Comparison of the Sorting Algorithms-The Experiment

Claudiu Stefan
Department of Computer Science,
West University of Timisoara,
Timișoara, Romania
Email: stefan.claudiu03@e-uvt.ro

April 2022

Abstract

Sorting algorithms in my opinion are one of the most important algorithms in computer science. They are used in almost every domain, because almost everyone needs to sort a list.

Because some of them are faster than others and some are more memory efficient, the main objective of this paper is to compare the running time of popular sorting algorithms taught and used in Computer Science. This paper is based on a comparison of the following six sorting algorithms: Selection Sort, Quick Sort, Merge Sort, Radix Sort, Bucket Sort, Heap Sort.
For all results in the tables below, the unit of measurement is ms.

Contents

1 Introduction

In the world of sorting algorithms there are three factors that we must always consider, the first is the computer, the second consists of 3 very important parameters (type, order and the number of items in the list) and the third element, which I have discovered as a result of trying to find the best code variant for the tests which I have performed for this experiment, is their implementation.

In order to start with an idea as clear as possible for reading this paper, I put below (Fig.1) a table with the time complexity of the algorithms.

Table 1: Table of Time Complexity

Algorithm	Time Complexity		
	Best	Average	Worst
Selection Sort	$\Omega(n^2)$	$\Omega(n^2)$	$\Omega(n^2)$
Quick Sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$
Merge Sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$
Radix Sort	$\Omega(nk)$	$\Omega(nk)$	$\Omega(nk)$
Bucket Sort	$\Omega(n+k)$	$\Omega(n+k)$	$\Omega(n^2)$
Heap Sort	$\Omega(n \log(n))$	$\Omega(n \log(n))$	$\Omega(n \log(n))$

* Fig.1

2 Algorithms

2.1 Selection Sort

Selection Sort is one of the easiest algorithms and is quite useful to work with small amount of elements. The algorithm divides the input list into two parts the sorted array and the unsorted array. At the beginning the sorted array is empty and the unsorted array is the entire input list. The algorithm works by finding the smallest element in the unsorted array and adding it to the sorted array.

2.2 Quick Sort

Quick sort is a divide and conquer algorithm. It picks an element as pivot and divides the array around the pivot. You have 4 options to choose the pivot, you can choose the first element, the last element, the median element or a random element. The sub-arrays are then sorted recursively.

2.3 Merge Sort

Merge sort is a divide and conquer algorithm as quick sort. It divides the array into 2 sub-arrays, like Quick Sort and does this until the split arrays are only composed of 1 element. After that, the array is merged back together in increasing order.

2.4 Radix Sort

Radix sort is a non-comparative sorting algorithm and it can only sort numbers, but because integers can be used to represent strings, radix sort works on data types other than just integers.

2.5 Bucket Sort

Bucket sort, is a sorting algorithm that works by distributing the elements of an array into a number of buckets. Each bucket is then sorted individually, or by recursively applying the bucket sorting algorithm.

2.6 Heap Sort

Heap sort uses a comparison method based on Binary Heap data structure and is similar to selection sort. Heap Sort creates a heap, which is a special binary tree where the root is either the minimum or the maximum and where the parent node of a leaf is either less or greater than it. The array gets converted to a heap, sorting it in the process.

3 The Experiment

This part is composed, as I said in the introduction, of three components that I consider the most important ones. Let's begin.

3.1 Comparison of Sorting Algorithms

I chose as a first subpoint the comparison of the algorithms to see better how the algorithms are influenced by the parameters. I divided the results into tables, each table having a fixed number of elements in order to see the difference that the type and order of the elements can make.

The first table contains lists of 10,000 items.
In this table I used four types of lists: random numbers list, close numbers list, sorted list, reversed sorted list.

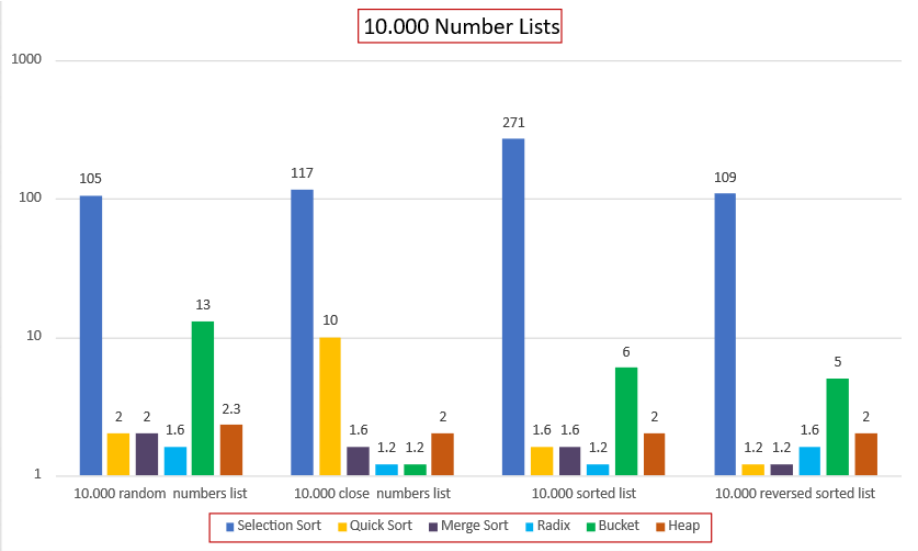

*Fig.2

The second table contains lists of 100,000 items. In this table I used three types of lists: random numbers list, sorted list, reversed sorted list.

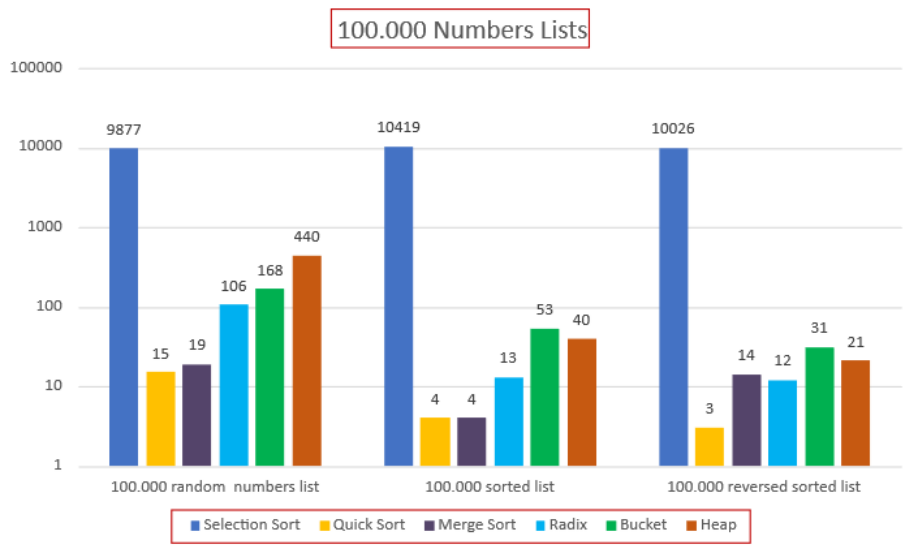

*Fig.3

The third table contains lists of 1 mil items. In this table I used three types of lists: random numbers list, sorted list, reversed sorted list.

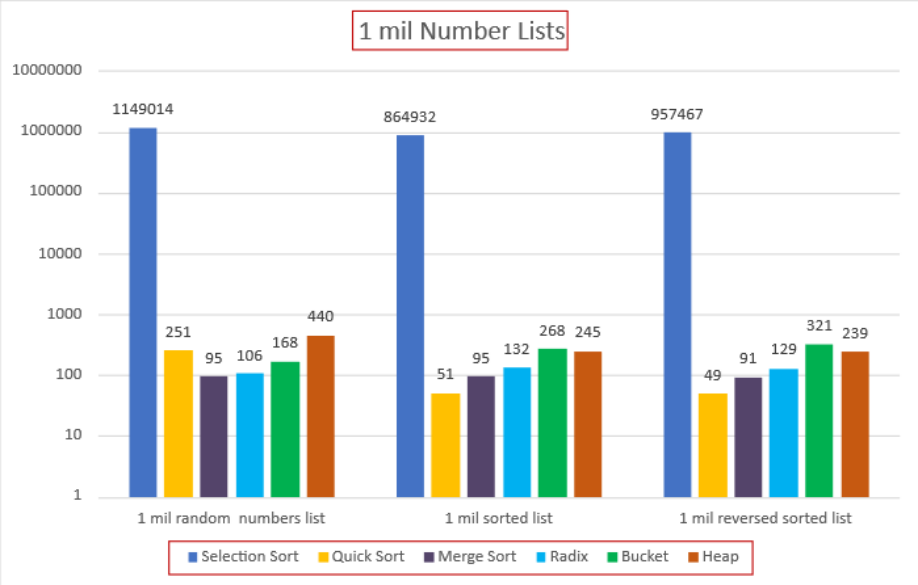

*Fig.4

After these tests, the best algorithm was Quick Sort. After these tests I had some surprises, being impressed by the small time differences between some algorithms. The biggest surprise was Radix Sort, which did extremely well in all the tests, not having very big variations between the different types of lists. Merge Sort works extremely good as well, being in some situations faster than Quick Sort even on large lists of items.

3.2 Performances Comparison

This comparison was conducted in collaboration with Rares Trif. In this part we thought that it would be interesting to test the difference in time between two different computer specs. The time for small numbers it's not that relevant and the difference is hardly seen, so we thought to compare just the 1 million lists.
The first computer has the following configuration (CPU and amount of RAM): Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz, 16 GB RAM.
The second computer has the following configuration (CPU and amount of RAM): Intel(R) Core(TM) i3-3110M CPU @ 2.40GHz, 8 GB RAM.

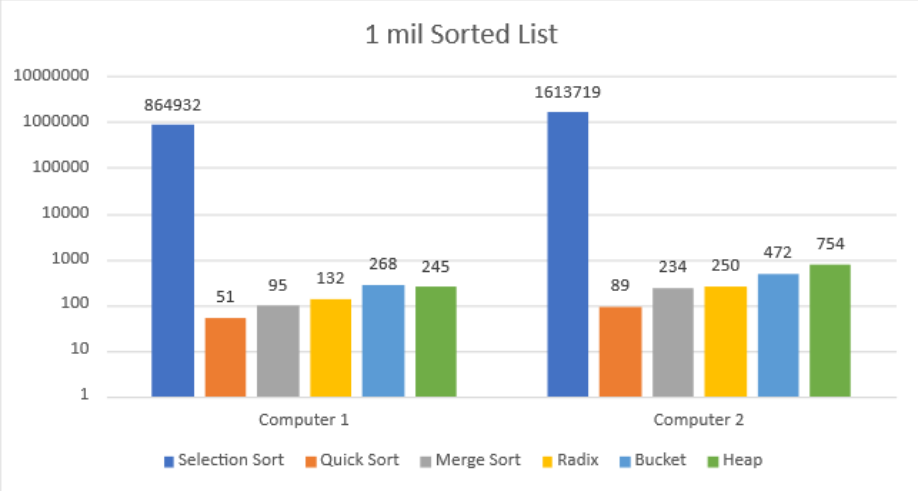

*Fig.5

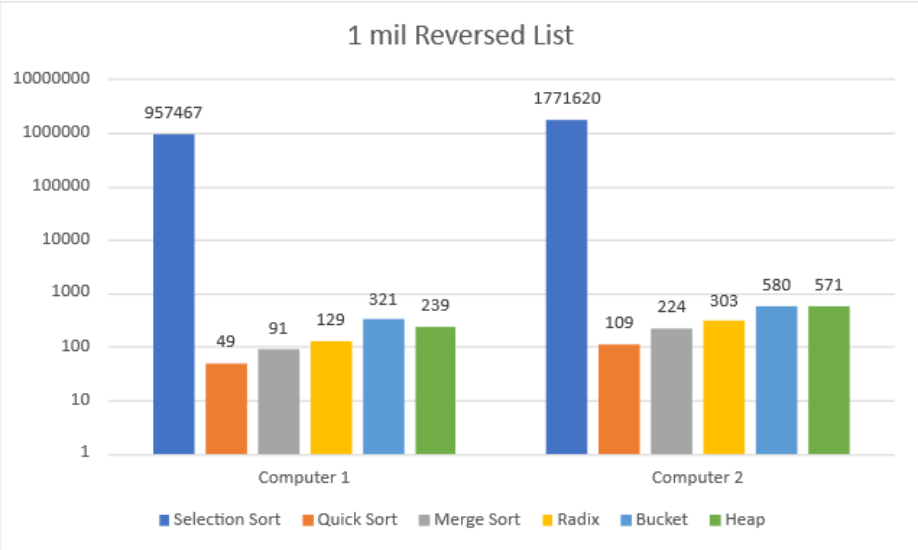

*Fig.6

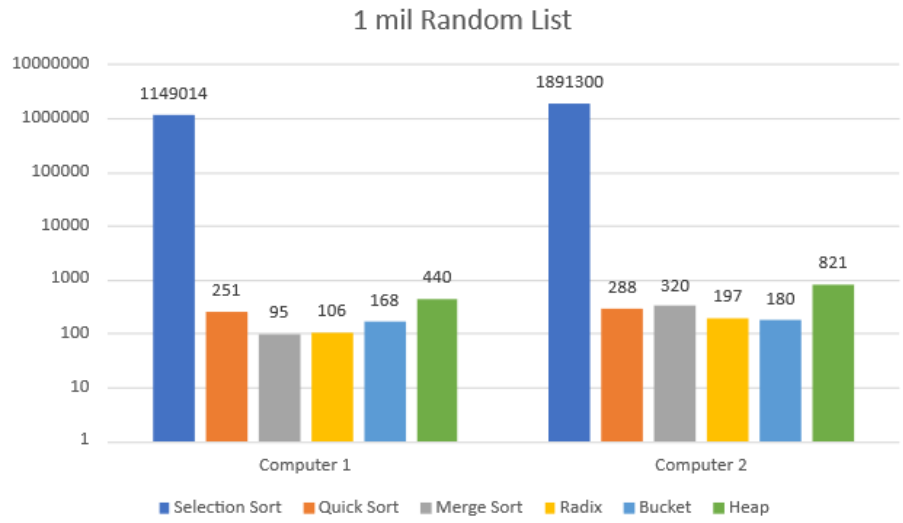

*Fig.7

As expected, the first computer performed better on all tests, being about 50% faster than the second, but that's not why we chose to do this experiment. We did this experiment to see if even the computer used can influence the comparison of algorithms times. As an example we can take the first table in this section (Fig.5) and notice that on the first computer the Heap Sorter gave a better result than the Bucket Sort, while on the second computer the time taken by Heap Sort is visibly longer.

3.3 Code Implementation

While running the tests for the experiment I noticed a very large variation between the tests with a smaller number of elements and those with a larger number.
I started doing research again in order to implement a better code version for Merge Sort and Quick Sort they were the main problem. During the research we found and implemented several ideas in order to optimize these two algorithms until I reached the final ones. After running the tests again I was able to see what a big difference a well-optimized code can make and how much time it can save us. From here we started with the idea that implementation is an extremely important component in the world of sorting algorithms.

4 Conclusion

My conclusion from this experiment is that there are many more things to keep in mind that can influence the choice and use of an algorithm.

5 Bibliography

Github Project
https://github.com/Clau03/MPI

Thomas H. Cormen, Charles E. Leiseron, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms, 3rd Edition*

Geegforgeeks Website
`https://www.geeksforgeeks.org/sorting-algorithms/`

Programming Algorithms
`https://www.programmingalgorithms.com/algorithm/bucket-sort/cpp/`

Programiz website
`https://www.programiz.com/dsa/sorting-algorithm`