ROLL NO: 20MIT3308

NAME: JAYNIL PATEL

BRANCH: DATA SCIENCE

COURSE: COMPLEXITY THEORY AND ALGORITHMS (3CS1109)

TOPIC: PRACTICAL 1

AIM

Implement following sorting algorithms.

a) Quick Sort

b) Insertion Sort

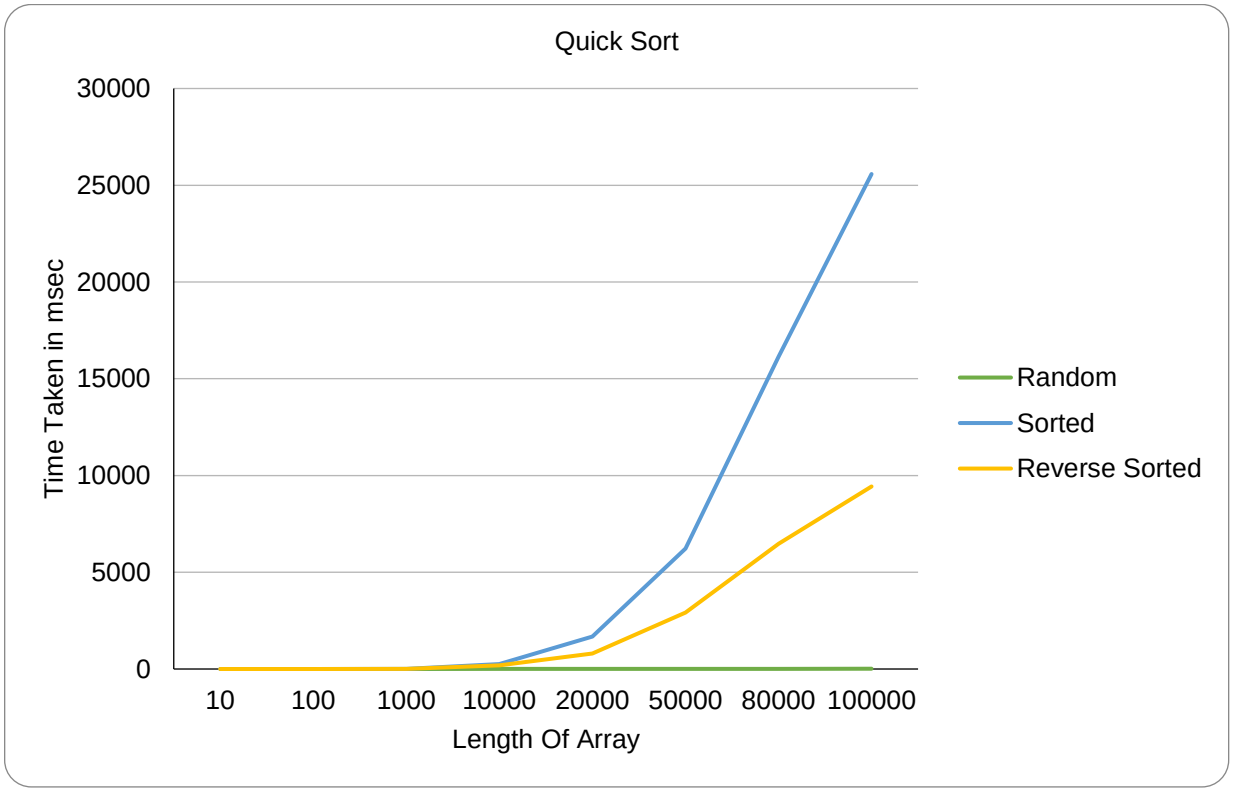
c) Bubble Sort

d) Selection Sort

- Evaluate the time complexity of each algorithm on an already sorted (ascending and descending) and non-sorted input values with varying size of input values.
- Visualize the same using graphical representation.

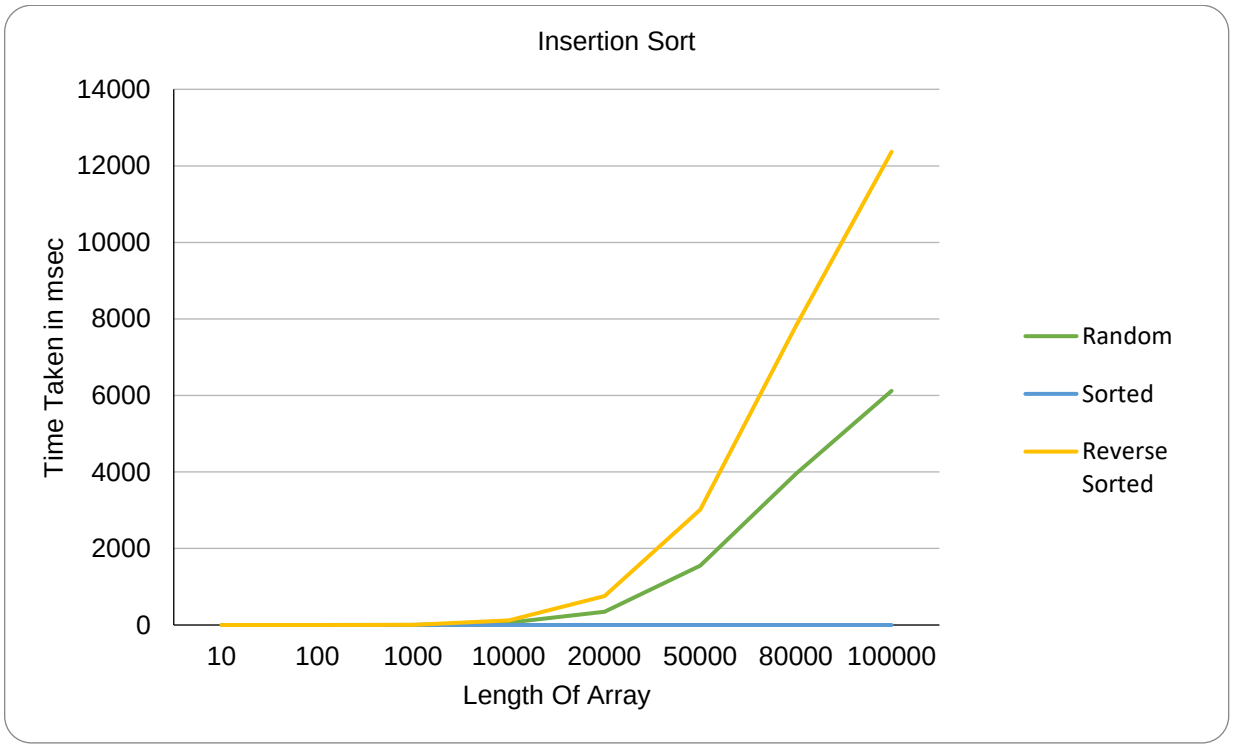
QUICK SORT

Length	Random	Sorted	Reverse Sorted
10	0	0	0
100	0	0	0
1000	0	3	3
10000	1	253	182
20000	3	1672	802
50000	7	6233	2926
80000	11	16147	6470
100000	16	25579	9433



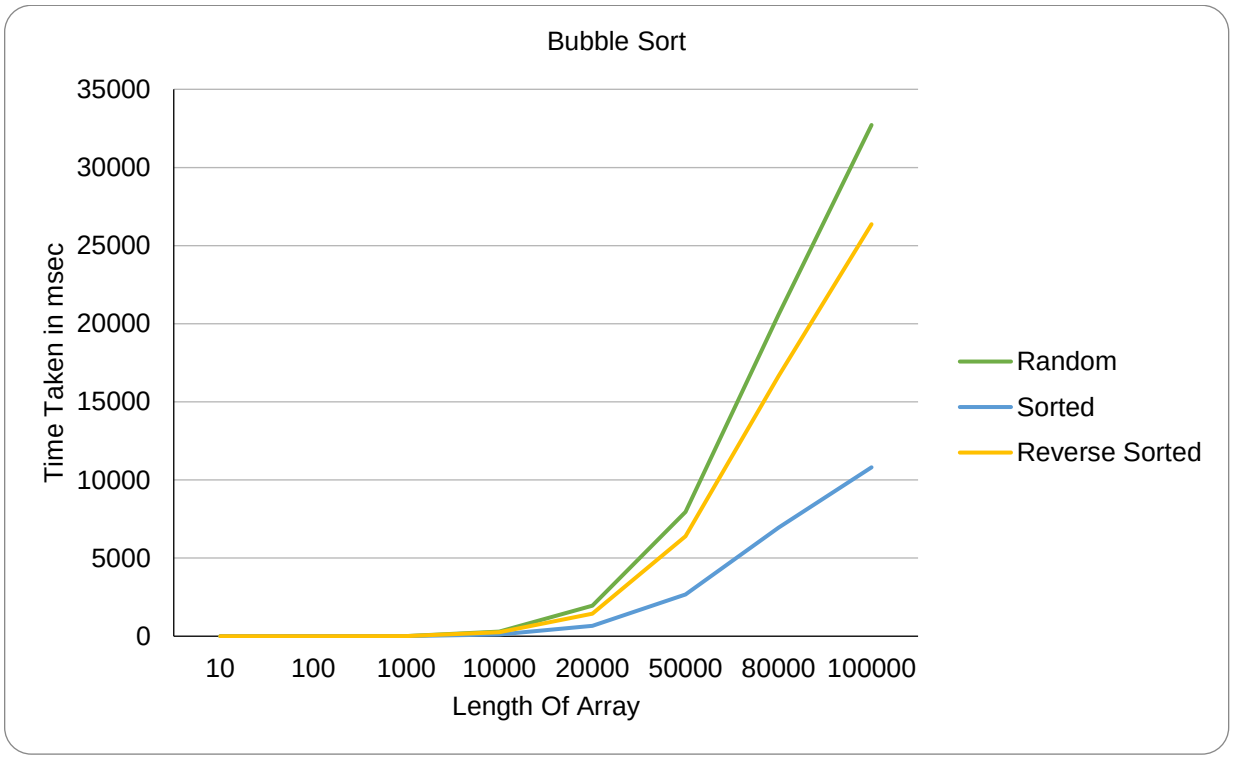
INSERTION SORT

Length	Random	Sorted	Reverse Sorted
10	0	0	0
100	0	0	0
1000	1	0	1
10000	62	0	122
20000	343	1	753
50000	1553	1	3012
80000	3952	0	7818
100000	6118	0	12366



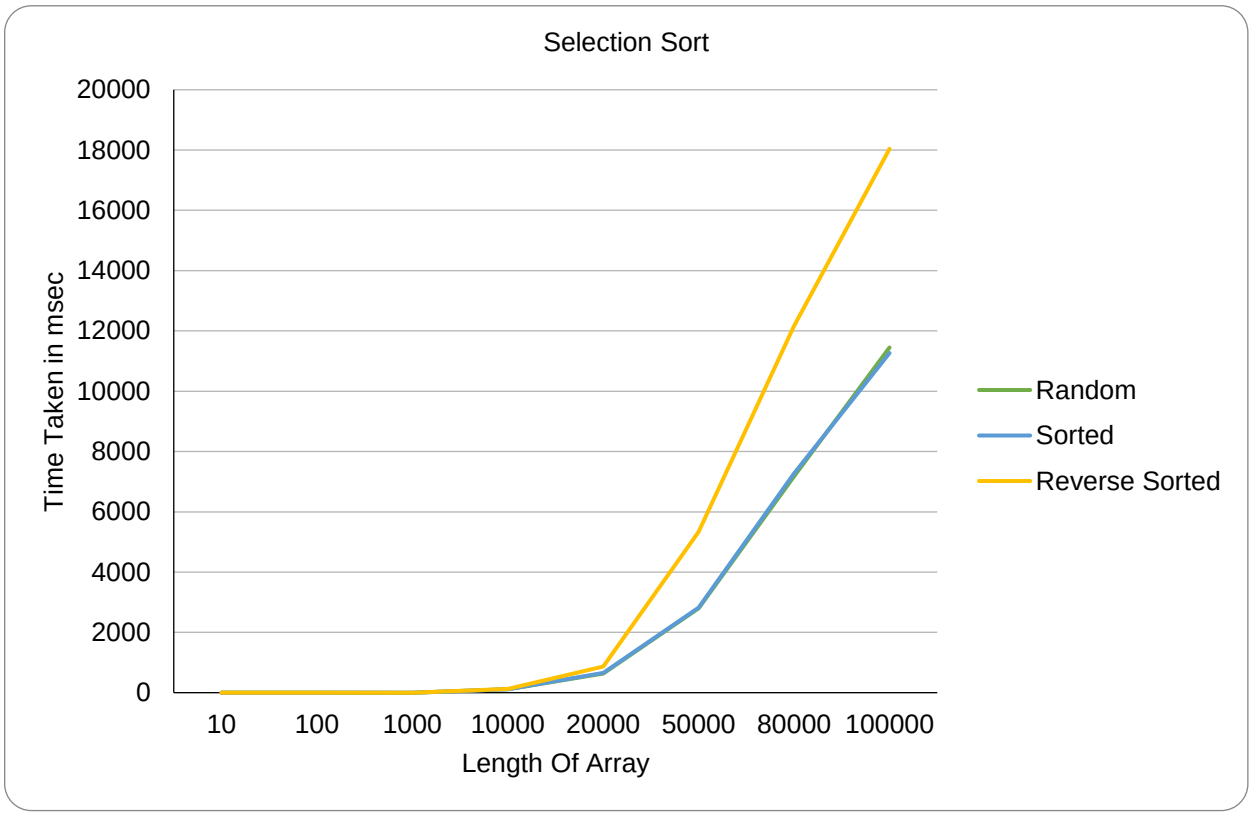
BUBBLE SORT

Length	Random	Sorted	Reverse Sorted
10	0	0	0
100	0	1	0
1000	2	1	2
10000	301	104	262
20000	1951	658	1442
50000	7946	2667	6397
80000	20594	6948	16659
100000	32715	10808	26367



SELECTION SORT

Length	Random	Sorted	Reverse Sorted
10	0	0	0
100	0	0	0
1000	1	1	1
10000	113	118	121
20000	634	659	864
50000	2805	2819	5341
80000	7171	7270	12161
100000	11450	11267	18042



CODE (C++)

```cpp
// C++ program by Jaynil Patel 3308
#include <bits/stdc++.h>
#include <cstdlib>
#include <ctime>
#include <algorithm>
#include <fstream>
#include <iostream>
#include <time.h>
using namespace std;

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
// partition the array using last element as pivot

int partition(int arr[], int low, int high)
{
    int pivot = arr[high]; // pivot
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++)
    {
        //if current element is smaller than pivot, increment the low element
        //swap elements at i and j
        if (arr[j] <= pivot)
        {
            i++; // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

//quicksort algorithm
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        //partition the array
        int pivot = partition(arr, low, high);
```

```
        //sort the sub arrays independently
        quickSort(arr, low, pivot - 1);
        quickSort(arr, pivot + 1, high);
    }
}

void selectionSort(int arr[], int n, int type)
{
    int i, j, min_idx;

    for (i = 0; i < n - 1; i++)
    {

        min_idx = i;
        for (j = i + 1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;

        swap(&arr[min_idx], &arr[i]);
    }
}

void bubbleSort(int arr[], int n, int type)
{

    int i, j;
    for (i = 0; i < n - 1; i++)

        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(&arr[j], &arr[j + 1]);
}

void insertionSort(int arr[], int n, int type)
{

    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

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

```
    cout << endl;
}

void reverseArray(int arr[], int n)
{
    for (int low = 0, high = n - 1; low < high; low++, high--)
    {
        swap(arr[low], arr[high]);
    }
}

int main()
{

    ofstream insertion, quick, bubble, selection;
    insertion.open("insertion1.csv");
    quick.open("quick1.csv");
    bubble.open("bubble1.csv");
    selection.open("selection1.csv");

    srand((unsigned)time(0));
    int randomNumber_len, randomNumber;
    time_t start_1, end_1, start_2, end_2, start_3, end_3, start_4, end_4;

    int len_arr[] = {500, 1000, 3000, 5000, 6000, 10000, 20000, 30000, 50000, 80000, 100000}
;
    //int len_arr[] = {140000};
    int debug_len = 11;

    insertion << "Length , Random , Sorted , Reverse Sorted\n";
    quick << "Length , Random , Sorted , Reverse Sorted\n";
    bubble << "Length , Random , Sorted , Reverse Sorted\n";
    selection << "Length , Random , Sorted , Reverse Sorted\n";

    for (int i = 0; i < debug_len; i++)
    {

        randomNumber_len = len_arr[i];
        int arr[randomNumber_len], arr_s[randomNumber_len], arr_b[randomNumber_len], arr_q[r
andomNumber_len];

        cout << "\nLength of  random numbers = " << randomNumber_len << "\n";
        cout << "Seq Type = Random";

        insertion << randomNumber_len << ",";
        selection << randomNumber_len << ",";
        quick << randomNumber_len << ",";
        bubble << randomNumber_len << ",";

        for (int index = 0; index < randomNumber_len; index++)
        {
            randomNumber = (rand() % (randomNumber_len * 10)) + 1;
            arr[index] = randomNumber;
            arr_s[index] = randomNumber;
            arr_b[index] = randomNumber;
            arr_q[index] = randomNumber;
```

```
        }

        cout << "(Insertion = ";
        //outfile << "(Insertion = ";
        clock_t t;
        t = clock();
        insertionSort(arr, randomNumber_len, 1);
        t = clock() - t;
        double time_taken_1 = ((double)t);
        cout << fixed << time_taken_1 << setprecision(5) << "sec)\t";
        insertion << fixed << time_taken_1 << setprecision(5) << ",";

        cout << "(Selection = ";
        //outfile << "(Selection = ";
        t = clock();
        selectionSort(arr_s, randomNumber_len, 4);
        t = clock() - t;
        double time_taken_2 = ((double)t);
        cout << fixed << time_taken_2 << setprecision(5) << "sec)\t";
        selection << fixed << time_taken_2 << setprecision(5) << ",";

        cout << "(Quick = ";
        //outfile << "(Quick = ";
        t = clock();
        quickSort(arr_q, 0, randomNumber_len - 1);
        t = clock() - t;
        double time_taken_4 = ((double)t);
        cout << fixed << time_taken_4 << setprecision(5) << "sec)\t";
        quick << fixed << time_taken_4 << setprecision(5) << ",";

        cout << "(Bubble = ";
        //outfile << "(Bubble = ";
        t = clock();
        bubbleSort(arr_b, randomNumber_len, 4);
        t = clock() - t;
        double time_taken_3 = ((double)t);
        cout << fixed << time_taken_3 << setprecision(5) << "sec)\t\n\n";
        bubble << fixed << time_taken_3 << setprecision(5) << ",";

        cout << "Seq Type = Sorted\n  ";

        cout << "(Insertion = ";
        //outfile << "(Insertion = ";
        t = clock();
        insertionSort(arr, randomNumber_len, 1);
        t = clock() - t;
        time_taken_1 = ((double)t);
        cout << fixed << time_taken_1 << setprecision(5) << "sec)\t";
        insertion << fixed << time_taken_1 << setprecision(5) << ",";

        cout << "(Selection = ";
        //outfile << "(Selection = ";
        t = clock();
        selectionSort(arr_s, randomNumber_len, 4);
        t = clock() - t;
        time_taken_2 = ((double)t);
```

```
        cout << fixed << time_taken_2 << setprecision(5) << "sec)\t";
        selection << fixed << time_taken_2 << setprecision(5) << ",";

        cout << "(Quick = ";
        // outfile << "(Quick = ";
        t = clock();
        quickSort(arr_q, 0, randomNumber_len - 1);
        t = clock() - t;
        time_taken_4 = ((double)t);
        cout << fixed << time_taken_4 << setprecision(5) << "sec)\t";
        quick << fixed << time_taken_4 << setprecision(5) << ",";

        cout << "(Bubble = ";
        //outfile << "(Bubble = ";
        t = clock();
        bubbleSort(arr_b, randomNumber_len, 4);
        t = clock() - t;
        time_taken_3 = ((double)t);
        cout << fixed << time_taken_3 << setprecision(5) << "sec)\t\n\n";
        bubble << fixed << time_taken_3 << setprecision(5) << ",";

        reverseArray(arr, randomNumber_len);

        copy_n(arr, randomNumber_len, arr_b);
        copy_n(arr, randomNumber_len, arr_s);
        copy_n(arr, randomNumber_len, arr_q);

        cout << "Seq Type = Reverse Sorted\n  ";

        cout << "(Insertion = ";
        //outfile << "(Insertion = ";
        t = clock();
        insertionSort(arr, randomNumber_len, 1);
        t = clock() - t;
        time_taken_1 = ((double)t);
        cout << fixed << time_taken_1 << setprecision(5) << "sec)\t";
        insertion << fixed << time_taken_1 << setprecision(5) << "\n";

        cout << "(Selection = ";
        //outfile << "(Selection = ";
        t = clock();
        selectionSort(arr_s, randomNumber_len, 4);
        t = clock() - t;
        time_taken_2 = ((double)t);
        cout << fixed << time_taken_2 << setprecision(5) << "sec)\t";
        selection << fixed << time_taken_2 << setprecision(5) << "\n";

        cout << "(Quick = ";
        //outfile << "(Quick = ";
        t = clock();
        quickSort(arr_q, 0, randomNumber_len - 1);
        t = clock() - t;
        time_taken_4 = ((double)t);
        cout << fixed << time_taken_4 << setprecision(5) << "sec)\t";
        quick << fixed << time_taken_4 << setprecision(5) << "\n";
```

```
        cout << "(Bubble = ";
        //outfile << "(Bubble = ";
        t = clock();
        bubbleSort(arr_b, randomNumber_len, 4);
        t = clock() - t;
        time_taken_3 = ((double)t);
        cout << fixed << time_taken_3 << setprecision(5) << "sec)\t\n\n";
        bubble << fixed << time_taken_3 << setprecision(5) << "\n";
    }

    insertion.close();
    selection.close();
    bubble.close();
    quick.close();

    return 0;
}
```

OBSERVATIONS

1. Bubble Sort

Bubble Sort repeatedly compares and swaps adjacent elements in every pass. In **i-th pass** of Bubble Sort (ascending order), **last (i-1) elements are already sorted**, and i-th largest element is placed at (N-i)-th position.

- **Best Case** Sorted array as input. Or almost all elements are in proper place.
- **Worst Case**: Reversely sorted / Very few elements are in proper place (Random Order) **$O(N^2)$** swaps.
- It is the simplest sorting approach.
- Bubble sort is comparatively slower algorithm.

2. Selection Sort

Selection sort selects i-th smallest element and places at i-th position. This algorithm divides the array into two parts: sorted (left) and unsorted (right) subarray. It selects the smallest element from unsorted subarray and places in the first position of that subarray (ascending order). It repeatedly selects the next smallest element.

- In Selection Sort when the inputs are in sorted order or in random order the time take by algorithm is almost same.
- **Worst Case**: Reversely sorted, and when the inner loop makes a maximum comparison.
- Selection Sort is **Inplace** Sorting algorithm**.**
- There is no best case scenario in Selection sort.

3. Insertion Sort

Insertion Sort is a simple comparison based sorting algorithm. It inserts every array element into its proper position. In i-th iteration, previous (i-1) elements are already sorted, and the i-th element (Arr[i]) is inserted into its proper place in the previously sorted subarray.

- Best case complexity is of **O(N)** while the array is already sorted.
- **Worst Case**: In case of reverse sorted inputs it gives worst case.
- For better efficiency we must use Insertion Sort when there is smaller inputs interm of length.
- Insertion Sort is also **Inplace** Sorting algorithm.

4. Quick Sort

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Pick first element as pivot.
2. Pick last element as pivot (implemented here)
3. Pick a random element as pivot.
4. Pick median as pivot.

- In quick sort we get the best case when the inputs are in random order.
- Quick Sort is faster in case of smaller inputs.