Sorting Algorithms Complexity

17/5/2023

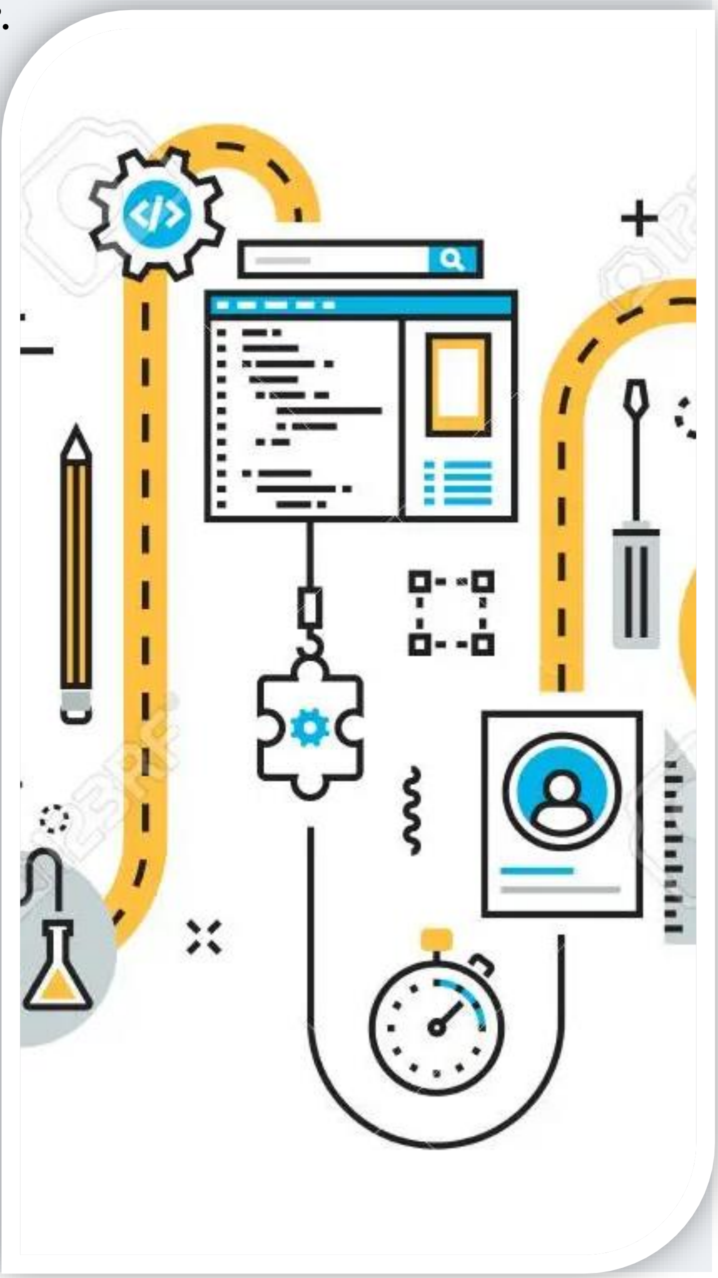
Dr.Imad Jawhar
Mostafa Shmaisani
10120028

Introduction

Software companies nowadays are competing in providing better and more effiecient softwares. One of the most important aspects in this competition is time complexity. Time complexity is important in all the programming aspects were with a large amount of data, better algorithms will save days from the bad ones. For example, in software development, companies will try it's best to save time, waiting for an output for a one second, better than from 1 min, where 60 sec will equal one 1 min and the 60 min will equal 1hour. So this will save hours for the user.

Another example is in CyberSecurity, where the password entropy project calculates the time the password needs to be cracked through different algorithms, the more time it takes, the more powerful the password is.

So in this research, I am going to compare different sorting algorithms with their time complexity to check which is the best for sorting a big amount of data.



Source Code

All the source code is provided in GitHub, but here there is a brief summary about the code:
https://github.com/shmaisanimostafa/Sorting-Algorithms-Complexity

Used Classes:

```
import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;
```

Bigging Part:

Here the program starts, at first it welcomes the developer, then it shows him/her that all the algorithms are working correctly through trying them on 10 unsorted elements.

```
public class App {
    public static void main(String[] args) {

        System.out.println("\n \n"); //Empty Lines

////////////////////////////////////////////////////////////////////////////////////////////
//Introduction
////////////////////////////////////////////////////////////////////////////////////////////
        System.out.println("Welcome to the sorting algorithms time comlexity reseatch!\n");

////////////////////////////////////////////////////////////////////////////////////////////
//Try the algorithms on an array to make sure all are sorting
////////////////////////////////////////////////////////////////////////////////////////////
        System.out.println("First we are going to test the algorithms by giving them 10
elements array input. \n");
        System.out.println("The UnSorted 10 elemtents array is: ");
        int[] arr = {6, 5, 2, 9, 7, 8, 3, 10, 4, 1};
        //blacblac(arr);
        System.out.println(); //Empty Line

        //Second Array is used to copy the unsorted array, after sorting it in each
algorithm, instead of creating same array for every algorithm.
        int[] arr2 = Arrays.copyOf(arr, arr.length);

        //Apply the array on all different algorithms
        insertion(arr, arr2);
        merge(arr, arr2);
        heap(arr, arr2);
        quick(arr, arr2);
        bubble(arr, arr2);
        selection(arr, arr2);
        count(arr, arr2);
```

Second Part:

In this part we generate a random array, where the user specifies it's length, then the user choose by which algorithm he/she wants to sort the data with. He/she may choose more than. Each one will show the Input Array (Unsorted), Output Array (Sorted), and the time it took to sort (Duration).

```
///////////////////////////////////////////////////////////////////////////////////////////
//Try a random generated array on different algorithms
///////////////////////////////////////////////////////////////////////////////////////////
        algoWork();
        System.out.println("\n \n \n"); //Empty Lines
    }
```

algoWork() Method:

```
   public static void algoWork() {
    System.out.println("Second we should generate a random array then we apply it on each 
algorithm to compare their time duration.");
        int[] randomArray = arrayGenerator();
        int[] randomArray2 = Arrays.copyOf(randomArray, randomArray.length);
        boolean tryAlgo = true;


        Scanner scanner = new Scanner(System.in);
        while (tryAlgo) {
            System.out.println("\nSelect on which algorithm you want to apply the array, 
choose by number (default is all):");
            System.out.println("Note to choose the default choose any other number");
            System.out.println("1)Insertion\n2)Merge\n3)Quick\n4)Counting\n5)Heap\n6)Selecti
on\n7)Bubble\n");
            int nub = scanner.nextInt();
            scanner.nextLine(); // Consume the remaining newline character (Scanner Problem)
            switch (nub) {
                case 1 :
                    {
                    insertion(randomArray, randomArray2);
                break;
                }

                case 2 :
                {
                    merge(randomArray, randomArray2);
                    break;
                }
                case 3 :
                {
                    quick(randomArray, randomArray2);
                    break;
                }
                case 4 :
                {
                    count(randomArray, randomArray2);
                    break;
                }
                case 5 :
                {
                    heap(randomArray, randomArray2);
                    break;
                }
                case 6 :
                {
                    selection(randomArray, randomArray2);
                    break;
                }
                case 7 :
                {
                   bubble(randomArray, randomArray2);
                    break;
                }
                default:
                {
                    insertion(randomArray, randomArray2);
                    merge(randomArray, randomArray2);
                    quick(randomArray, randomArray2);
                    count(randomArray, randomArray2);
                    heap(randomArray, randomArray2);
                    selection(randomArray, randomArray2);
                    bubble(randomArray, randomArray2);
                    break;
                }

            }

            System.out.println("Do you want to try on another algorithm? (Y/N)");
            String a = scanner.nextLine();
            if (!a.equalsIgnoreCase("y")) {
              tryAlgo = false;
            }

        }
        System.out.println("\nHave a Good Day :)");
}
```

printArray Method:

```
    public static void printArray(int arr[])
{
    int n = arr.length;
    for (int i = 0; i < n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
}
```

arrayGenerator() Method:

```
    public static int[] arrayGenerator() {
    Scanner scanner = new Scanner(System.in);
        Random random = new Random();

        System.out.print("Enter the number of random elements (make sure positive number)
:  ");
        int n = scanner.nextInt();
        int[] randomArray = new int[n];
        for (int i = 0; i < n; i++) {
            randomArray[i] = random.nextInt(98) + 1;
        }
        System.out.println("\nThe numbers are between 1 and 99 \nThe random generated array
is: ");
        printArray(randomArray);
        return randomArray;
}
```

There are also the Algorithms methods and also a method for each algorithm that shows the input and output array for each algorithm (before using the algorithm and after) and the duration it took to sort, here is an example for insertion sort:

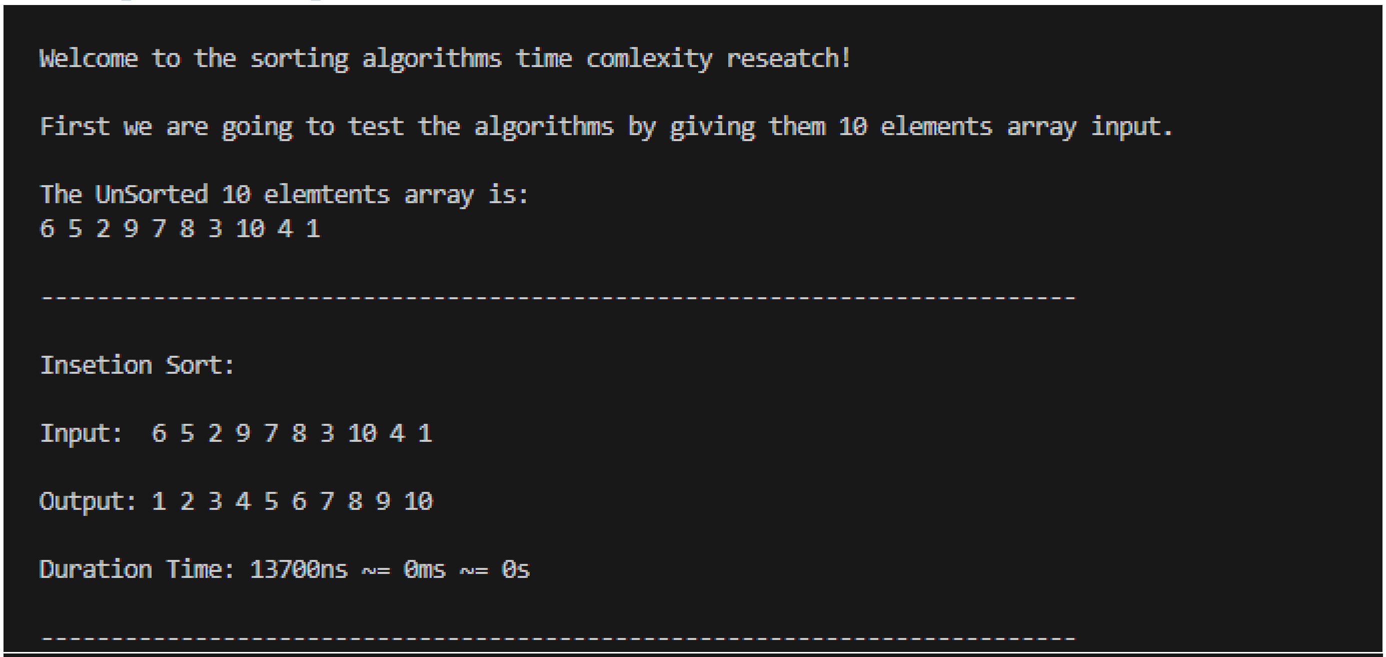
The Normal Algorithms

```
    public static void Insertionsort(int arr[])
{
    int n = arr.length;

    for (int i = 1; i < n; ++i) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

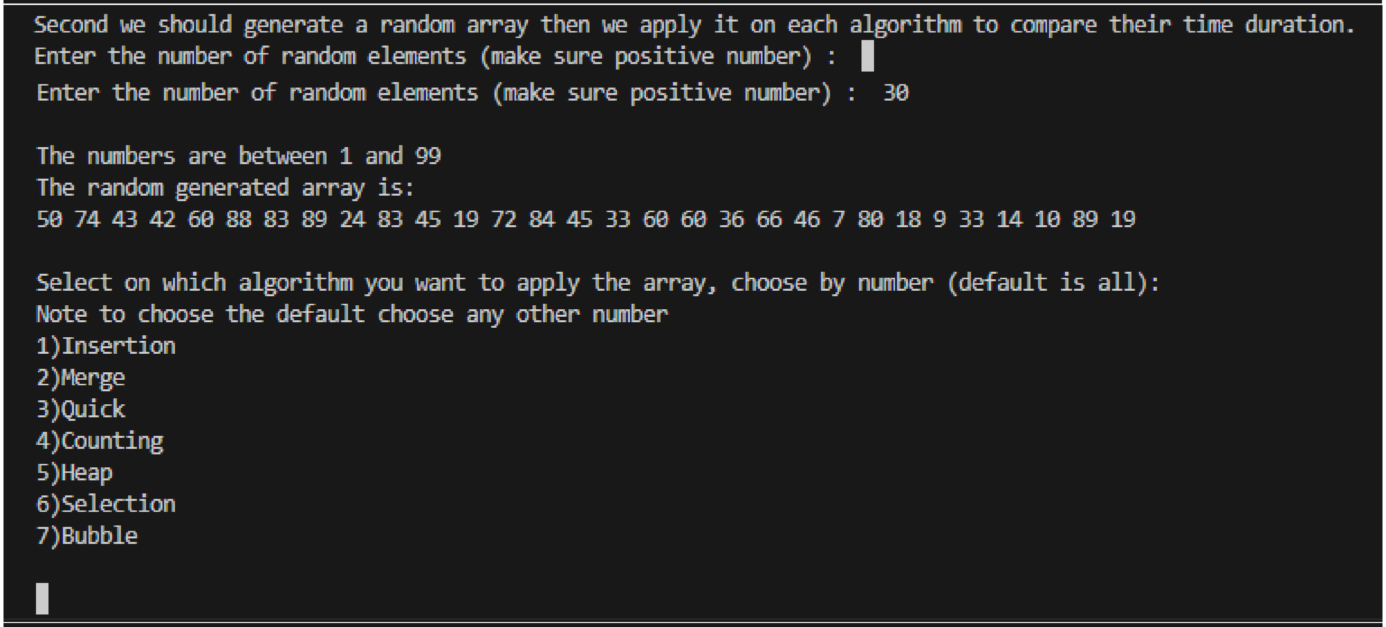
The Algorithm with input, output, and duration:

```
   public static void insertion(int[] randomArray,int[] randomArray2) {
System.out.println("-------------------------------------------------------------------------
--");

    System.out.println("\nInsetion Sort:  ");
                System.out.print("\nInput:  ");
                printArray(randomArray);

                long timerStart = System.nanoTime();
                Insertionsort(randomArray2); //Applying Insetion Sort
                long timerEnd = System.nanoTime();
                long timer = (timerEnd - timerStart);

                System.out.print("\nOutput: ");
                printArray(randomArray2);
                randomArray2 = Arrays.copyOf(randomArray, randomArray.length);
                System.out.println("\nDuration Time: " + timer + "ns ~= " + timer /1000000 +
"ms ~= " + timer /1000000000 + "s");
                System.out.println();//Empty Line
}
```

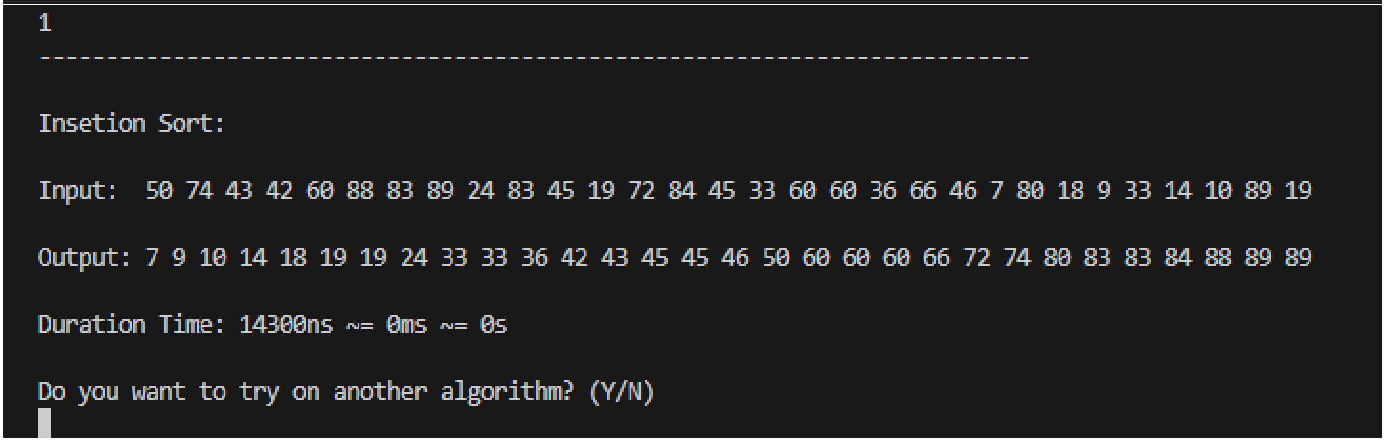
Sample Output



```
Welcome to the sorting algorithms time comlexity reseatch!

First we are going to test the algorithms by giving them 10 elements array input.

The UnSorted 10 elemtents array is:
6 5 2 9 7 8 3 10 4 1

--------------------------------------------------------------------------

Insetion Sort:

Input:  6 5 2 9 7 8 3 10 4 1

Output: 1 2 3 4 5 6 7 8 9 10

Duration Time: 13700ns ~= 0ms ~= 0s

--------------------------------------------------------------------------
```



```
Second we should generate a random array then we apply it on each algorithm to compare their time duration.
Enter the number of random elements (make sure positive number) :
Enter the number of random elements (make sure positive number) :  30

The numbers are between 1 and 99
The random generated array is:
50 74 43 42 60 88 83 89 24 83 45 19 72 84 45 33 60 60 36 66 46 7 80 18 9 33 14 10 89 19

Select on which algorithm you want to apply the array, choose by number (default is all):
Note to choose the default choose any other number
1)Insertion
2)Merge
3)Quick
4)Counting
5)Heap
6)Selection
7)Bubble
```

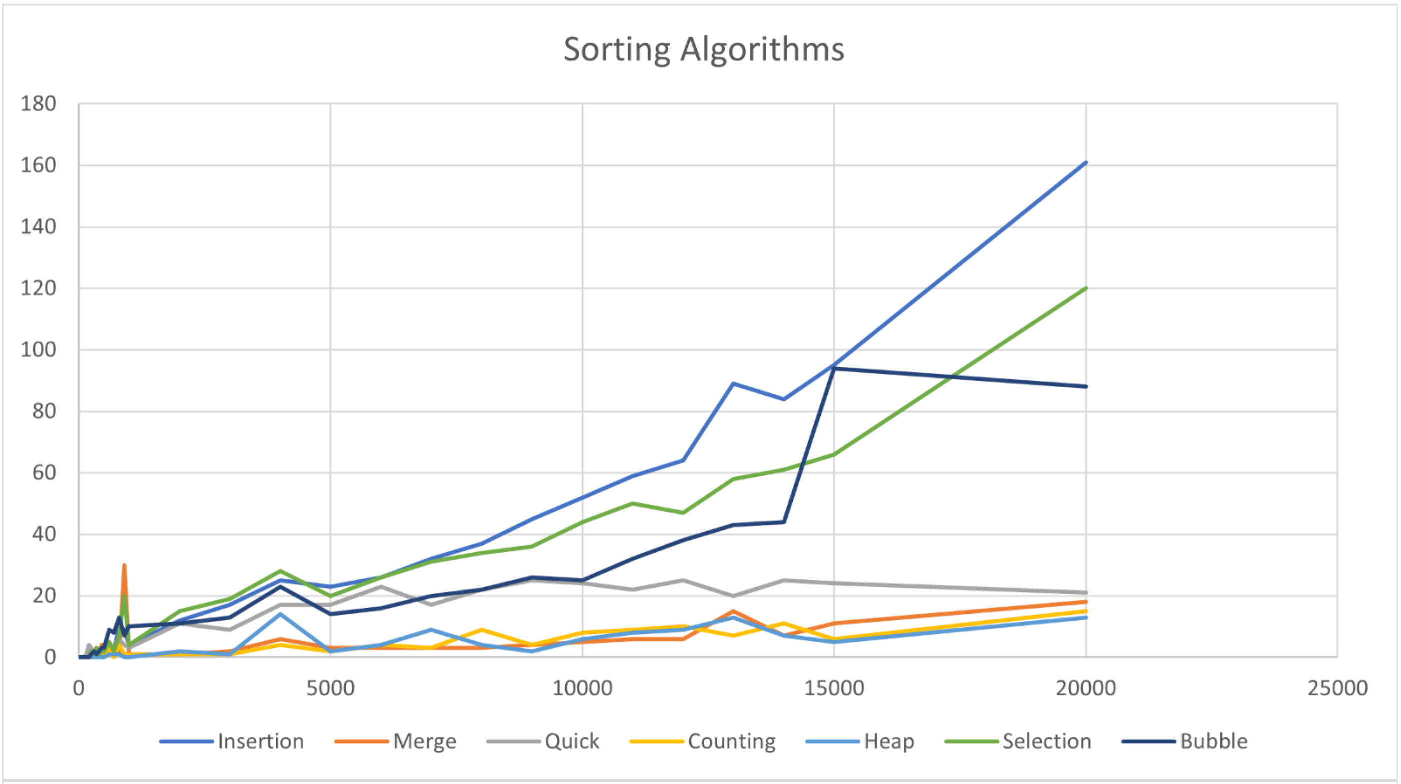


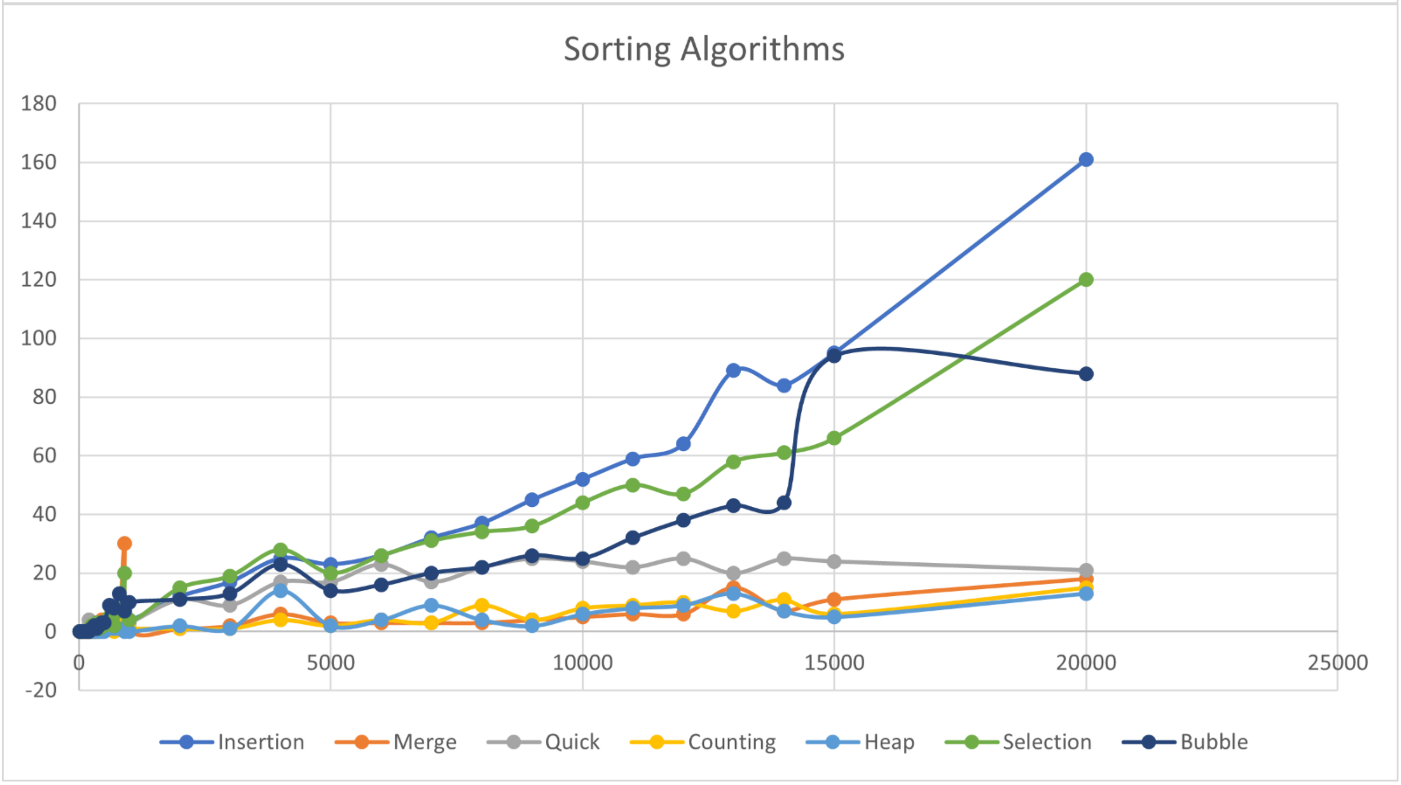
```
1
-----------------------------------------------------------------------

Insetion Sort:

Input:  50 74 43 42 60 88 83 89 24 83 45 19 72 84 45 33 60 60 36 66 46 7 80 18 9 33 14 10 89 19

Output: 7 9 10 14 18 19 19 24 33 33 36 42 43 45 45 46 50 60 60 60 66 72 74 80 83 83 84 88 89 89

Duration Time: 14300ns ~= 0ms ~= 0s

Do you want to try on another algorithm? (Y/N)
```

Analysis

Table 1 This table shows how much time(ms) did every algorithm took to sort with respect to the number of random elements.

n\Algorithm	Insertion	Merge	Quick	Counting	Heap	Selection	Bubble
10	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0
150	0	0	1	0	0	0	0
200	0	0	4	0	0	0	0
250	0	0	3	0	0	2	1
300	0	0	1	0	0	1	2
350	1	0	3	2	0	3	1
400	1	1	2	0	0	1	2
450	1	4	2	1	0	2	3
500	2	4	2	1	0	2	3
600	2	1	5	3	1	5	9
700	2	2	2	0	1	2	8
800	10	1	6	4	1	7	13
900	9	30	4	1	0	20	7
1000	4	1	3	1	0	4	10
2000	12	1	11	1	2	15	11
3000	17	2	9	1	1	19	13
4000	25	6	17	4	14	28	23
5000	23	3	17	2	2	20	14
6000	26	3	23	4	4	26	16
7000	32	3	17	3	9	31	20
8000	37	3	22	9	4	34	22
9000	45	4	25	4	2	36	26
10000	52	5	24	8	6	44	25
11000	59	6	22	9	8	50	32
12000	64	6	25	10	9	47	38
13000	89	15	20	7	13	58	43
14000	84	7	25	11	7	61	44
15000	95	11	24	6	5	66	94
20000	161	18	21	15	13	120	88


Sorting Algorithms
180
160
140
120
100
80
60
40
20
0
0
5000
10000
15000
20000
25000
Insertion
Merge
Quick
Counting
Heap
Selection
Bubble


Sorting Algorithms
180
160
140
120
100
80
60
40
20
0
-20
0
5000
10000
15000
20000
25000
Insertion
Merge
Quick
Counting
Heap
Selection
Bubble

It’s clear that Heap, Merge, and Counting sort algorithms are the best for sorting data.

*We can see that Insertion sort takes the most time where it have exponentiol complexity O(n^2) Then Selection Sort and Bubble Sort comes also with complexity of O(n^2). These three offcourse are less used when time is taken into consideration. Here Quick Sort comes with O(n*Log(n)) complexity, it works bad with already sorted arrays to be O(n^2) and very will with unsorted array to reach sometimes complexity of O(log(n)), so it is used in special cases. At the end, the last 3 Heap, Counting, and Merge Sort Algorithms work the best in sorting that data with a nearly stable complexity of O(nlog(n)).*

Conclusion

The best sorting algorithm depends on it’s use, but Heap, Merge, and Counting Sort are the best for 90% of the cases, with the priority for Heap, for each many useful methods (insert, get max, ...).

Thanks for reading my research!