MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF THE REPUBLIC OF MOLDOVA

Technical University of Moldova Faculty of Computers, Informatics and Microelectronics Department of Software and Automation Engineering

Postoronca Dumitru FAF-233

Report

Laboratory Work №2

of AA

Checked by:
Fistic Cristofor, *university assistant*
DISA, FCIM, UTM

Chișinău – 2024

Conditions of the Task

Study and empirical analysis of sorting algorithms. Analysis of quickSort, mergeSort, heapSort, one of your choice (Counting Sort)

1. Implement the algorithms listed above in a programming language
2. Establish the properties of the input data against which the analysis is performed
3. Choose metrics for comparing algorithms
4. Perform empirical analysis of the proposed algorithms
5. Make a graphical presentation of the data obtained
6. Make a conclusion on the work done.

Algorithm Implementation

Implementation of the algorithm will be done in Javascript and running in Node environment. In order to reduce the possible fluctuations, all the algorithms will be run in bytecode mode without any optimizations. In this way I ensure that all algorithms are runned in same conditions and are not influences by the particular optimizations applyed by runtime.[1]

Input data

To analyze and compare the sorting algorithms, we define the following input data conditions:

1. Size of the Input (n):
    - Small datasets ($n = 100$)
    - Large datasets ($n = 1{,}000{,}000$)
2. Order of the input Elements:
    - Random ordered elements
    - Elements are already sorted in **ascending order**
    - Elements are already sorted in **descending order**
3. Range of Values:
    - Small Range [1, 100]
    - Large Range [1, 1 000 000]
4. Edge Cases:
    - All elements identical

Metrics in Algorithm analysis

Disabling Turbofan

In order to perform the algorithm analysis I've written Javascript programs that would track the execution time of each algorithm and run them in Node js. Node js uses V8 engine in order to execute the JavaScript programs. This engine consists of 2 components:

- Interpreting machine Ignition
- Optimizing compiler **Turbofan**, that can optimize and translate some of the JavaScript code into the machine code.

To reduce the influence of Turbofan on the validity of data, I have run the algorithms with *–no-opt* flag to disable Turbofan. This decision is motivated by the unpredictability of the optimizing compiler which can optimize the code for an algorithm with a long execution time and not optimize the efficient algorithm. Thus I put all algorithms to equal execution conditions.

Algorithms Implementation

All the algorithms can be found in the GitHub[2] repository of this laboratory work. Each of them corresponds to the most popular implementations in JavaScript that are designed to reduce the influence on call stack and memory

Results

After running of all the tests these are the results obtained:

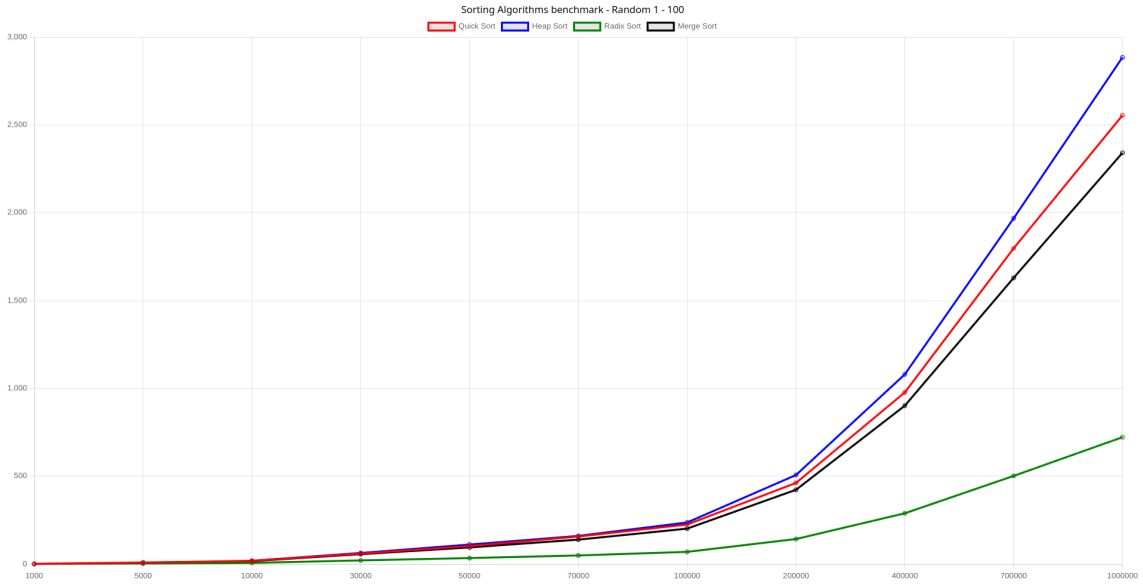

Figure 1: Random numbers from range 1 - 100

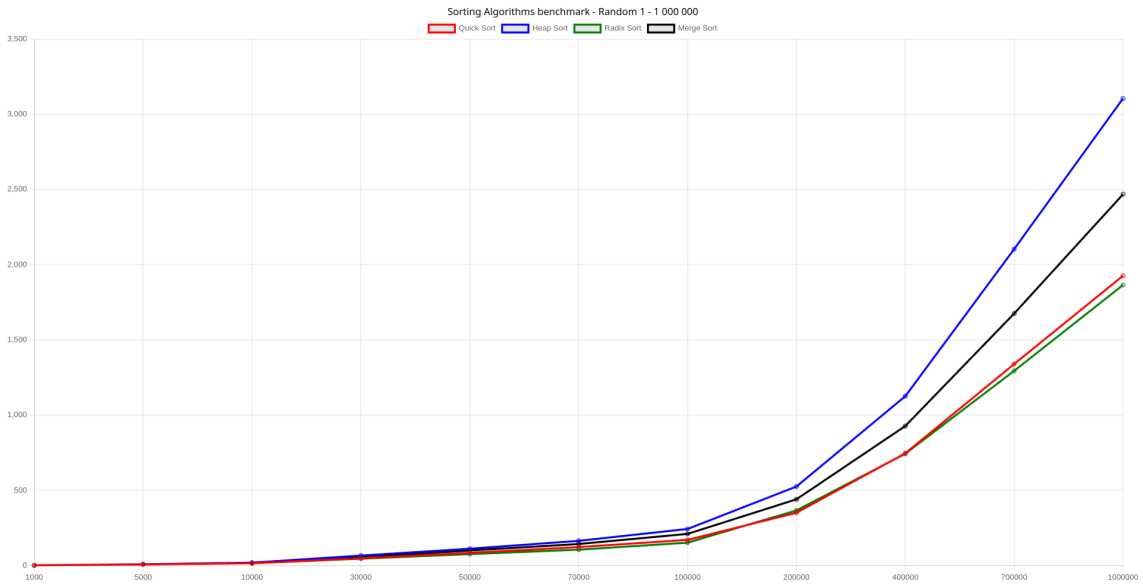

Figure 2: Random numbers from range 1 - 1 000 000

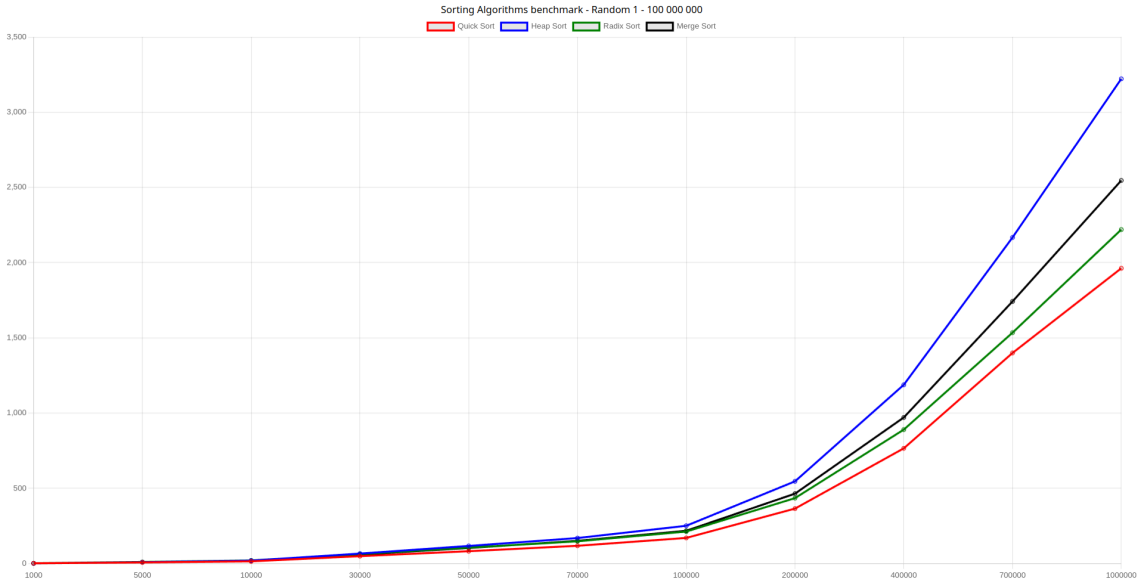

Figure 3: Random numbers from range 1 - 100 000 000

We can easily observe that even if sorting algorithms are performing nearly the same as expected, ***Radix sort*** is vastly influenced by the number of digits implied in the input data. You can see the differences in figure 1 and figure 3. Even if the length of arrays is still the same, Radix sort has to perform editional placements of elements for each aditional digit that the largest number has. This is why it is important to mention that even if performance of Radix sort can be low, it's time execution is highly influenced by the number of digits of the largest number and complexity of the Radix sort is always mentioned as ***O(n*d)***, not O(n)

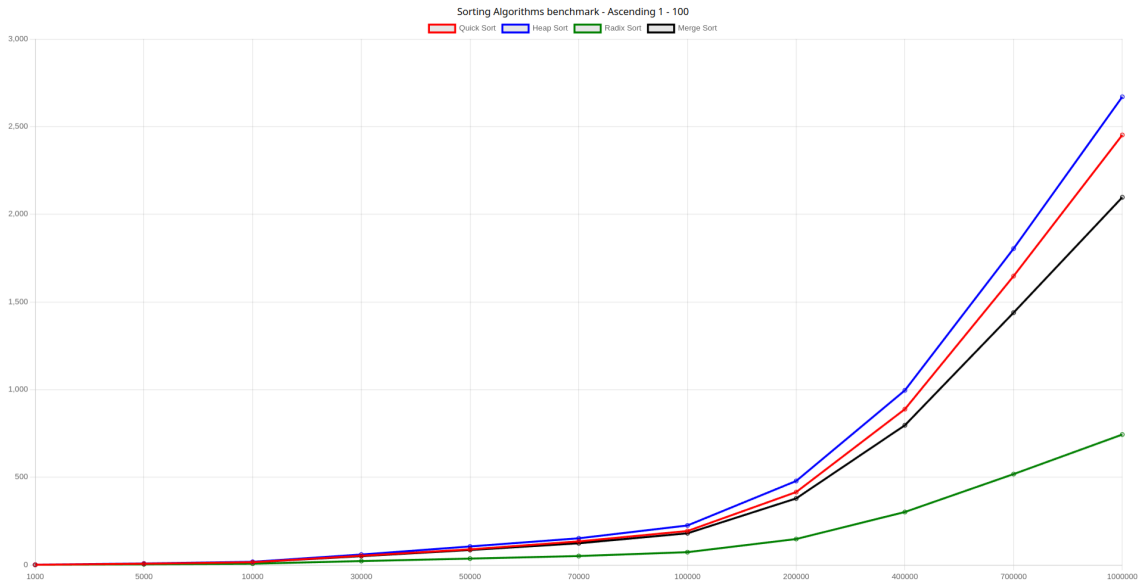

Figure 4: Numbers in ascending order in range 1 - 100

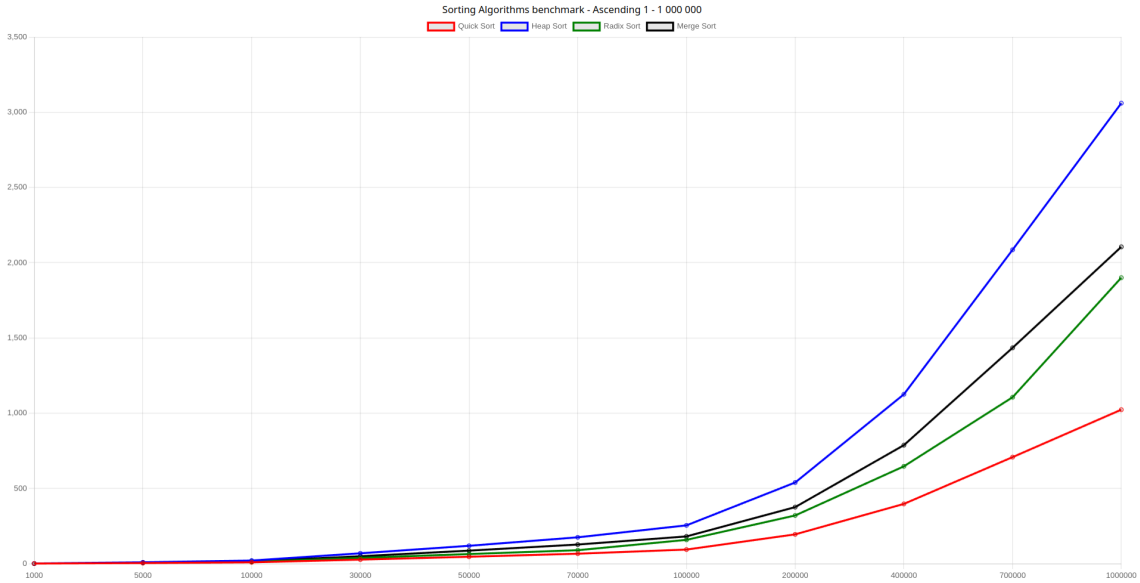

Figure 5: Numbers in ascending order in range 1 - 100 000 000

Graphical Representation

Conclusions

References

[1] Franceska Hinkelman (2017) - Understanding bytecode *Medium*

[2] GitHub repository of current laboratory work