A **theoretical** and **experimental** comparison of popular sorting algorithms

Raul-Andrei Ariton
Department of Computer Science
West University of Timișoara

April 23, 2024

Abstract

Aside from it being an assignment for a university course, this paper aims to compare the performance, in terms of time complexity of popular and well-known sorting algorithms among the Computer Science community.

Mainly, however, it is an opportunity for me to put **theory into practice** and confirm that what I have learned so far regarding sorting algorithms is in fact correct.

It is also my first formal paper written, as well as my first paper written in LaTeX.

In this paper I will compare the following sorting algorithms, some of which were taught in the Algorithmics (ADS I) course.

I end this abstract with one of my favorite quotes, that of Gottfried Wilhelm Leibniz, the inventor of one of the first mechanical calculators, the step reckoner:

"... it is beneath the dignity of excellent men to waste their time in calculation when any peasant could do the work just as accurately with the aid of a machine."

– Gottfried Wilhelm Leibniz

Contents

1 The problem of sorting

Before getting into sorting methods / algorithms, I shall formally introduce the problem of sorting in Computer Science. Consider the following problem:

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$
Output: A permutation (reordering) $\langle a_1^{'}, a_2^{'}, \ldots, a_n^{'} \rangle$ of the inputs such that

$$a_1^{'} \preceq a_2^{'} \preceq \ldots \preceq a_n^{'}$$

where $\preceq$ is an order relation (predicate).

The above describes the sorting problem in Computer Science.

1.1 Elements of a sorting problem

A sorting problem may consist of:

- The **record**, the collection of data to be sorted.

  In my theoretical explanations and implementations, the record will be considered an array of numbers.

- The **key**, the value to be sorted, or in other words, the value that determines the reordering of the record.

  In my theoretical explanations and implementations, the keys will be considered numerical values (numbers)

- The **satellite data**, which are values carried around with the key, which do not determine the reordering of the record. In practice, when reordering keys, a sorting algorithm must permute the satellite data along with its respective key.

2 Why is the problem of sorting so important?

Although sorting might seem like a rare problem in Computer Science applications, it is considered the most fundamental problem in the study of algorithms. Here's a few reasons why:

- Many algorithms use sorting as a key subroutine, or require a sorted array as a prerequisite.

  An example of such an algorithm is Binary Search, which has a complexity of $O(\log n)$ and requires a sorted array which speeds things up compared to the well-known Linear Search algorithm, with a complexity of $O(n)$.

- The vast catalog of sorting algorithms that exist today employ a rich set of techniques. In fact, teaching various sorting algorithms also teaches various algorithm design techniques. I will discuss the various techniques used by the sorting algorithms in this present paper in each case.

  In essence, understanding the various techniques used in sorting algorithms, one obtains *an amazing amount of power*, that can be applied to solve other types of problems.

3 The“superior" sorting algorithm

There is no such thing as “the best sorting algorithm”. Many might say Quick Sort is the one, with a complexity of $O(n \log n)$ (*We will notice in our experiments that in some cases Quick Sort is not so reliable*).
However, the best sorting algorithm depends on the situation. Not all sorting problems are the same, therefore there isn’t a “one-for-all” algorithm which should be used each time.

- Bubble Sort is used by some TV providers in order to sort channels based on audience viewing time.
- Merge sort is used on databases, because of the large sets of data, that are too large to be loaded into memory as a whole.
- Quick sort is used to *quickly* sort sports scores in real-time.

4 The Algorithms

This section aims to describe the main idea each sorting algorithm individually, its pseudocode implementation as well as its implementation in the **Python** programming language.

Part I
In-place comparison-based sorting algorithms

4.1 Bubble Sort

Bubble Sort is an in-place, comparison-based sorting algorithm.

Main Idea

The array to be sorted is traversed repeatedly, adjacent elements are compared and swapped if they are out of order. The traversal stops when during a traversal no swapping has taken place (*i.e. when the array becomes sorted*).

Pseudocode

Implementation (in Python)

4.2 Insertion Sort

Insertion Sort is an in-place, comparison based algorithm.

Main Idea

The algorithm starts sorting the elements starting from the right-most element to the left-most element.

Starting with the *second element* in the array (**we consider the first element as sorted**), we compare it with every previous element (*for example, in the first instance, the second element gets compared with the first*) and proceed moving leftwards in the array, until we find an element such that if we place the current element after it, the sorting order will hold.

Pseudocode

Implementation (in Python)

4.3 Selection Sort

Selection Sort is an in-place, comparison based algorithm.

Main Idea

Starting with the first element in the array, for each element at position `i` we search for a **minimum value** in the sub-array `x[i...n]`, where `n` is the size of the array.

If the minimum value found in the sub-array is less than the element at position `i`, we swap the element at position `i` with that minimum value (*hence* `IF arr[x]<arr[i] THEN swap(x[i],x[k])`).

This process is repeated `n-1` times, because after all the iterations, the last element will be **naturally sorted** (*hence* `FOR i := 1 , n-1 DO`).

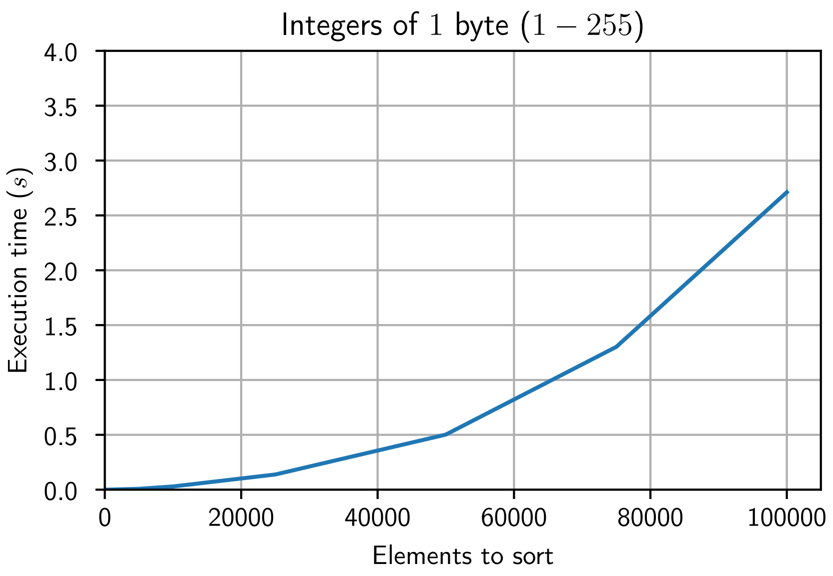
Pseudocode

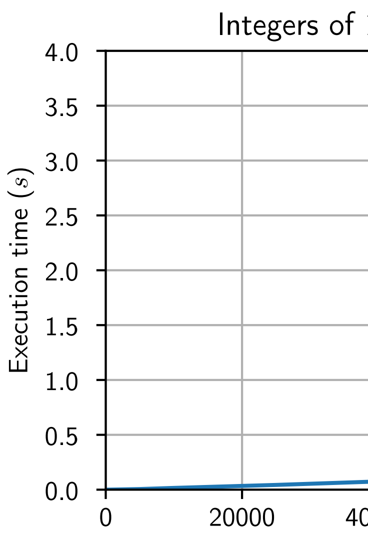
Implementation (in Python)

Part II
Not-in-place comparison-based sorting algorithms

4.4 Quick Sort

Quick Sort is different from what we've encountered so far. It uses the **"divide and conquer"** (also known as the "divide and rule") technique. *Who knew that a policy used in ancient times to maintain power, by great rulers, could be implemented in solving a sorting problem?* Additionally, Quick Sort does not sort in place like all sorting algorithms mentioned so far.





Main Idea

4.5 Merge Sort

Part III
Not comparison-based sorting algorithms

4.6 Counting Sort

4.7 Radix Sort

5 Experimental Comparison

5.1 Methodology

How measurements were made (how execution time was measured), comparisons made to confirm results, make table with comparison times, etc.

5.2 Implementation, in Python

Write the code in Python,