HACETTEPE UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BBM 204 ASSIGNMENT 1 - 2022 SPRING

March 9, 2022

Student Name: Umut GÜNGÖR
Student Number: 21946198

Problem Definition

In this assignment, we are asked to implement 4 different sorting algorithms. Once implement these sorting algorithms, we are going to show their time complexities using plotting graphs.

Java Code

```java
public void countingSort()
{

    int n = arr.length;                 // length of the array
    int[] output = new int[n + 1];      // auxiliary space O(n)

    int max = arr[0];                   // the maximum element in the array

    for (int i = 1; i < n; i++)
    {
        if (arr[i] > max)
            max = arr[i];
    }

    int[] count = new int[max + 1];     // auxiliary space O(k) where k is max element

    for (int i = 0; i < max; i++)
    {
        count[i] = 0;
    }

    for (int i = 0; i < n; i++)
    {
        count[arr[i]]++;
    }

    for (int i = 1; i <= max; i++)
    {
        count[i] += count[i - 1];
    }

    for (int i = n - 1; i >= 0; i--)
    {
        output[count[arr[i]] - 1] = arr[i];
        count[arr[i]]--;
    }

    for (int i = 0; i < n; i++)
    {
        arr[i] = output[i];
    }
}
```

Counting Sort

```java
public void insertionSort()
{
    int N = arr.length;                         // length of the array

    for(int i = 0; i < N; i++)
    {
        int j = i;

        while(j != 0 && arr[j] < arr[j-1])      // classic insertion sort algorithm
        {
            int temp = arr[j];
            arr[j] = arr[j-1];
            arr[j-1] = temp;
            j--;
        }

    }
}
```

Insertion Sort

```java
public void pigeonholeSort()
{
    int N = arr.length;
    int min = arr[0];
    int max = arr[0];

    for(int i = 0; i < N; i++)
    {
        if(arr[i] < min)
            min = arr[i];                   // the minimum element in the array

        if(arr[i] > max)
            max = arr[i];                   // the maximum element in the array
    }

    int range = max - min + 1;

    int[] holes = new int[range];           // auxiliary space O(N) where N is max element - min element

    for(int i = 0; i < N; i++)
    {
        holes[arr[i] - min]++;
    }

    int index = 0;

    for(int i = 0; i < range; i++)
    {
        while(holes[i] --> 0)               // make sure the position is checked first time which means zero but initial zero
        {
            arr[index++] = i + min;
        }
    }
}
```

Pigeonhole Sort

```java
public void callMergeSort() { mergeSort( low: 0,  high: arr.length - 1); }


private void mergeSort(int low, int high)               //low = 0   &   high = arr.length - 1
{

    if(low < high && (high - low) >= 1)
    {
        int mid = (high + low) / 2;
        mergeSort(low, mid);                            // recursively called the function
        mergeSort( low: mid + 1, high);

        merge(low, high);                               // merge the arrays
    }
}


private void merge(int low, int high)
{
    int mid = (low + high) / 2;
    int left = low;
    int right = mid + 1;

    ArrayList<Integer> aux = new ArrayList<>();         // auxiliary space O(n) because of recursion where n is length of the array which is going to be sorted

    while(left <= mid && right <= high)
    {
        if(arr[left] <= arr[right])
        {
            aux.add(arr[left++]);
        }

        else
        {
            aux.add(arr[right++]);
        }
    }
```

```java
        while(left <= mid)
        {
            aux.add(arr[left++]);
        }

        while(right <= high)
        {
            aux.add(arr[right++]);
        }

        int i = 0;
        int j = low;

        while(i < aux.size())
        {
            arr[j++] = aux.get(i++);
        }
    }
```

Merge Sort

Algorithm	Input Size									
	512	1024	2048	4096	8192	16384	32768	65536	131072	251281
Insertion Sort	0.0…	0.0…	0.0…	0.0…	2	10	38	149	626	2566
Merge Sort	0.0…	0.0…	0.0…	0.0…	1	3	6	14	37	63
Pigeonhole Sort	70	62	62	62	62	62	62	63	64	65
Counting Sort	178	127	127	127	127	128	127	128	129	132

Random Data

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…	0.0…	0.0…	0.0….	1	2
Merge Sort	0.0…	0.0…	0.0…	0.0…	0.0…	2	4	9	19	40
Pigeonhole Sort	71	62	62	62	63	62	63	63	64	65
Counting Sort	144	127	128	127	127	128	128	128	129	136

Sorted Data

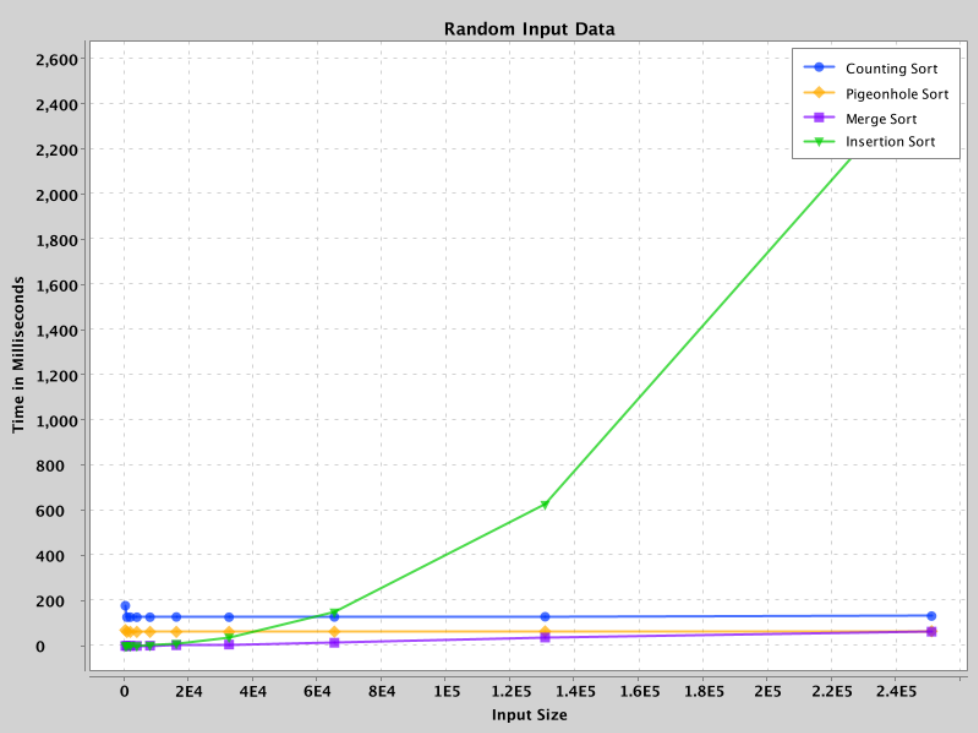
Algorithm	Input Size									
	512	1024	2048	4096	8192	16384	32768	65536	131072	251281
Insertion Sort	0.0…	0.0…	0.0…	1	4	9	76	307	1232	4568
Merge Sort	0.0…	0.0…	0.0…	0.0…	1	2	4	9	20	45
Pigeonhole Sort	76	63	63	63	63	64	64	64	65	64
Counting Sort	159	128	128	128	128	128	129	130	130	130

Reversely Sorted Data

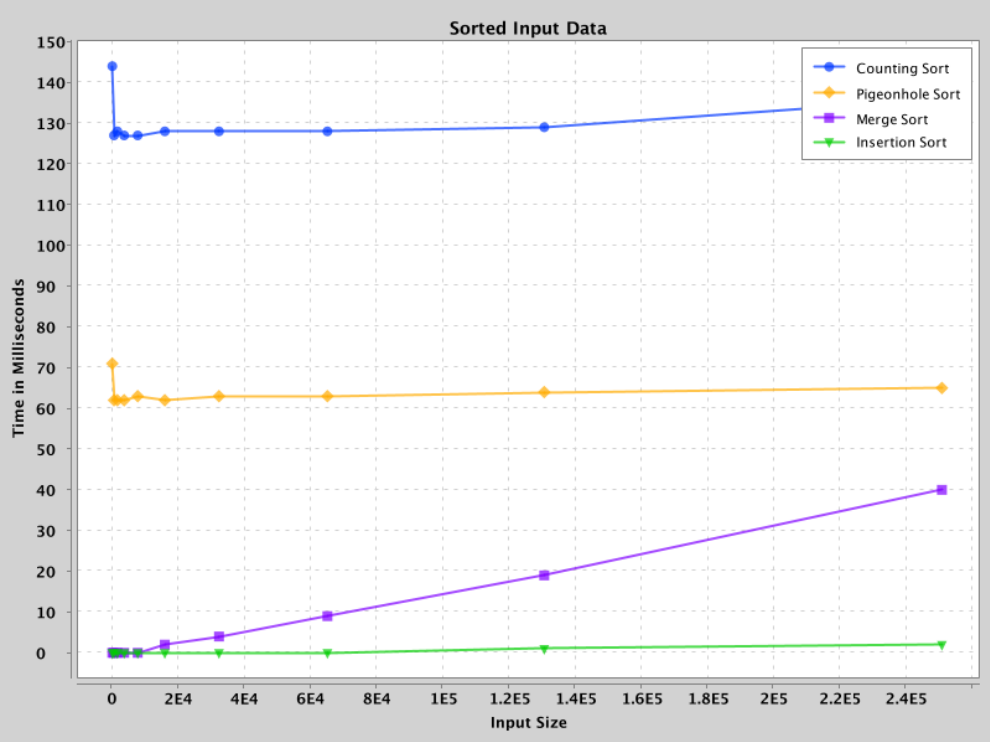
Algorithm	Best Case	Average Case	Worst Case
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$
Merge Sort	O(nlgn)	O(nlgn)	O(nlgn)
Pigeonhole Sort	O(N + n)	O(N + n)	O(N + n)
Counting Sort	O(n + k)	O(n + k)	O(n + k)

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

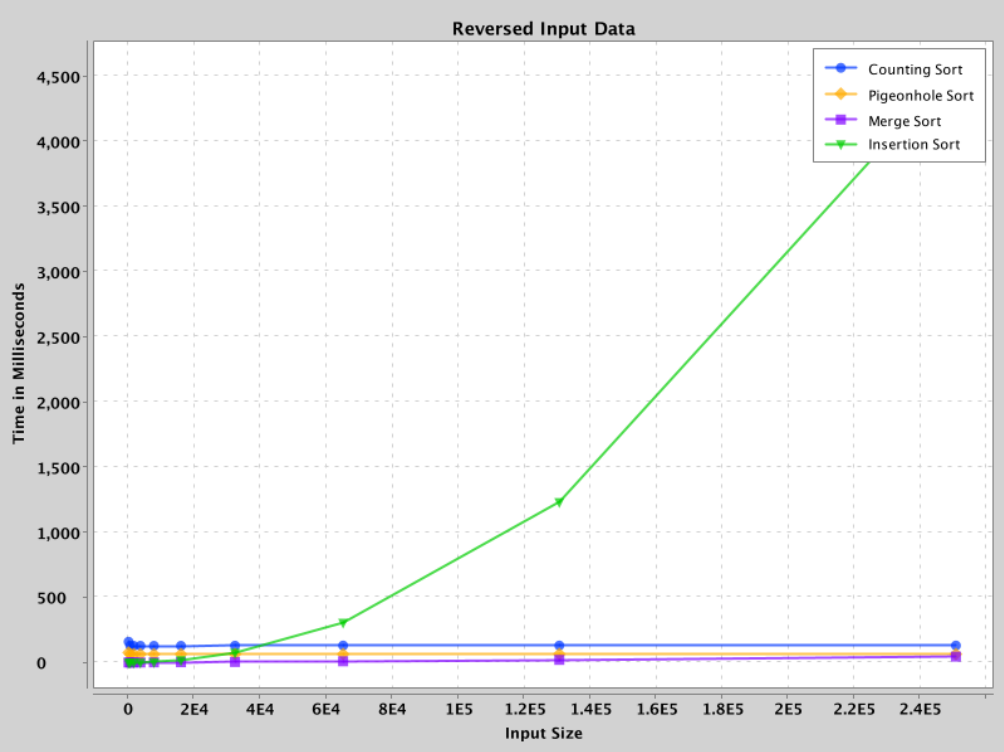
For insertion sort, it does not use extra memory but in average and worst case its time complexity is $O(n^2)$. Insertion sort's best case is O(n) where the input array is already sorted. Merge sort is an algorithm which has O(nlgn) worst case time complexity. Also, its best and average cases are O(nlgn). However, it uses an extra array so auxiliary space complexity is O(n). These two are comparison-based sorting algorithms. Pigeonhole and counting sort algorithms are non-comparison-based sorting algorithms. Actually, they are really similar algorithms with each other. In pigeonhole sort best, average and worst case time complexities are O(N + n) where N is maximum element's value – minimum element's value. Its auxiliary space complexity is O(N) because it uses an extra array sized N. When looking at counting sort algorithm similar to the pigeonhole sort its best, average and worst case time complexities are O(n + k) where k is the maximum element's value in the array. It's auxiliary space complexity is O(n + k) beacuse it uses extra two arrays. One of them is sized n which is input array's size and another is sized k.



Random Input Data



Sorted Input Data



Reversely Sorted Input Data

For best case insertion sort is the best algorithm because its time complexity is O(n). For average and worst case the best algorithm is pigeonhole sort because O(N + n). This N is less than k in counting sort so, pigeonhole sort is better.

My results which are obtained from graphs are close to theoretical asymptotic complexities. However, first two or three inputs it does not obey theoretical asymptotic complexites. One of the reason is cache. Computer remmebers the data which it process for a short time. Apart from that, insertion sort is actually works perfectly. The other algorithms are also nearly works like their expected complexities, especially merge sort.

References

stackoverflow.com
geeksforgeeks.org
wikipedia.org