# Chapter. I. General Overview of the Topic

Sorting data is one of the most frequently used techniques in programming. It finds applications in a wide range of fields, from mathematics (in mathematical statistics) to linguistics (for creating dictionaries). Therefore, it is important to identify the most efficient algorithms and present both their advantages and disadvantages. We will focus on the most recognized and important ones.

Sorting methods can be classified into two categories: direct methods and advanced methods.

Direct methods are based on simple algorithms that are easy to understand and implement. These include selection sort (SelectSort), insertion sort (InsertSort), and bubble sort (BubbleSort).

Advanced methods involve more complex algorithms but do not require advanced knowledge of algorithms. The most well-known among them include quicksort (QuickSort), merge sort (MergeSort), and heap sort (HeapSort).

# Chapter II. Required Hardware and Software Resources

A minimum system configuration for running this program efficiently is:

Display: Super VGA, 256 colors, minimum resolution 800 x 600

HDD Capacity (GB): 165 MB available space for a typical installation. (165 MB max); 20 MB additional space - special features

Additional space: 44 MB for SQL Server 2000 Desktop Engine (MSDE) optional

Memory (MB): 64 MB RAM; 128 MB or more recommended

Processor Frequency: PC with a Pentium processor

Operating System: Windows 2000 with SP2 or a newer version

Drive: Optical CD-ROM drive

Software: C++

# Chapter III. Direct Sorting Methods

## **3.1 BubbleSort**

This method consists of comparing each element with the others and swapping them if the larger element has a lower index. It is the simplest sorting method and does not require detailed knowledge of programming. It can be successfully used by beginners.

Bubble Sort is a simple sorting technique, efficient for a small number of elements (less than 15), but not for large tables. It does not require much memory, but it is twice as slow as InsertionSort in almost any situation. The execution time depends on the input, meaning the initial order of elements.

If the table is already sorted, only one step is necessary, i.e., N-1 comparisons. In the worst case, N × (N-1)/2 comparisons and N × (N-1)/2 swaps are required. The algorithm's performance in the general case is difficult to analyze but is similar to the worst case. The time complexity for Bubble Sort is O(N^2).

BubbleSort does not use other data structures, making it recommendable in cases with few elements to sort and in situations where memory is limited.

## **3.2 SelectionSort**

Selection Sort is one of the simplest sorting methods and works very well for small tables, where each element (record) is moved at most once. The implementation of the algorithm is simple. At each step "i", the minimum element among a[i+1], a[i+2]... a[n] is found and swapped with a[i].

Most time is spent searching for the minimum element in the unsorted part of the table. The search is performed from left to right. During each step, only one swap is needed, so the total number of swaps for N elements is: N-1.

The time complexity for Selection Sort is O(N^2).

## **3.3 InsertionSort**

Direct insertion is part of the sorting techniques family based on the "bridge player" method. It is an algorithm almost as simple as Selection Sort but can be more flexible. In this algorithm, elements A[1]...A[i-1] are considered sorted, and then element A[i] is inserted into its correct place.

If we have a table A with N unsorted elements, we traverse the table and insert each element into its correct position among the sorted elements. For each i = 2..N, elements A[1]...A[i] are sorted by inserting A[i] among the sorted list: A[1]...A[i-1]. The elements to the left of the index are sorted but not in their final position. The table is completely sorted when the index reaches the right end of the table.

Insertion sort is a simple, linear sorting algorithm: O(N) for tables containing N nearly sorted elements. The algorithm's execution time depends on the number of inversions, thus on the initial order of elements. Generally, (N^2)/4 comparisons and swaps, (N^2)/8, are needed, making the order of magnitude O(N^2).

## **3.4 Interclasarea Vectorilor**

Although it does not follow the same principle as the others, merging two vectors is a sorting method. This is done by merging two already sorted vectors into a third vector. The first element is chosen from each vector, and the smallest one is inserted into the third vector. The index of the vector with the smaller element is incremented and compared until one of the two vectors is exhausted. Once one vector is finished, the remaining elements in the other vector are added in order, without requiring further comparison.

# Chapter IV. Advanced Sorting Methods

## **4.1 MergeSort**

Merge sort is based on the following idea: to sort a table with N elements, we divide it into two sub-tables, sort them separately, and then merge them. It is a sorting method that utilizes the fundamental "divide and conquer" strategy, where the problem is divided into two subproblems of the same type, and after solving them, the results are combined. The algorithm sorts elements in ascending order. The table is initially divided into n/2, then the sub-tables are recursively divided in half, continuing this process until the formed sub-tables have at most k elements (in our case, k = 2, as it is easiest to compare two elements).

The time complexity for Merge Sort is O(N \* log N).

## **4.2 QuickSort**

In practice, the fastest sorting algorithm is Quicksort, also known as quick sorting, which uses partitioning as its core concept. It is faster than any other simple sorting method, performing well for large files or tables but inefficient for small ones.

The basic strategy used is "divide and conquer," as it is easier to sort two small tables than one large one. The algorithm is easy to implement, works well in various situations, and consumes fewer resources than any other sorting method in many cases. It requires approximately (N \* log N) operations in the general case to sort N elements.

The Quicksort method involves finding the final position occupied by the element at the first position by comparing it with elements on the other side of the array. This process is repeated until the partition has only one element left.

According to the algorithm, each element is compared with the pivot, which represents an O(N) operation. The table is then split into two parts, each part being further divided into two. If each part is approximately halved, this results in (log 2N) splits. Thus, the average-case execution time of Quicksort is O(N log 2N), while in the worst case, it is O(N^2).

Quicksort is generally efficient but not in the worst case, where using three partitioning indices is recommended. Randomization is an important and useful concept, serving as a general tool for improving the algorithm. However, Quicksort is sensitive to the order of input data and is not a stable algorithm.

The main disadvantage of this algorithm is its recursive nature, requiring approximately O(N^2) operations in the worst case. It is fragile, and a small mistake in its implementation may lead to incorrect execution for certain datasets.

## **4.3 HeapSort**

Heap Sort is a comparison-based sorting technique founded on the binary heap data structure. It is similar to selection sort, where we first find the minimum element and place it at the beginning. We repeat the same process for the remaining elements.

First, convert the array structure into a heap data structure using heapify, then delete the root node of the Max-heap one by one and replace it with the last node in the heap, followed by heapifying the heap's root. Repeat this process until the heap size is greater than 1.

# Cap. V. Bibliography

1. Mariana Milosescu: Informatica, manual pentru clasa a IX-a, C++ profil real intensiv, specializarea matematica-informatică , Editura: Didactică și Pedagogica, 2005
2. Mariana Milosescu: Informatica, manual pentru clasa a X-a, C++ profil real intensiv, specializarea matematica-informatică , Editura: Didactică și Pedagogica, 2019
3. Mariana Milosescu: Informatica, manual pentru clasa a XI-a, C++- profil real intensiv, specializarea matematica-informatica, Editura: Didactica si Pedagogica, 2019
4. <https://sortari.weebly.com/>
5. <https://www.programiz.com>
6. <https://www.geeksforgeeks.org>
7. <https://www.w3schools.com>

**CUPRINS**

[Chapter. I. General Overview of the Topic 1](#_Toc192930606)

[Chapter II. Required Hardware and Software Resources 2](#_Toc192930607)

[Chapter III. Direct Sorting Methods 3](#_Toc192930608)

[**3.1 BubbleSort** 3](#_Toc192930609)

[**3.2 SelectionSort** 3](#_Toc192930610)

[**3.3 InsertionSort** 4](#_Toc192930611)

[**3.4 Interclasarea Vectorilor** 4](#_Toc192930612)

[Chapter IV. Advanced Sorting Methods 5](#_Toc192930613)

[**4.1 MergeSort** 5](#_Toc192930614)

[**4.2 QuickSort** 5](#_Toc192930615)

[**4.3 HeapSort** 6](#_Toc192930616)

[Cap. V. Bibliography 7](#_Toc192930617)