faster than the insertion and also better when we compare them theoretically and as an real implementation.