Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Software Engineering Department

Computer Programming

Laboratory work #2

# One-Dimensional Array Operations and Processing

*Author:*
Alexandru Rudoi
std. gr. FAF-231

*Verified:*
Alexandru Furdui

Chișinău 2023

# Theory Background

Before implementing any sorting algorithms, it's crucial to have a solid understanding of how each algorithm works. So, here's a brief overview of the research and concepts which I have explored:

**Bubble Sort:** Bubble Sort repeatedly compares adjacent elements and swaps them if they are in the wrong order. This process continues until the entire array is sorted. *Time complexity: $O(n^2)$ in the worst case.*

**Selection Sort:** Selection Sort divides the array into two parts: a sorted part and an unsorted part. It repeatedly selects the smallest element from the unsorted part and moves it to the end of the sorted part. This process continues until the entire array is sorted. *Time complexity: $O(n^2)$ in the worst case.*

**Insertion Sort:** Insertion Sort divides the array into two parts: a sorted part and an unsorted part. It iterates through the unsorted part, taking one element at a time and inserting it into its correct position within the sorted part. This process continues until the entire array is sorted. *Time complexity: $O(n^2)$ in the worst case.*

**Quick Sort:** Quick Sort selects a pivot element from the array and partitions the array into two sub-arrays: one with elements less than the pivot and one with elements greater than the pivot. Quick Sort recursively applies this partitioning process to the sub-arrays until the entire array is sorted. *Time complexity: $O(n^2)$ in the worst case, but typically $O(n \log n)$ on average.*

## 0.1 Key Concepts and Notions

**Time Complexity:** Measure of how the runtime of an algorithm grows as the input size increases.

**Comparison-Based Sorting:** Sorting algorithms that rely on comparing elements to determine their order.

**Stable Sorting:** Sorting that maintains the relative order of equal elements.

**In-Place Sorting:** Sorting that rearranges elements within the original array without using additional memory.

**Divide and Conquer:** Algorithmic strategy that breaks a problem into smaller sub-problems and combines their solutions.

**Pivot:** A chosen element used to partition data in Quick Sort.

**Efficiency:** Evaluation of how well an algorithm performs in terms of time and space usage.

## The Task

You will have to read from console an array of numbers(must be int) of length n(also from console), and implement these sorting algorithms, and explain them:

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Quick Sort

## Technical implementation

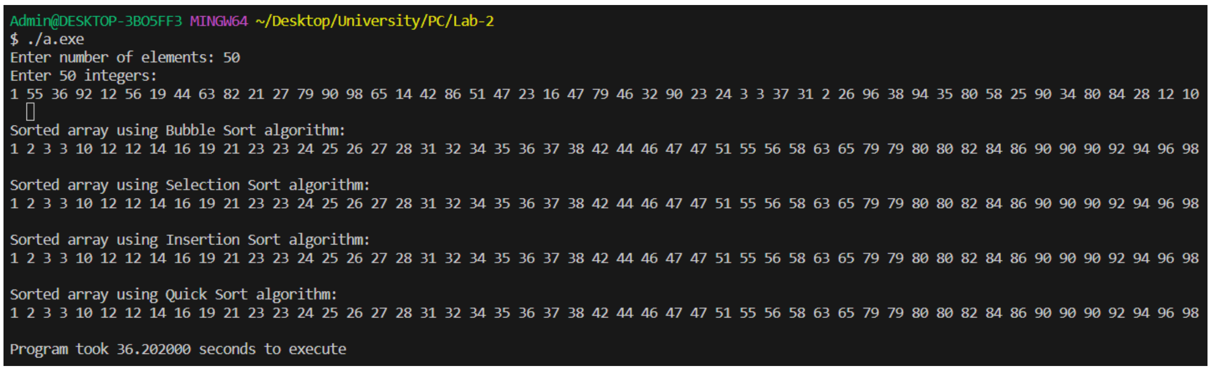
Listing of the program: Github
Pseudo-code:

```
Start Timer

Read n  // Number of elements

Declare arr[n], arr2[n]  // Create two arrays of size n

For i = 0 to n-1:
    Read arr[i]  // Read n integers into the first array

// Bubble Sort
Copy arr into arr2
BubbleSort(arr2, n)
Print "Sorted␣array␣using␣Bubble␣Sort␣algorithm:"
Print arr2

// Selection Sort
Copy arr into arr2
SelectionSort(arr2, n)
Print "Sorted␣array␣using␣Selection␣Sort␣algorithm:"
Print arr2

// Insertion Sort
Copy arr into arr2
InsertionSort(arr2, n)
```

```
Print "Sorted␣array␣using␣Insertion␣Sort␣algorithm:"
Print arr2

// Quick Sort
Copy arr into arr2
QuickSort(arr2, 0, n-1)
Print "Sorted␣array␣using␣Quick␣Sort␣algorithm:"
Print arr2

End Timer

Calculate elapsed time

Print "Elapsed␣time:" + elapsed time + "seconds"
```

## Results

In the next picture, it can be seen that the program successfully sorted a list of 50 integers in ascending order using four different sorting algorithms: Bubble Sort, Selection Sort, Insertion Sort, and Quick Sort. All four algorithms produced identical sorted sequences. The program's execution took approximately 36.202 seconds.



```
Admin@DESKTOP-3BO5FF3 MINGW64 ~/Desktop/University/PC/Lab-2
$ ./a.exe
Enter number of elements: 50
Enter 50 integers:
1 55 36 92 12 56 19 44 63 82 21 27 79 90 98 65 14 42 86 51 47 23 16 47 79 46 32 90 23 24 3 3 37 31 2 26 96 38 94 35 80 58 25 90 34 80 84 28 12 10

Sorted array using Bubble Sort algorithm:
1 2 3 3 10 12 12 14 16 19 21 23 23 24 25 26 27 28 31 32 34 35 36 37 38 42 44 46 47 47 51 55 56 58 63 65 79 79 80 80 82 84 86 90 90 90 92 94 96 98

Sorted array using Selection Sort algorithm:
1 2 3 3 10 12 12 14 16 19 21 23 23 24 25 26 27 28 31 32 34 35 36 37 38 42 44 46 47 47 51 55 56 58 63 65 79 79 80 80 82 84 86 90 90 90 92 94 96 98

Sorted array using Insertion Sort algorithm:
1 2 3 3 10 12 12 14 16 19 21 23 23 24 25 26 27 28 31 32 34 35 36 37 38 42 44 46 47 47 51 55 56 58 63 65 79 79 80 80 82 84 86 90 90 90 92 94 96 98

Sorted array using Quick Sort algorithm:
1 2 3 3 10 12 12 14 16 19 21 23 23 24 25 26 27 28 31 32 34 35 36 37 38 42 44 46 47 47 51 55 56 58 63 65 79 79 80 80 82 84 86 90 90 90 92 94 96 98

Program took 36.202000 seconds to execute
```

## Conclusion

In summary, working on these sorting algorithms was a significant step in my journey toward becoming a future software engineer. It deepened my understanding of efficient code and expanded my knowledge in algorithmic thinking. This experience has equipped me with essential skills and insights that will undoubtedly benefit my future in software engineering.

## Bibliography

1. The overview of Computer Programming course lessons for students (lecturer: associate professor M. Kulev). Chisinau, UTM, FCIM, 2023.
2. https://www.programiz.com/dsa/bubble-sort
3. https://www.geeksforgeeks.org/selection-sort/
4. https://www.javatpoint.com/quick-sort
5. https://www.khanacademy.org/computing/computer-science/algorithms/insertion-sort/a/insertion-sort