Hacettepe University

Computer Engineering Department

BBM204 Software Practicum II - 2024 Spring

Programming Assignment 1

March 19, 2024

Student name:
Gülvera Yazilitaş

Student Number:
b2210356111

1 Problem Definition

The problem is understanding the behavior of insertion, merge, and counting sort, and linear and binary search algorithms by the input size and order type.

2 Solution Implementation

The Java code of the algorithms and running time test results are given below to analyze the time and auxiliary space complexity.

2.1 Insertion Sort Algorithm

```
public boolean compare(int a, int b){
        return a<b;
    }
    public int[] insertion_sort(int[] arr) {
        int n = arr.length;
        for (int i = 1; i < n; i++) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && compare(key, arr[j])) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
        }
        return arr;
    }
```

2.2 Merge Sort Algorithm

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

      return merge(left, right);
  }
   private int[] merge(int[] left, int[] right) {
       int[] result = new int[left.length + right.length];
       int leftIndex = 0, rightIndex = 0, resultIndex = 0;
```

```
31
32          while (leftIndex < left.length && rightIndex < right.length) {
33              if (left[leftIndex] < right[rightIndex]) {
34                  result[resultIndex++] = left[leftIndex++];
35              } else {
36                  result[resultIndex++] = right[rightIndex++];
37              }
38          }
39
40          while (leftIndex < left.length) {
41              result[resultIndex++] = left[leftIndex++];
42          }
43
44          while (rightIndex < right.length) {
45              result[resultIndex++] = right[rightIndex++];
46          }
47
48          return result;
49      }
```

2.3 Counting Sort Algorithm

```
50  public int[] countingSort(int[] arr ){
51          int k = Arrays.stream(arr).max().getAsInt(); //max
52          int[] countArr = new int[k+1];
53          Arrays.fill(countArr, 0);
54          int[] outArr = new int[arr.length];
55
56          for (int j : arr) {
57              countArr[j]++;
58          }
59
60          for (int i = 1; i < k+1; i++) {
61              countArr[i]=countArr[i]+countArr[i-1];
62          }
63
64          for (int m = arr.length-1; m >=0 ; m--) {
65              int j=arr[m];
66              countArr[j]=countArr[j]-1;
67              outArr[countArr[j]]=arr[m];
68          }
69
70          return outArr;
71      }
```

2.4 Linear Search Algorithm

```
 public int linearSearch(int[] arr, int value){
        for (int i = 0; i < arr.length; i++) {
            if(arr[i]==value){
                return i;
            }
        }
        return -1;
    }
```

2.5 Binary Search Algorithm

```
public int binarySearch(int[] sortedArr,int value){
        int low=0;
        int high=sortedArr.length-1;
        while (high>=low){
            int mid=(high+low)/2;
            if(sortedArr[mid]==value){
                return mid;
            }else if(sortedArr[mid]<value){
                low=mid+1;
            }else {
                high=mid-1;
            }
        }
        return -1;
    }
```

3 Results

Running time test results for sorting algorithms are given in Table 1.

Table 1: Results of the running time tests performed for varying input sizes (in ms).

	Input Size n									
Algorithm	**500**	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**	**64000**	**128000**	**250000**
	Random Input Data Timing Results in ms									
Insertion sort	0	1	0	1	6	31	106	304	1158	4413
Merge sort	0	0	0	0	1	3	5	9	19	42
Counting sort	655	287	298	297	309	273	280	301	282	736
	Sorted Input Data Timing Results in ms									
Insertion sort	0	0	0	0	0	0	0	0	0.1	0.2
Merge sort	0	0	0	0	1	1	2	13	16	30
Counting sort	297	315	287	271	272	272	284	272	277	278
	Reversely Sorted Input Data Timing Results in ms									
Insertion sort	0	0	0	2	8	35	145	571	2305	8795
Merge sort	0	0	0	0	0	1	2	5	9	21
Counting sort	272	275	291	280	299	277	269	274	273	268

Running time test results for search algorithms are given in Table 2.

Table 2: Results of the running time tests of search algorithms of varying sizes (in ns).

	Input Size n									
Algorithm	**500**	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**	**64000**	**128000**	**250000**
Linear search (random data)	1289	1655	276	472	876	1559	2519	4299	9435	14691
Linear search (sorted data)	216	521	634	856	1557	2948	4988	9517	15660	34736
Binary search (sorted data)	347	274	394	463	885	1592	2045	2619	3220	3406

Complexity analysis tables to complete (Table 3 and Table 4):

Table 3: 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	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$
Counting Sort	$\Omega(n+k)$	$\Theta(n+k)$	$O(n+k)$
Linear Search	$\Omega(1)$	$\Theta(n)$	$O(n)$
Binary Search	$\Omega(1)$	$\Theta(logn)$	$O(logn)$

Table 4: Auxiliary space complexity of the given algorithms.

Algorithm	**Auxiliary Space Complexity**
Insertion sort	$O(1)$
Merge sort	$O(n)$
Counting sort	$O(n+k)$
Linear Search	$O(1)$
Binary Search	$O(1)$

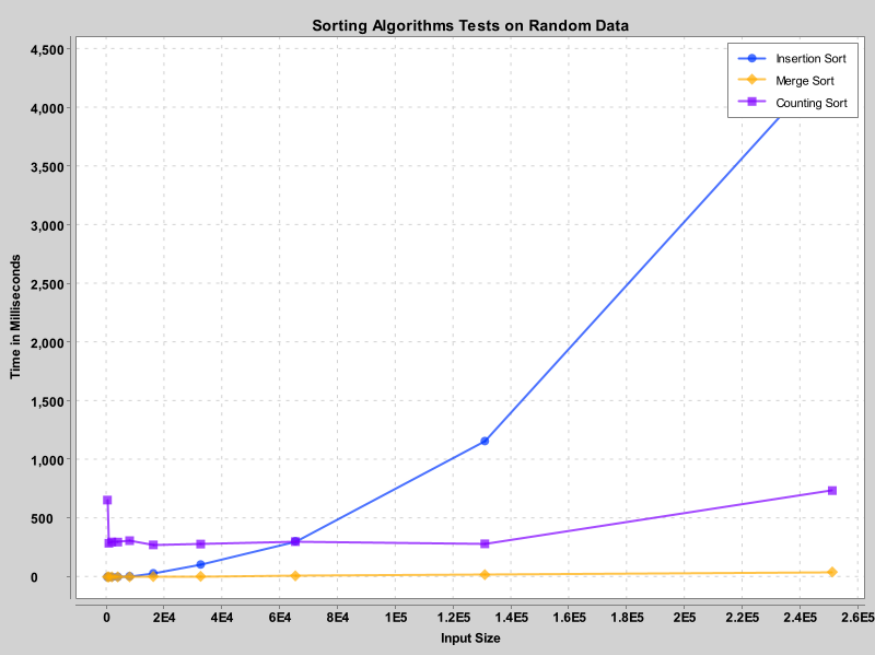

Figure 1: Random Data Graph

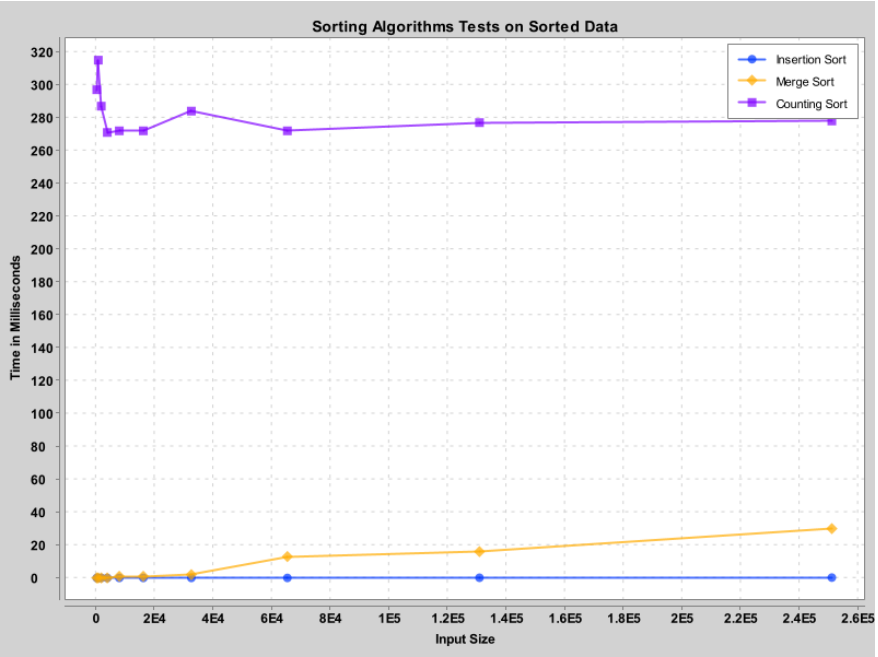

Figure 2: Sorted Data Graph

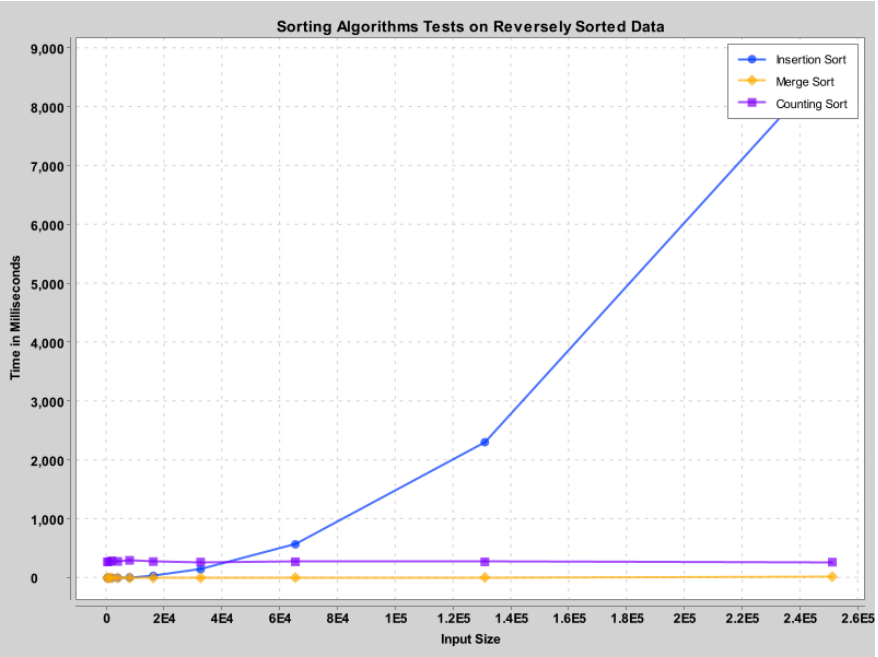

Figure 3: Reversely Sorted Data Graph

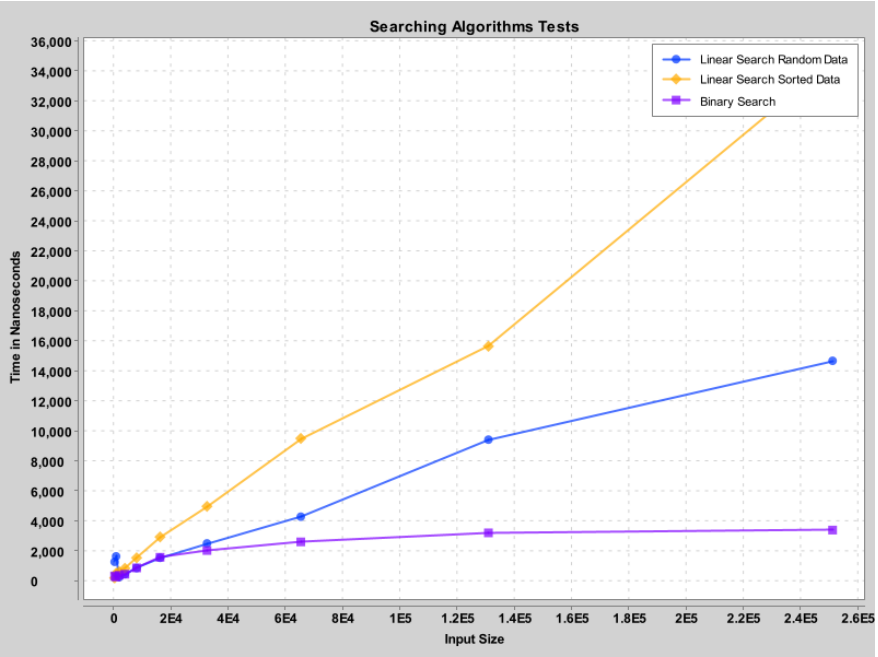

Figure 4: Searching Algorithms Graph

4 Anlysis and Discussion

Firstly, as can be seen from Fig.1 and Fig.3, the merge sort algorithm is the fastest when the data is either random or reversely sorted, while insertion sort is by far the slowest. However, when the data is already sorted, the running time of insertion sort is almost 0 ms, which proves that the insertion sort's best-case time complexity is O(1). Furthermore, by examining all three graphs, it is evident that the counting sort algorithm shows notable consistency. Of particular note is that counting sort's best, worst, and average case time complexities tend to be the same.

Secondly, upon examining Fig.4, the binary search algorithm seems to be the best choice in terms of running time. Its line on the graph is quite similar to the (logn) line, which tells that the experiments provided accurate results. On the other hand, when examining linear search, we observe that its slope is more vertical, indicating a longer running time. This observation aligns with the fact that its time complexity is O(n).

As a result, it can be concluded that both merge sort and binary search are more efficient algorithms.