Ministry of Education, Culture and Research of the Republic of Moldova

Technical University of Moldova

Department of Software and Automation Engineering

**REPORT**

Laboratory work No. 2

Discipline: Algorithms’ Analysis

Topic: Study and empirical analysis of sorting algorithms

Elaborated: Sclifos Tudor,

student group FAF-211

Verified: Fiștic Cristofor,

university assistant

Chișinău – 2023

Algorithm analysis

**Objective:**

Study and analyze different sorting algorithms

**Tasks:**

1. Implement 4 sorting algorithms

2. Decide properties of input format that will be used for algorithm analysis;

3. Decide the comparison metric for the algorithms;

4. Analyze empirically the algorithms;

5. Present the results of the obtained data;

6. Deduce conclusions of the laboratory.

**Theoretical Notes:**

An alternative to mathematical analysis of complexity is empirical analysis.

This may be useful for: obtaining preliminary information on the complexity class of an algorithm; comparing the efficiency of two (or more) algorithms for solving the same problems; comparing the efficiency of several implementations of the same algorithm; obtaining information on the efficiency of implementing an algorithm on a particular computer.

**Introduction:**

Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order.

**Comparison metric:**

The comparison metric for this laboratory work will be considered the time of execution of each algorithm (T(n))

**Input format:**

As input, each algorithm will receive a random array of 100000 numbers

**IMPLEMENTATION**

All four algorithms will be implemented in python an analyzed empirically based on the time required for their completion. The particular efficiency in rapport with input will vary depending on memory of the device used.

**Quick Sort**

Quicksort is a [divide-and-conquer algorithm](https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm). It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. For this reason, it is sometimes called partition-exchange sort. The sub-arrays are then sorted [recursively](https://en.wikipedia.org/wiki/Recursion_(computer_science)). This can be done [in-place](https://en.wikipedia.org/wiki/In-place_algorithm), requiring small additional amounts of [memory](https://en.wikipedia.org/wiki/Main_memory) to perform the sorting.

Quicksort is a [comparison sort](https://en.wikipedia.org/wiki/Comparison_sort), meaning that it can sort items of any type for which a "less-than" relation (formally, a [total order](https://en.wikipedia.org/wiki/Total_order)) is defined. Most implementations of quicksort are not [stable](https://en.wikipedia.org/wiki/Sorting_algorithm#Stability), meaning that the relative order of equal sort items is not preserved.

[Mathematical analysis](https://en.wikipedia.org/wiki/Analysis_of_algorithms) of quicksort shows that, [on average](https://en.wikipedia.org/wiki/Best,_worst_and_average_case), the algorithm takes �(�log⁡�)O(n logn) comparisons to sort *n* items. In the [worst case](https://en.wikipedia.org/wiki/Best,_worst_and_average_case), it makes �(�2)O(n2) comparisons.

**Implementation:**

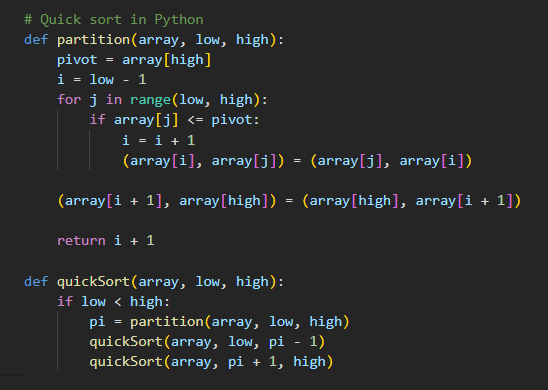


Figure 1. Quick sort implementation

**Results:**

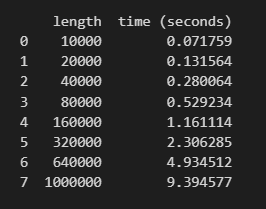


Figure 2. Quick sort result table

**The plot:**

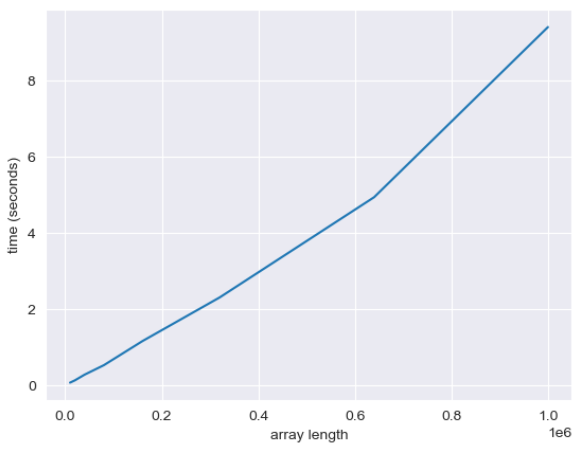


Figure 3. Quick sort plot result

Space complexity: O(1)

Best case: O (n\*logn)

Worst case: O (n\*\*2)

Stable: No

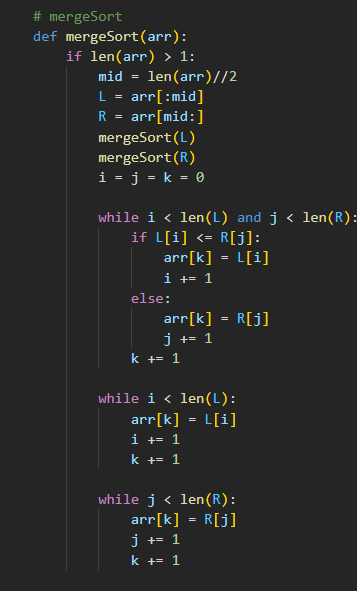
**Merge Sort**

Merge sort is a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array.

Conceptually, a merge sort works as follows:

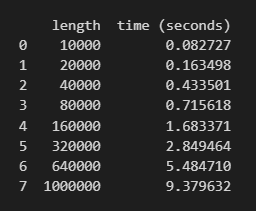
1. Divide the unsorted list into n sublists, each containing one element (a list of one element is considered sorted).
2. Repeatedly merge sublists to produce new sorted sublists until there is only one sublist remaining. This will be the sorted list.

**Implementation:**



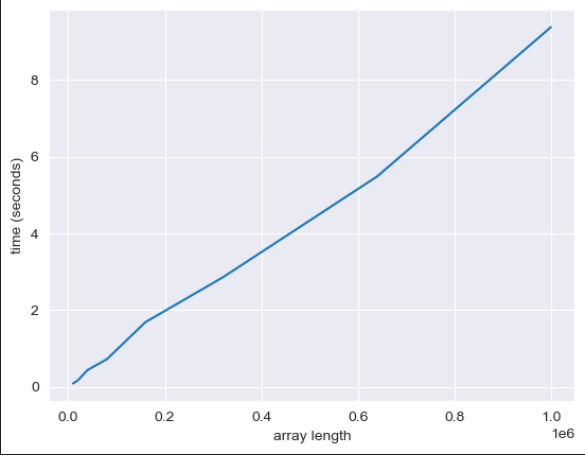
**Figure 4. Merge sort implementation**

**Results:**



**Figure 5. Merge sort result table**

**The plot:**



**Figure 7. Merge sort plot result**

Space complexity: O(n)

Best case: O (n\*logn)

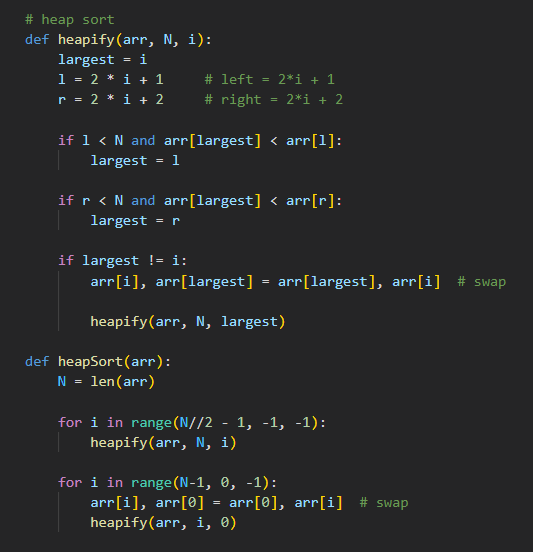
Worst case: O (n\*logn)

Stable: Yes

**Heap Sort**

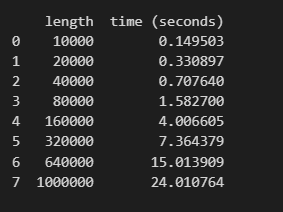
Heapsort is a comparison-based sorting algorithm. It divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element from it and inserting it into the sorted region. Unlike selection sort, heapsort does not waste time with a linear-time scan of the unsorted region; rather, heap sort maintains the unsorted region in a heap data structure to more quickly find the largest element in each step.

**Implementation:**



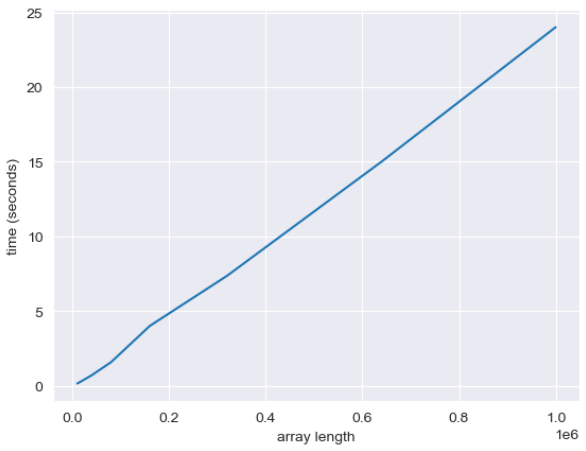
**Figure 8. Heap sort implementation**

**Results:**



**Figure 9. Heap sort table result**

**The plot:**



**Figure 10. Heap sort plot result**

Space complexity: O(n)

Best case: O (n)

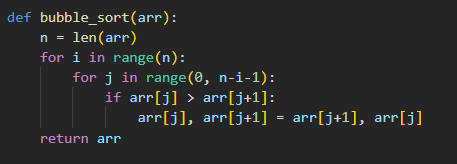
Worst case: O (n\*logn)

Stable: Yes

**Bubble Sort**

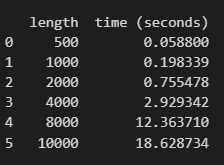
Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly steps through the input list element by element, comparing the current element with the one after it, swapping their values if needed. These passes through the list are repeated until no swaps had to be performed during a pass, meaning that the list has become fully sorted. The algorithm, which is a comparison sort, is named for the way the larger elements "bubble" up to the top of the list.

**Implementation:**



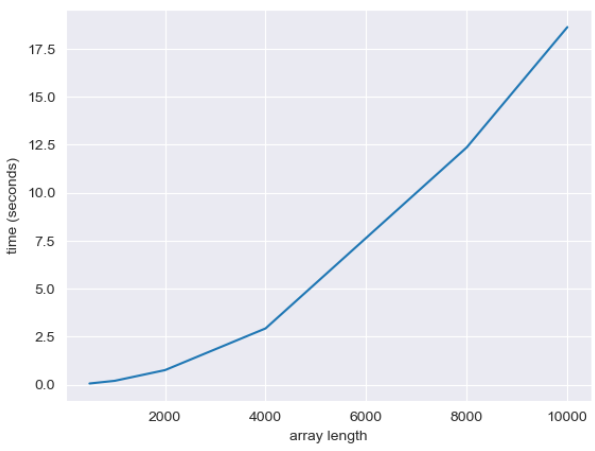
**Figure 11. Bubble sort implementation**

**Results:**



**Figure 12. Bubble sort table result**

**The plot:**



**Figure 13. Bubble sort plot result**

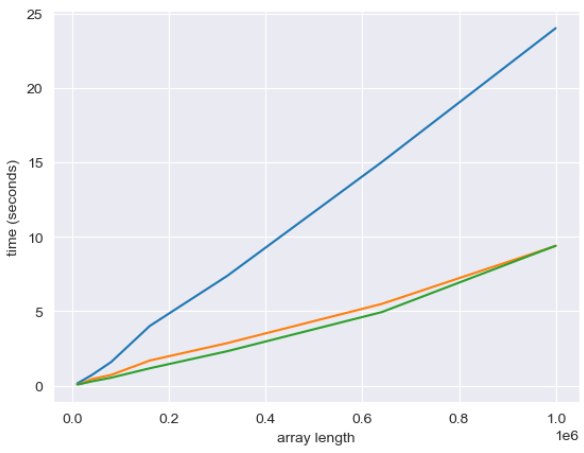
Space complexity: O(1)

Best case: O (n logn)

Worst case: O (n\*n)

Stable: Yes

**Conclusion:**



**Figure 14. All sorting algorithms**

The slowest algorithm turned out to be Heap sort. The fastest is a tie between Quick Sort and Merge Sort.

**Link to GitHub:** [**https://github.com/SexomQ/AlgorithmsAnalysis-labs**](https://github.com/SexomQ/AlgorithmsAnalysis-labs)