Hacettepe University

Computer Engineering Department

BM204 Software Practicum II - 2022 Spring

Programming Assignment 1

March 9, 2022

Student name:
Berk BUBUŞ

Student Number:
b21945939

1 Problem Definition

Testing 4 sorting algorithms (insertion, merge, pigeonhole, counting) and getting some results.

2 Solution Implementation

I coded pseudocodes one-by-one. After that I did a bit upgrades. These upgrade are not about Sorting Algorithm, especially about Array usage.

2.1 Insertion Sort

```
1   public void Insertion(int[] Array){
2       int key,j;
3       for(int i = 1;i< Array.length;i++){
4           key = Array[i];
5           j = i - 1;
6           while(j>-1 && Array[j]>key){
7               Array[j+1]=Array[j];
8               j=j-1;
9           }
10          Array[j+1]=key;
11      }
12  }
```

2.2 Merge Sort

```
13  public int[] Merge(int[] Array){
14      int n = Array.length;
15      if(n<=1)
16          return Array;
17      int[] left = new int[(n/2)];
18      int[] right = new int[n-(n/2)];
19      for(int i = 0;i<n/2;i++)
20          left[i]=Array[i];
21      for(int i = 0;i<n-(n/2);i++)
22          right[i]=Array[i+ n/2];
23      left=Merge(left);
24      right=Merge(right);
25      return Merger(left,right);
26  }
27  public int[] Merger(int[] Array, int[] Brray){
28      int[] Crray = new int[Array.length+ Brray.length];
29      int a=0;
30      int b=0;
31      int c=0;
```

```
        int ar = Array.length;
        int br = Brray.length;
        while(ar>0 && br>0){
            if (Array[0]> Brray[0]){
                Crray[c++]= Brray[0];
                if(br>1)
                    Brray[0] = Brray[++a];
                br--;
            }
            else{
                Crray[c++]= Array[0];
                if(ar>1)
                    Array[0]=Array[++b];

                ar--;
            }
        }
        while(ar>0){
            Crray[c++]= Array[0];
            if(ar>1)
                Array[0]=Array[++b];
            ar--;
        }
        while(br>0){
            Crray[c++]= Brray[0];
            if(br>1)
                Brray[0] = Brray[++a];
            br--;
        }
        return Crray;
    }
```

2.3 Pigeonhole Sort

```
    public int[] Pigeonhole(int[] Array){
        int n=Array.length;
        int min = Array[0];
        int max = Array[0];
        for(int a=0; a<n; a++) {
            if(Array[a] > max)
                max = Array[a];
            if(Array[a] < min)
                min = Array[a];
        }
        int range = max-min+1;
        int[] output = new int[n];
        int[][] holes = new int[range][];
```

```
76          int h;
77          int[] holes2;
78          for (int k : Array) {
79              if(holes[k - min]==null)
80                  holes[k - min]=new int[0];
81              h = holes[k - min].length;
82              holes2 = new int[h+1];
83              for(int i=0;i<h;i++)
84                  holes2[i]=holes[k - min][i];
85              holes2[h]=k;
86              holes[k-min]=holes2;
87          }
88          int index=0;
89          for(int i=0;i<range;i++)
90              if(holes[i]!=null)
91                  for (int hole : holes[i])
92                      output[index++]=hole;
93          return output;
94      }
```

2.4 Counting Sort

```
95      public int[] Counting(int[] Array,int k){
96          int[] count = new int[k+1];
97          for(int i=0;i<k+1;i++)
98              count[i]=0;
99          int size = Array.length;
100         int[] output = new int[size];
101         int j;
102         for(int i=0;i<size;i++){
103             j= Array[i];
104             count[j]+=1;
105         }
106         for(int i=1;i<k+1;i++)
107             count[i]+=count[i-1];
108         for(int i=size-1;i>-1;i--){
109             j= Array[i];
110             count[j]-=1;
111             output[count[j]]=Array[i];
112         }
113         return output;
114     }
```

Table 1: Results of the running time tests performed on the random data of varying sizes (in ms).

Algorithm	Input Size									
	512	1024	2048	4096	8192	16384	32768	65536	131072	251281
Insertion sort	0	1.2	0.4	2.4	7.6	34	121	470	1919.6	7782.9
Merge sort	0	0.8	0.4	0.4	0.4	1.2	4	8.4	17.6	37.2
Pigeonhole sort	198.3	93.9	93.8	94.7	95.5	96.3	99	112.2	145.6	193.8
Counting sort	287.2	289.9	299.3	290.9	296.3	291.7	294.5	292.5	287.8	327

Table 2: Results of the running time tests performed on the sorted data of varying sizes (in ms).

Algorithm	Input Size									
	512	1024	2048	4096	8192	16384	32768	65536	131072	251281
Insertion sort	0	0	0	0	0	0	0	0	0	0.4
Merge sort	0	0	0.4	0	0.8	1.2	0	4	10	20.4
Pigeonhole sort	197.6	93.8	93.8	96.2	95.1	94.7	98.6	109.1	135.2	166.9
Counting sort	284.8	292.3	294.4	298	290	300.8	291.4	312.6	293.1	294.7

Table 3: Results of the running time tests performed on the reversed sorted data of varying sizes (in ms).

Algorithm	Input Size									
	512	1024	2048	4096	8192	16384	32768	65536	131072	251281
Insertion sort	0	0.4	0.4	3.6	14.4	60.8	238.3	947.5	3788.9	13875.7
Merge sort	0	0.4	0	0	1.6	1.1	2	4.8	10.4	22.8
Pigeonhole sort	195	92.3	94.1	93.4	92.8	94	98.6	106.6	136.2	167.9
Counting sort	363.3	296.5	294.3	297.9	296.1	298.6	292.5	294.5	292	294.1

3 Results, Analysis, Discussion

Complexity analysis tables to complete:

Table 4: Computational complexity comparison of the given algorithms.

Algorithm	**Best Case**	**Average Case**	**Worst Case**
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Merge Sort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$
Pigeonhole Sort	$\Omega(n + r)$	$\Theta(n + r)$	$O(n + r)$
Counting Sort	$\Omega(n + k)$	$\Theta(n + k)$	$O(n + k)$

Table 5: Auxiliary space complexity of the given algorithms.

Algorithm	**Auxiliary Space Complexity**
Insertion Sort	$O(1)$
Merge Sort	$O(n)$
Pigeonhole Sort	$O(n+r)$
Counting Sort	$O(n+k)$

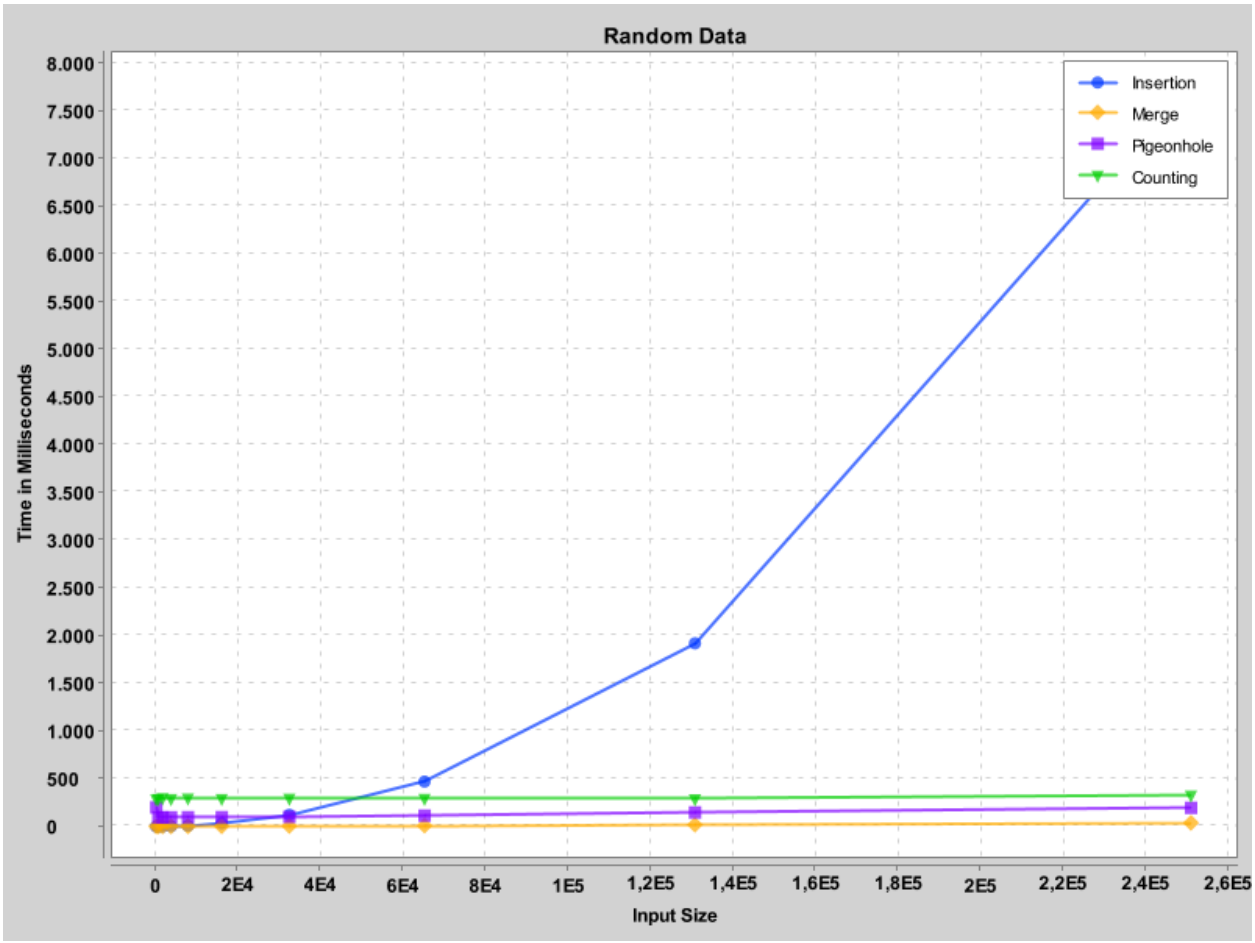

Figure 1: Random Data

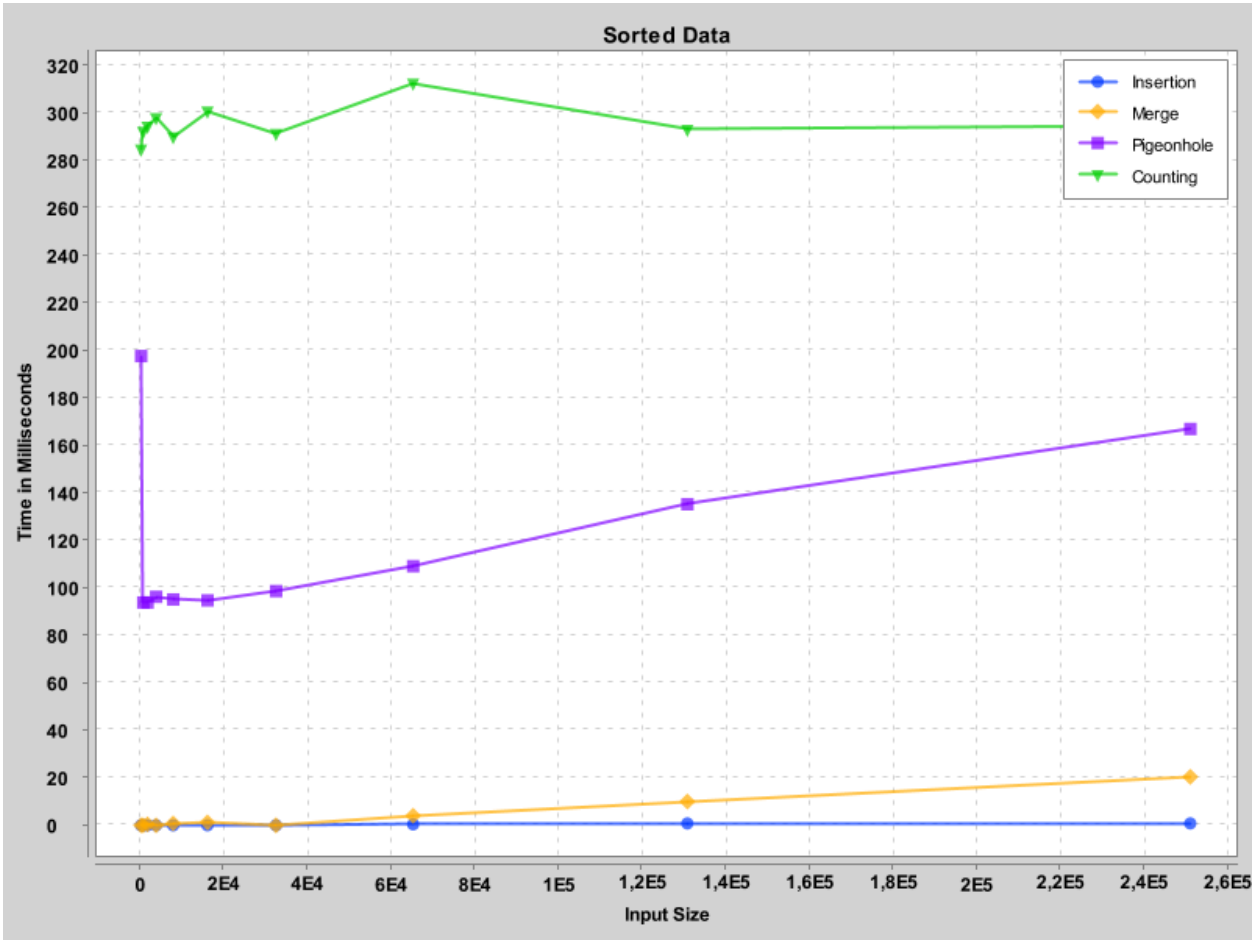

Figure 2: Sorted Data

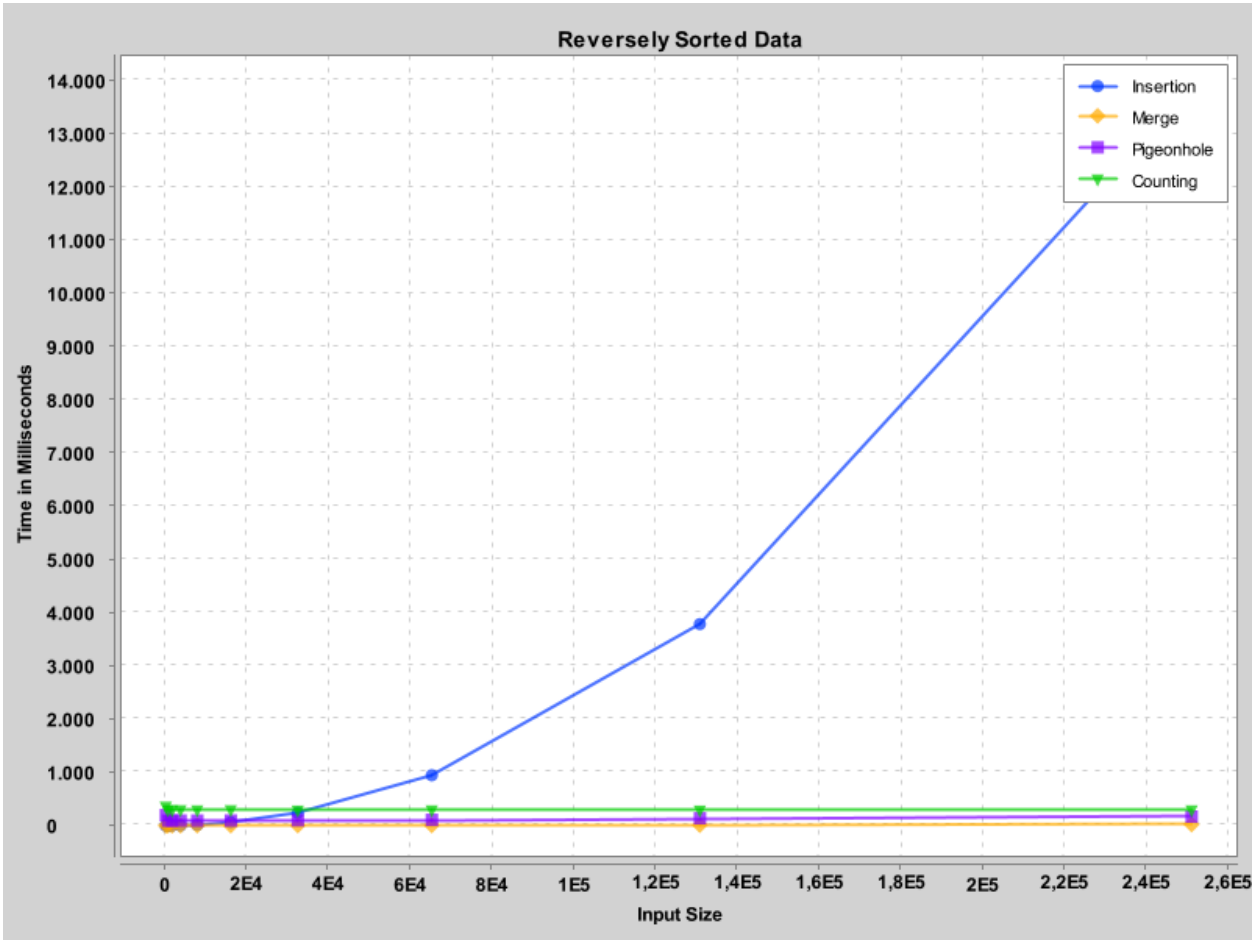

Figure 3: Reversely Sorted Data

There are graphs that shows what sorting algorithm is better or faster in which situation. As we can see Merge is better at average and worst case. Insertion is better at best case because Insertion is just O(n) when the data is sorted. Pigeonhole and Counting is very bad at sorted data. Pigeonhole and Counting have similiar results at reverse and random data.

4 Notes

There is a strange situation in my computer. When I run my code, there were 2 results. Sometimes pigeonhole was faster than counting. Sometimes counting was faster than pigeonhole. I uploaded pigeonhole faster version to this report. I don't know why that is happening but essentially the results are matched with their theoretical asymptotic complexities.

References

- https://www.youtube.com/watch?v=nVQz0kZNC64
- https://www.youtube.com/watch?v=7zuGmKfUt7s