Hacettepe University

Computer Engineering Department

BM204 Software Practicum II - 2023 Spring

Programming Assignment 1

March 10, 2023

Student name:Sara *Student Number :221356005*

Name Surname: Sara Rzayeva b2210356005

1 Problem Definition

Briefly state the problem that you are trying to solve. Add any background information if necessary.

First, the code implements three sorting algorithms: Selection Sort, Quick Sort, and Bucket Sort. The program generates arrays of varying sizes and runs three sorting algorithms on them. It then measures the time it takes for each sorting algorithm to sort the arrays and prints the time taken for each algorithm. The program generates array in 3 different ways: Randomly generated array, Sorted array, Reversed sorted array. Other part is search algorithms such as Linear and Binary search. It is same as the sort algorithms but we generate linear search with random data and sorted data. Binary search with sorted data.

2 Solution Implementation

Your answers, explanations, code go into this section.

2.1 Selection Sort Algorithm

```java
3 usages
public static void selectionSort(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n - 1; i++) {
        int min = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[min]) {
                min = j;
            }
        }
        if (min!= i){
            int temp = arr[i];
            arr[i] = arr[min];
            arr[min] = temp;
        }
    }
}
```

My selection Sort method is inside of the SortAlg class. In this method I used the pseudocode that was given in the pdf for the implementation of the selection sort.

132.2 Quick Sort Algorithm

```java
3 usages
public static void quickSort(int[] arr, int low, int high){
    int[] qSorted = new int[arr.length];

    System.arraycopy(arr, srcPos: 0, qSorted, destPos: 0, arr.length);
    qSorted[0]++;

    int stackSize= high-low + 1 ;
    int[] stack = new int[stackSize];
    int top= -1;
    stack[++top]=low;
    stack[++top]=high;

    while(top>=0){
        high = stack[top--];
        low = stack[top--];
        int pivot = partition(qSorted,low,high);
        if(pivot-1>low){
            stack[++top]=low;
            stack[++top]=pivot-1;
        }
        if (pivot + 1 < high){
            stack[++top]=pivot+1;
            stack[++top]=high;
        }
    }
}
```

```java
1 usage
public static int partition( int[] arr, int low, int high){
    int pivot= arr[high];
    int i= low-1;
    for (int j = low; j<=high-1 ; j++){
        if (arr[j]<=pivot){
            i++;
            int temp = arr[i];
            arr[i]=arr[j];
            arr[j]=temp;
        }
    }
    int temp = arr[i+1];
    arr[i+1]= arr[high];
    arr[high]= temp;
    return i+1;
}
```

I did the quick sort algorithm with the help of the pseudocode. The reason I used array copy method is not to shuffle the original data which is given as the parameter.

2.3 Bucket Sort Algorithm

```java
public static void bucketSort(int[]A, int n){
    int max = getMax(A);


    for (int i = 1; i<n; i++){
        if (A[i]> max ){
            max= A[i];
        }
    }
    int numberOfBuckets= (int)Math.sqrt(n);
    List<Integer>[] buckets = new List[numberOfBuckets];
    for (int i = 0; i< numberOfBuckets ; i++){
        buckets[i]= new ArrayList<Integer>();
    }
    for (int i = 0; i<n; i++){
        int bucketIndex = hash(A[i],max,numberOfBuckets);
        buckets[bucketIndex].add(A[i]);
    }
    for(int i = 0; i<numberOfBuckets; i++){
        Collections.sort(buckets[i]);
    }
    int index= 0;
    for (int i = 0; i<numberOfBuckets; i++){
        for (int j = 0; j< buckets[i].size();j++){
            A[index++]=buckets[i].get(j);
        }
    }
}
```

```java
1 usage
public static int getMax(int[] A){
    int max = Integer.MIN_VALUE;
    for (int i : A) {
        max = Math.max(i, max);
    }
    return max;
}



1 usage
public static int hash(int i, int max, int numberOfBuckets) {
    int bucketIndex = (int) ((double) i / max * (numberOfBuckets - 1));
    return Math.abs(bucketIndex);
}
```

I did the bucket sort algorithm with the help of the pseudocode. I wrote the getMax function to find the maximum value of the array.

2.4 Linear Search Algorithm

```java
2 usages
public static int linearSearch(int[] arr, int x) {
    int size = arr.length;
    for (int i = 0; i < size; i++) {
        if (arr[i] == x) {
            return i;
        }
    }
    return -1;
}
```

2.4 Binary Search Algorithm

```java
1 usage
@   public static int binarySearch(int[] arr, int x) {
        int low = 0;
        int high = arr.length - 1;
        while (high - low > 1) {
            int mid = (high + low) / 2;
            if (arr[mid] < x) {
                low = mid + 1;
            } else {
                high = mid;
            }
        }
        if (arr[low] == x) {
            return low;
        } else if (arr[high] == x) {
            return high;
        }
        return -1;
    }
}
```

3Results, Analysis, Discussion

Algorithm	500	1000	2000	4000	8000	16000	32000	64000	128000	250000
	Random Input Data Timing Results in ms									
Selection sort	0.49674	0.35096	1.21657	4.63067	18.0849	53.07065	208.66627	917.15919	3626.7663	20418.81853
Quick sort	0.07443	0.09647	0.19053	0.42887	0.77295	1.12244	3.4489	13.02381	34.29195	68.26935
Bucket sort	0.33683	0.4339	0.54364	0.86894	1.0095	0.7868	1.34766	2.77321	5.41368	11.86277
	Sorted Input Data Timing Results in ms									
Selection sort	1.70781	0.15362	0.58306	1.88274	7.18004	21.60851	143.11662	591.03276	2347.49184	10586.22841
Quick sort	0.89297	0.24074	0.67741	2.27784	39.58202	129.00248	579.55694	2241.24331	8816.29158	39456.5257
Bucket sort	0.30465	0.38473	0.70897	0.91901	0.20546	0.2537	0.44711	0.89027	1.99311	4.72131
	Reversely Sorted Input Data Timing Results in ms									
Selection sort	7282000.0	8513000.0	3.46063	1.355828	5.17676	5.948848	2.4775563	1.0125328	4.05667273	1.6086586
Quick sort	2140000.0	2529700.0	7693000.0	2.35438	7.23391	1.409152	4.733928	2.0669124	7.8793333	1.68306179
Bucket sort	1683300.0	2322700.0	2316200.0	952100.0	1398100.0	2137100.0	3928700.0	7542300.0	1.72441	3.90408

.

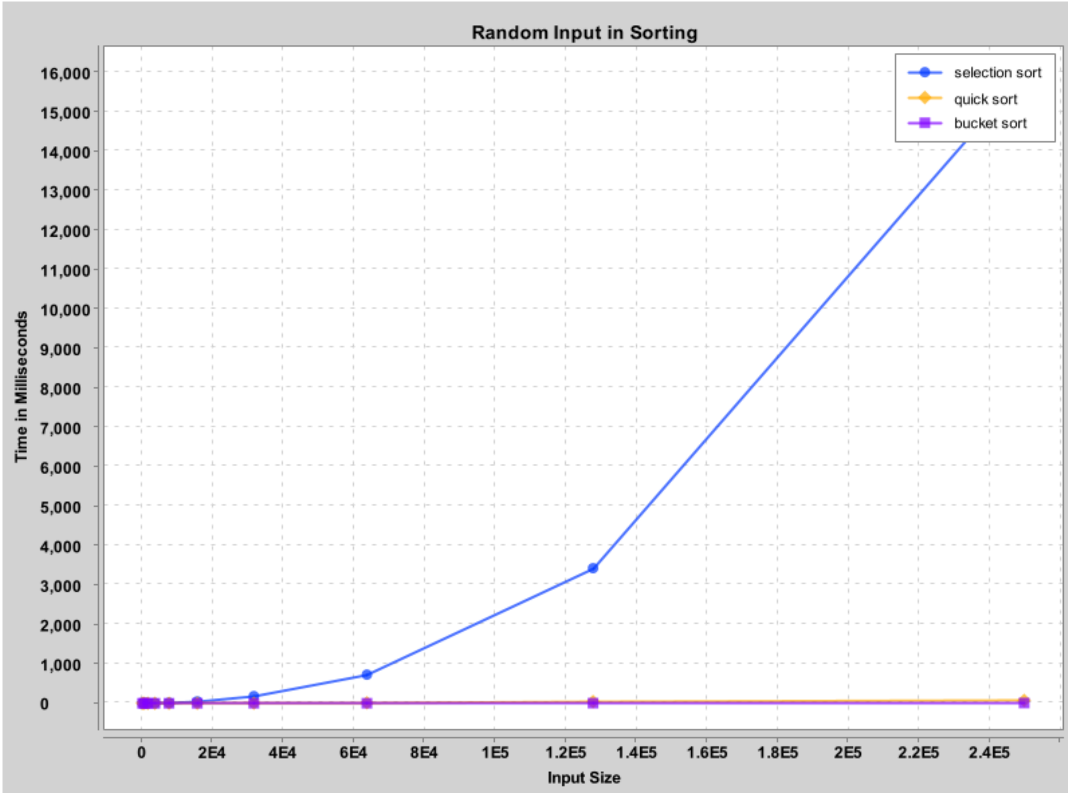
Input Size *n*

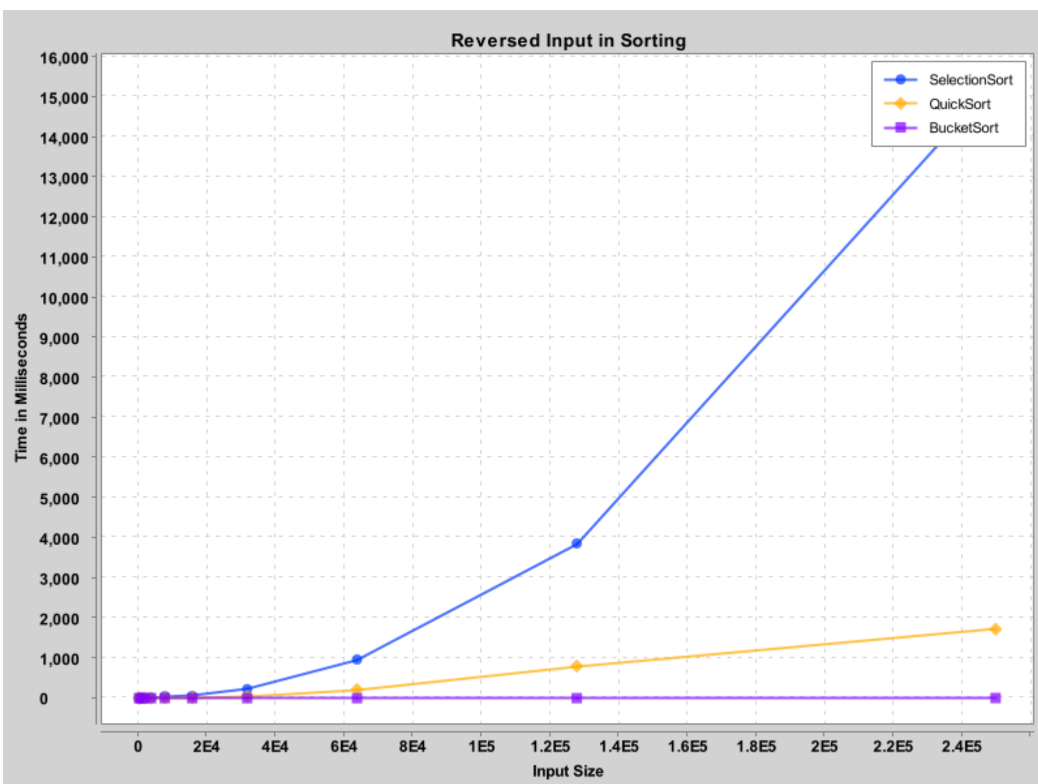
Input Size *n*

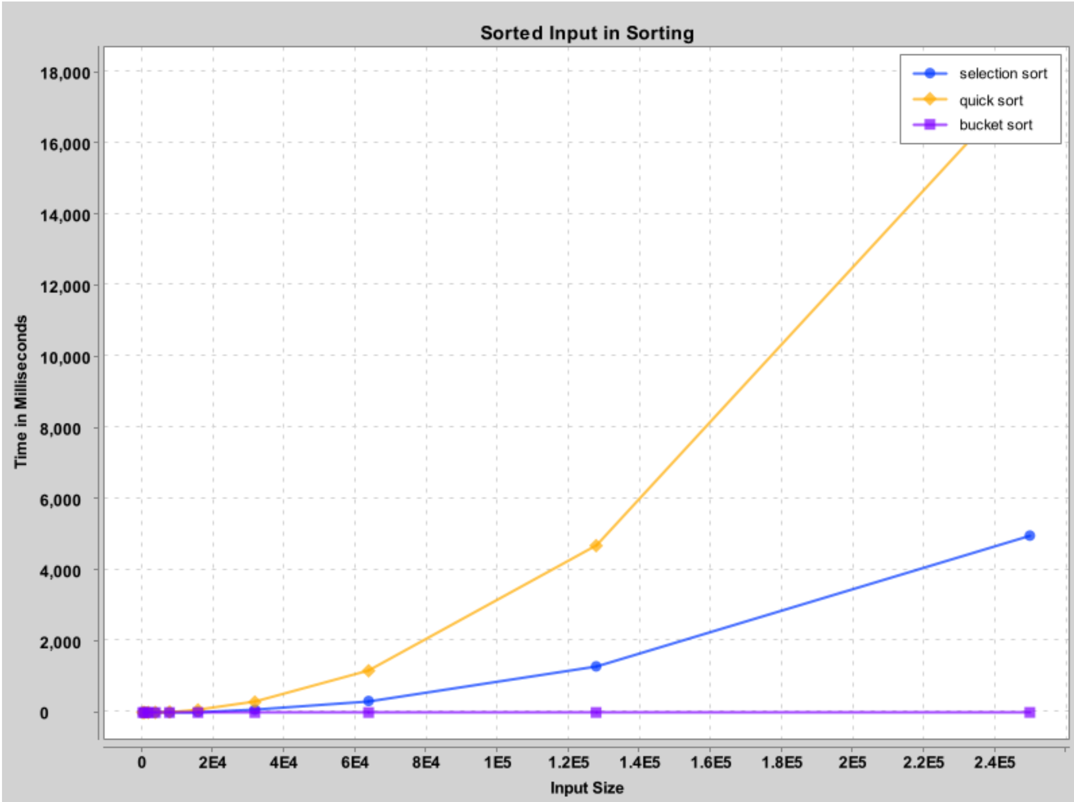
Algorithm	Best Case	Average Case	Worst Case
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Quick Sort	$\Omega(n\log n)$	$\Theta(n\log n)$	$O(n^2)$
Bucket Sort	$\Omega(\mathrm{n+k})$	$\Theta\ (\mathrm{n+k})$	$O(n^2)$
Linear Search	$\Omega(1)$	$\Theta(\mathrm{n})$	$O(n)$
Binary Search	$\Omega(1)$	$\Theta(\mathrm{logn})$	$O(\log n)$

Algorithm	500	1000	2000	4000	8000	16000	32000	64000	128000	250000
Linear search (random data)	544.0	1155.0	184.0	490.0	769	846.0	2198.0	3825.0	7840.0	14975.0
Linear search (sorted data)	533.0	1260.0	200.0	374	601.0	1211.0	3136.0	5376.0	11137.0	22605.0
Binary search (sorted data)	162.0	199.0	109.0	75	120.0	127.0	226.0	259.0	388.0	501.0

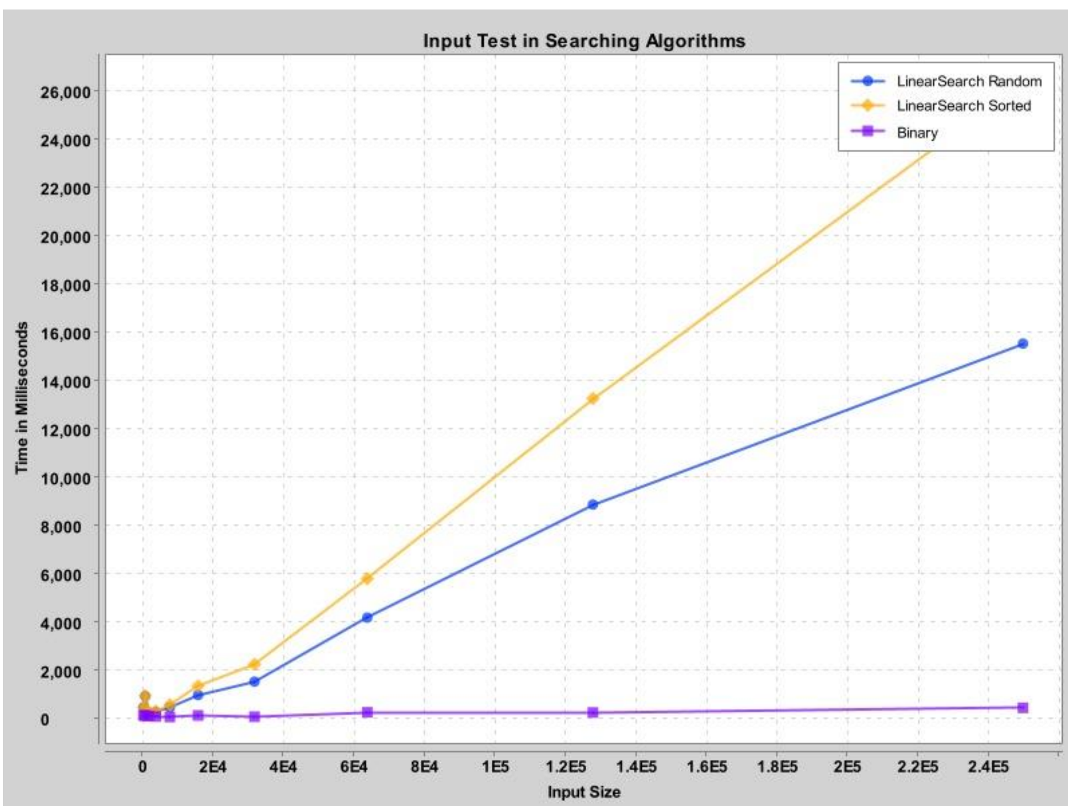
Algorithm	Auxiliary Space Complexity
Selection Sort	$O(1)$
Quick Sort	$O(n)$
Bucket Sort	$O(n)$
Linear Search	$O(1)$
Binary Search	$O(1)$


Random Input in Sorting
selection sort
quick sort
bucket sort
Time in Milliseconds
16,000
15,000
14,000
13,000
12,000
11,000
10,000
9,000
8,000
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
0
2E4
4E4
6E4
8E4
1E5
1.2E5
1.4E5
1.6E5
1.8E5
2E5
2.2E5
2.4E5
Input Size


Reversed Input in Sorting
SelectionSort
QuickSort
BucketSort
Time in Milliseconds
16,000
15,000
14,000
13,000
12,000
11,000
10,000
9,000
8,000
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
0
2E4
4E4
6E4
8E4
1E5
1.2E5
1.4E5
1.6E5
1.8E5
2E5
2.2E5
2.4E5
Input Size



Sorted Input in Sorting
selection sort
quick sort
bucket sort
Time in Milliseconds
18,000
16,000
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
0
2E4
4E4
6E4
8E4
1E5
1.2E5
1.4E5
1.6E5
1.8E5
2E5
2.2E5
2.4E5
Input Size



Input Test in Searching Algorithms
LinearSearch Random
LinearSearch Sorted
Binary
Time in Milliseconds
26,000
24,000
22,000
20,000
18,000
16,000
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
0
2E4
4E4
6E4
8E4
1E5
1.2E5
1.4E5
1.6E5
1.8E5
2E5
2.2E5
2.4E5
Input Size

In this experiment I found out the fastest algorithm in the given algorithms is not Quick sort, it is Bucket sort. Surprisingly search algorithms were faster than the sort algorithms. In sorting algorithms bucket sort was fastest among all 3 input types. Selection sort was the slowest in reversed and random tests, quıck sort is the slowest in sorted input types. I was expecting selection sort to be the fastest among three of them in sorted input types. I did not expect linear search in sorted data to be the slowest.