EasyChair Preprint

№ 8030

Sorting Algorithms

Casian Jors

May 22, 2022

Sorting Algorithms

Casian Jors
West University,
Timișoara, Romania
Email: casian.jors02@e-uvt.ro

May 2022

Abstract

In this paper, we will discuss important properties of different sorting techniques including their complexity, stability and memory constraints and then compare their efficiency. Each sorting technique has an advantage in a certain situations, but is innefficient in another. Using raw data, we have experimented with the Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quicksort, Heap Sort, Counting Sort and Radix Sort.

Although the results are different depending on the size and type of the data, the overall conclusion is that Radix Sort is in the majority cases, the most efficient algorithms.

Contents

1 Introduction

Arranging or sorting things or items is not an overnight development. Its footprints can be traced back in 7th century BCE. Abdul Wahab and O.Issa (2009) state that King of Assyria used the idea of arranging clay tablets for Royal Library sorted according to their shape. Sorting is not a jaguar leap but it has emerged in parallel with the development of human mind. In computer science, alphabetizing, arranging, categoring or putting data items in an ordered sequence on the basis of similar properties is called sorting. Sorting is of key importance because it optimizes the usefulness of data. We can observe plenty of sorting examples in our daily life, e.g. we can easily find required items in a shopping maal or utility store because the items are kept categorically. Finding a word from dictionary is not a tideous task because all the words are given in sorted form. Similarly, finding a telephone number, name or address from a telephone directory is also very easy due to the blessings of sorting.

1.1 Motivation

In computer science, sorting is one of the most important fundamental operations because of its pivotal applications. Priority scheduling and shortest job first scheduling are examples of sorting. Thomas Cormen, Charlese Ronald, Clifford Stein and Rivest Leiserson (2001) state "An algorithms is any well-defined computational procedure that takes some value, or set of values, as input and produces some value or set of values as output". A number of sorting algorithms are available with pros and cons. Alfred Aho, John Hopcroft and Jeffrey Ullman (1974) has classified algorithms on the basis of computational complexity, number of swaps, stability, usage of extra resources and recursion . The items to be sorted may be in various forms i.e. random as a whole, already sorted, very small or extremely large in numer, sorted in reverse order etc. There is no algorithm which is best for sorting all types of data.

The algorithms compared by us are the following:

- Bubble Sort,
- Selection Sort,
- Insertion Sort,
- Merge Sort,
- Quicksort,
- Heap Sort,
- Radix Sort.

1.2 Methodology

Empirical comparison is always machine dependent. It is essential to explicitly describe the machine used for experiments in particular to facilitate the researchers who intend to reverify the results. A program developed in Python 3.10 has been used for calculation of CPU time taken by each algorithm. The data set used for this purpose consists of 10-150 in best, worst and average cases. The program was executed on Windows 7 (64 bit). Computer used for this purpose was CPU Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz . Memory installed was 8.00 GB. The results were calculated after tabulation and then their graphical representation was developed using LaTex.

2 The algorithms and their properties

In a theoretical setting, the running times of algorithms can be very similar, they are mostly either $O(n^2)$ or $O(nlog(n))$. However, because of their different approach on memory, they aren't so similar in practice. In this section we will discuss each algorithm one by one. This is necessary for their comparison.

2.1 Bubble Sort

Bubble sort repeatedly compares and swaps(if needed) adjacent elements in every pass. In i-th pass of Bubble Sort (ascending order), last (i-1) elements are already sorted, and i-th largest element is placed at (N-i)-th position, i.e. i-th last position.

Average case time complexity	n^2	
Best case time complexity	n	when data is already sorted
Worst case time complexity	n^2	when data is reverse sorted

2.2 Selection Sort

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

Average case time complexity	n^2
Best case time complexity	n^2
Worst case time complexity	n^2

2.3 Insertion Sort

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and

an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Average case time complexity	n^2
Best case time complexity	n
Worst case time complexity	n^2

2.4 Merge Sort

Merge Sort is a Divide and Conquer algorithm. It divides the input array into two halves, calls itself for the two halves, and then it merges the two sorted halves. The merge() function is used for merging two halves.

Average case time complexity	n log(n)
Best case time complexity	n log(n)
Worst case time complexity	n log(n)

2.5 QuickSort

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

Average case time complexity	n log(n)
Best case time complexity	n log(n)
Worst case time complexity	n^2

2.6 Heap Sort

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

X

Average case time complexity	n log(n)
Best case time complexity	n log(n)
Worst case time complexity	n log(n)

2.7 Radix Sort

In computer science, radix sort is a non-comparative sorting algorithm. It avoids comparison by creating and distributing elements into buckets according to their radix. For elements with more than one significant digit, this bucketing process is repeated for each digit, while preserving the ordering of the prior step, until all digits have been considered. For this reason, radix sort has also been called bucket sort and digital sort.

Radix sort can be applied to data that can be sorted lexicographically, be they integers, words, punch cards, playing cards, or the mail.n.

Average case time complexity	nk
Best case time complexity	nk
Worst case time complexity	nk

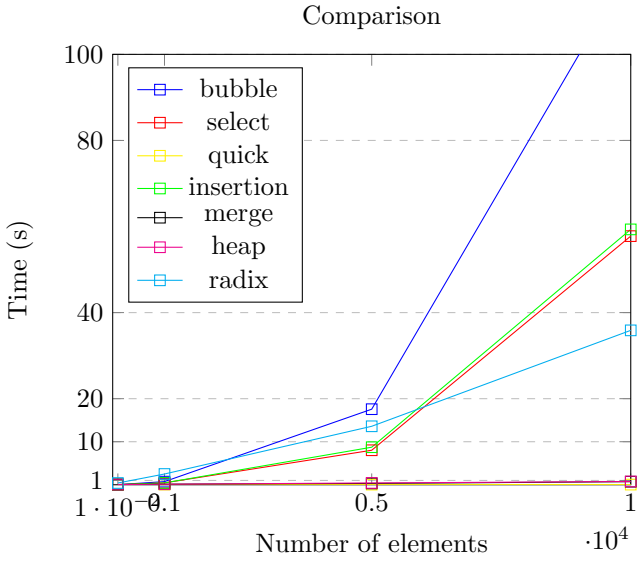
3 The algorithms in practice

3.1 How the code works

The code (see reference) is written in python and uses random and sorting modules to make the code cleaner and easier to read. In the program, each sorting algorithm is tested multiple times for the same amount of elements and the average of the results is calculated for a more accurate result. The number of elements that are tested for each is 100,1000,5000 and 10000.

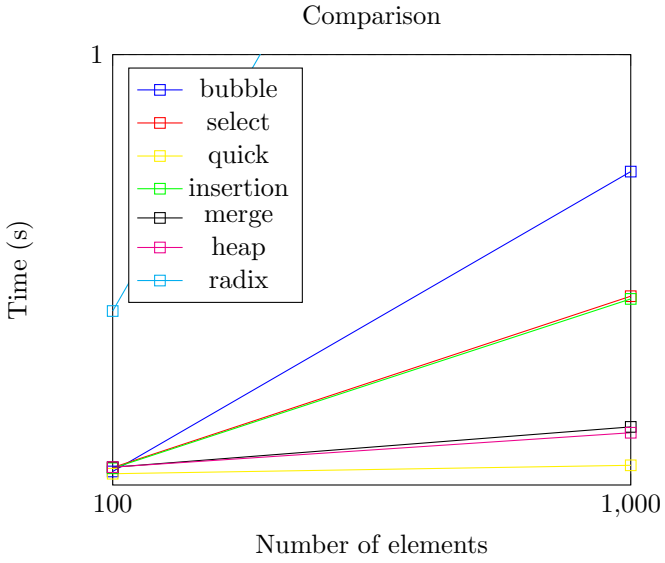
3.2 Comparing all of them

In this graph, you can see the efficiency of the algorithms on a very big scale. The number of elements range from 100 to 10000. The conclusion is that as we initially thought, the algorithms with logarithmic complexity can handle more elements in a significant amount of time.



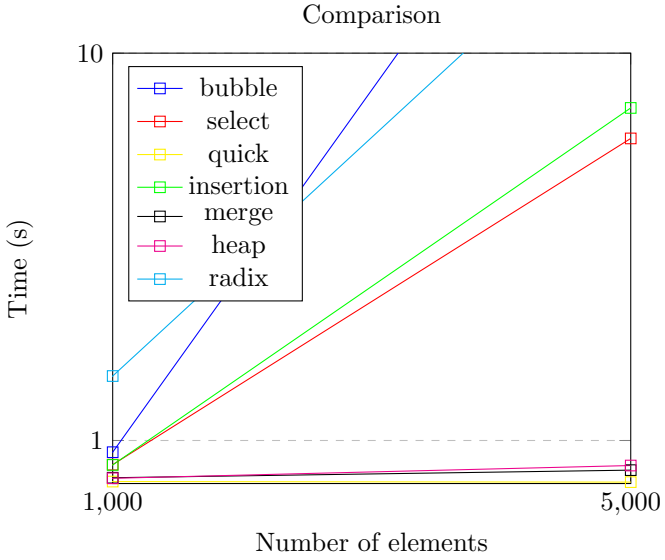
3.3 Comparing on a smaller scale

In this graph, we are offering a closer look for the smaller values. here the algorithms are close but the difference is still noticeable.



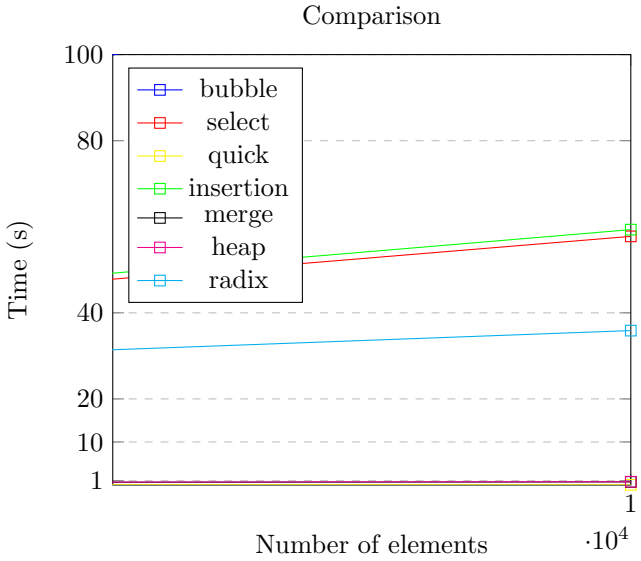
3.4 Comparing on average scale

Here the difference is very clear, Bubble, Radix, Insertion and Select sorting algorithms are a lot less efficient than Heap, Merge and Quicksort.



3.5 Comparing on the highest scale

As we can see, the most efficient algorithms are indeed Heap and Merge sort, followed by Radix.



4 References and repository

- Wahab, A., Issa, O.A. Fundamentals of Library Information Sciences, (1st ed.). Cataloguing-in-Publication Data, Ilorin (2009)
- https://github.com/casistieinfo/MPI$_{P}aper/blob/main/sorting.py$
- https://www.geeksforgeeks.org/sorting-algorithms/?ref=shm