CTA Project

Benchmarking Sorting Algorithms

By Magdalena Malik G00398294

Table of Contents

Introduction.

1.1 Sort idea.

According to the Wikipedia sorting is a process of arranging items systematically. [1] As it is well known, sorting is important part of computing, it can be said that sorting algorithms are one of the most studied algorithms in computer science.

Sorting can be used to solve various of problems, such as:

- Searching for element in a collection
- Selecting chosen element from a collection
- Finding duplicate values in a collection
- Analyzing distribution of items in a collection.[2]

Consequently, a conclusion can be made, that idea of sort can be used in every field of life, from commercial to academic implications and everything between those two.

1.2 Sorting algorithms

According to the previously mentioned definition of sorting, in case of sorting algorithm can be added that here sorting is dictated by early defined rules related to results that are expected.

Also is worth noting that order of sort can be decided in two most common ways:

- lexicographical order (example: “moon” → sorted to → “mnoo”)
- numerical order (example: “1981” → sorted to → “1189”)

Worth noting is fact, that data can be sorted in descending or ascending order.

To achieve properly functioning sorting algorithm, few condition must be fulfilled, such as:

- each element of sorted list must be less than or equal to its successor
- elements will be organized in the way that if list[i] < list[j] then i < j
- sorted list must be a permutation of the elements that originally formed list.[3]

The usefulness of sorting algorithms is highly appreciated, thanks to their help it has been possible to reduce the complexity of many problems, they facilitate the analysis of large data sets, where in the current world of big data effective searching and collecting of data is highly desirable.

1.3 Complexity

During the process of choosing appropriate sorting algorithm, there are various parameters that must be taken into consideration. The most significant are:

- size of data set
- time of processing the data
- computational resources.

1.3.1 Time complexity

As a size of data set is not directly related to complexity, time is. Therefore, time complexity refers to the time that algorithm takes to accomplish task according to the size of inputted data. This relation is denoted as **Order of Growth** with given notation **O(n)** where **O** is order of growth and **n** is the size of the input, it is also called **Big O Notation**. In other words, Big O Notation measures how quickly, a function grows or declines. [4]

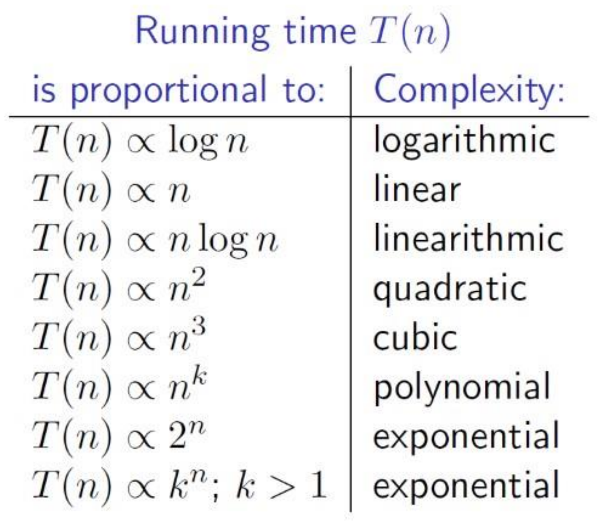
Preferably, Big O Notation is used to describe the complexity of the algorithm in the worst-case scenario. Other words, if algorithm A has less complex Big O Notation than algorithm B, can be said that algorithm A is more efficient.

Worth noting is that to check best- case scenario Omega Notation can be used, whereas Theta Notation can be used to describe the running time for average-case scenario.

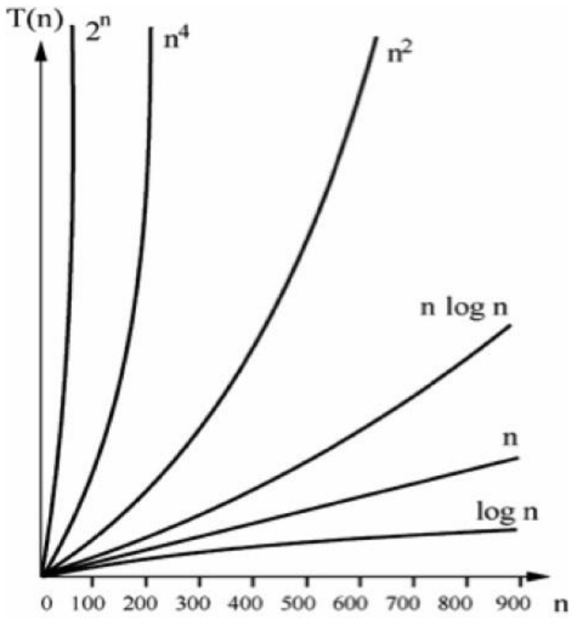
1.3.2 Space complexity

Space complexity is the space or storage that will be used by algorithm. It is directly related to the amount of the input that will be given to algorithm. Importance of space complexity can be explained by the simple fact that the working algorithm can be only useful if it will not outrun the space available for computation.

Worth noting is the fact that space complexity can by also measured by Big O notation. Below is presented diagram of orders of growth, where we can see how input will determinate time.



Running time $T(n)$ is proportional to:	Complexity:
$T(n) \propto \log n$	logarithmic
$T(n) \propto n$	linear
$T(n) \propto n \log n$	linearithmic
$T(n) \propto n^2$	quadratic
$T(n) \propto n^3$	cubic
$T(n) \propto n^k$	polynomial
$T(n) \propto 2^n$	exponential
$T(n) \propto k^n;\ k > 1$	exponential



Performance

1.4.1 Stable Sorting

One of the desirable properties of sorting algorithm is stability. In this case it means, that if a list has equal elements, they will be sorted according to key:

if i<j then final location for list[i] must be to the left of the final location for list[j]. [5]

All algorithms that have that property are considered as stable algorithms, such as:

- Insertion Sort
- Bubble Sort
- Merge Sort
- Counting Sort.

Worth to note, that if algorithm is considered as unstable, sorting elements of equal values might differ from previous result. [6]

1.4.2 In-place sorting

Another desirable property of sorting algorithm is ability to sort in-place. Is efficient way of using minimum of space to perform sorting, preferably swapping elements instead of using extra space.

That behavior greatly saved amount of used memory, with best usage in application that runs in limited memory.

1.4.3 Comparison-base sort

Algorithms which are comparison-based type, using comparison operator to determinate if elements in the list differ from each other. Order is gained by single compare between two elements and result is listed as a first.

Examples of comparison-Based Algorithms:

- Bubble Sort
- Insertion Sort
- Quicksort
- Selection Sort.

It is worth noting that comparison-based algorithms are mostly used in case of diverse input data. Which might be another property worth to consider while choosing proper sorting algorithm.

1.4.4 Non-comparison based sort

Use of a non-comparison algorithm is determinate by previously made assumption about the list that will be sorted. Condition like data which are in some range or has some distribution will not require compering each element, and non-comparison algorithm can be used.

Examples of non-comparison algorithms:

- Bucket Sort
- Radix Sort
- Counting Sort.

Sorting Algorithms

2.1 Bubble Sort

Bubble sort is known as the simplest sorting algorithm. The way of sorting is based on compering each element except the last one. After comparison the highest element is going at the end of the array, and another comparison takes place now without compering two last elements, and process continue till all elements are sorted in expected order.
Time and space complexity for bubble sort:

Name	Best Case	Worst Case	Average Case	Space Complexity	Stable
Bubble Sort	n	n2	n2	1	Yes

As can be noticed for bubble sort worst and average case times are very low, that's mean that bubble sort won't be efficient for sorting big set of data. Worth noting is a fact that bubble sort is in-place sorting algorithm, which means that it is not using extra space to execute sorting task.
Mostly bubble sort can be practical in cases where given set of data is nearly sorted. [7][8]

2.2 Insertion Sort

Insertion Sort is also example of a simple sorting algorithm. Procedure for accomplish task is based on choosing a 'key' which will be the value to which other elements would be compare and moved on the left side of the set if they are greater than a key. At first iteration the 'key' is set on position 1, and during the procedure, the 'key' is moved to another position (2,3,….).

Time and space complexity for insertion sort:

Name	Best Case	Worst Case	Average Case	Space Complexity	Stable
Insertion Sort	n	n2	n2	1	Yes

Clearly can be noticed that time complexity values in insertion sort algorithm case, are the same as in case of bubble sort. Conclusion can be made, that even though the sorting procedure here is different than in bubble sort case, the results and times are inefficient in the same way.
Therefore, usefulness of insertion sorting is mostly implemented in small set of data or data that are pre-sorted. [8][9]

2. 3 Selection Sort

Selection sort is another example of comparison-based sorting algorithm. The procedure used to get sorted data in case of sorting algorithm is based on repeatedly search for the smallest value. Search is taking place in the part, which is unsorted, and each element acknowledged as a smaller is moved to sorted part of the array.

Time and space complexity for selection sort:

Name	Best Case	Worst Case	Average Case	Space Complexity	Stable
Selection Sort	n2	n2	n2	1	No

Here can be noticed that selection sort has best case time worse than bubble sort and insertion sort. Also, it is very inefficient for large data sets. Basic case of selection sort is unstable, although it is worth to noting that stable version can be achieved.

2.4 Quicksort

Quicksort is representant of comparison-based algorithms, as a one of the most efficient from them is vastly used and highly appreciate. Method of working is based on "Divide and Conquer" rule, other words quicksort algorithm is choosing a 'pivot' around which are created two subarrays, where are stored data, smaller than pivot on one side and greater than pivot on the other side accordingly.

Demonstration of efficiency can be seen in complexity table:

Name	Best Case	Worst Case	Average Case	Space Complexity	Stable
Quicksort	n log n	n2	n log n	n	No

Quicksort with its best and average case – n log n, deservedly is described in literature as one of the best. Even though one of desirable properties of good sorting algorithms is absent here – space complexity. [10][11]

2.5 Bucket Sort

Bucket sort also called bin sort, reason of that name is a fact that data from list are distributed in buckets, and then process of sorting occurs on each bucket individually. Bucket sort is an algorithm of non-comparison type. The best case of use is when data are uniformly distributed.

Time and space complexity table:

Name	Best Case	Worst Case	Average Case	Space Complexity	Stable
Bucket Sort	n + k	n2	n + k	n x k	Yes

Performance of bucket sort is dependable on many conditions, for example using this type of sort on an array filled with values which many of them occur close together, will create one big bucket and that will assuredly affect the time of performance. [11][12]

Implementation and Benchmarking

3.1 Implementation

Environment specification:

Specification	
OS	Windows 10 Home, Version 21H2, x64
Device	Alienware 15 R2
Processor	Intel Core i7-6700HQ CPU 2.60 GHz
RAM	16.0 GB
Python	3.8.8, Anaconda 4.10.1

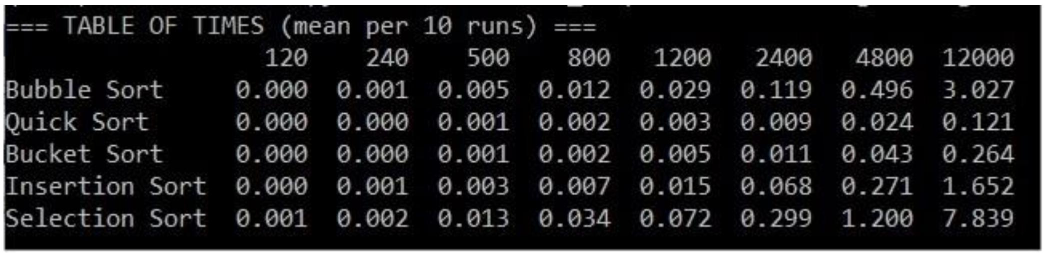
The process of implementation application for chosen algorithms, starts from import of needed library:

- Numpy
- Time
- Random
- Pandas
- Matplotlib.pyplot.

Next step included choice of algorithms and check their behaviours. Chosen algorithms: bubble sort, insertion sort, selection sort, quicksort and bucket sort, were individually tested on small arrays.
Subsequently function to create random array and function to check time were created, which then were implemented on sorting algorithms. Results of times (mean), after each function runs for 10 times, were collected in separate array. Array with time results for each sorting algorithms were then passed to create table and consequently a plot.
Application process is based on randomly generated array of integers in range 100. Array then is passed to sorting functions which are called in *_timing() functions. Times generated after each of 10 runs, are then transform into floating point numbers with 3 digits after coma. Afterwards times are collected in arrays, there is individual array of times for each sorting algorithm.
Visual effect of above process is generated to pandas table and plot – display in prompt and save in file.

3.2 Results of benchmarking

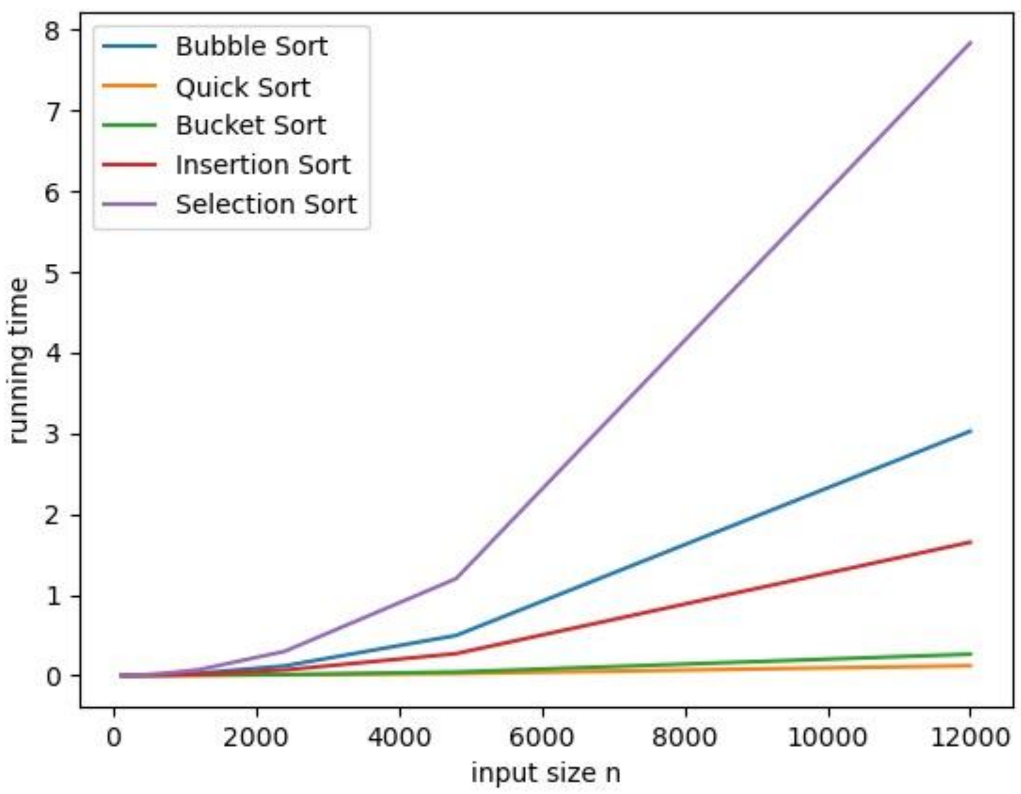
As mentioned earlier, results are stored in array of data and table representation is shown below:



```
=== TABLE OF TIMES (mean per 10 runs) ===
                  120    240    500    800   1200   2400   4800  12000
Bubble Sort     0.000  0.001  0.005  0.012  0.029  0.119  0.496  3.027
Quick Sort      0.000  0.000  0.001  0.002  0.003  0.009  0.024  0.121
Bucket Sort     0.000  0.000  0.001  0.002  0.005  0.011  0.043  0.264
Insertion Sort  0.000  0.001  0.003  0.007  0.015  0.068  0.271  1.652
Selection Sort  0.001  0.002  0.013  0.034  0.072  0.299  1.200  7.839
```

I must add that results shown above might differ from others. That run was made while CPU was in 28% of use and memory on 70% of use.

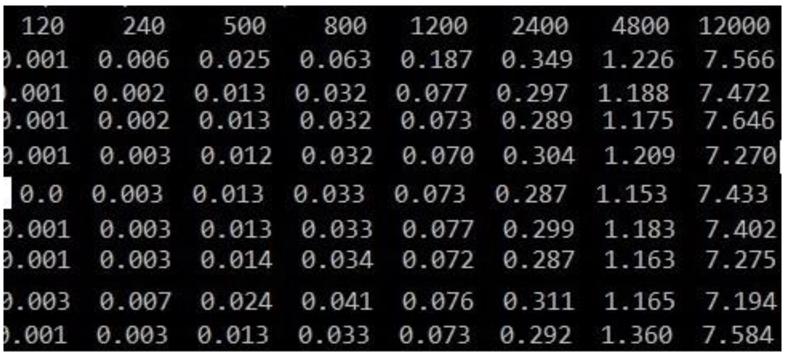
Plot associated with table of times is shown below:



Analysing the table, and the plot clearly can be seen that all algorithms in case of small size of array (sizes: 120, 240), are performing very well. The differences start to be more visible on sizes: 500 and 800, where selection sort and bubble sort, with their times, few times or more bigger than rest of chosen algorithms, become the slowest. In case of the biggest array – 12000, time differences are clear, and time gaps between the fastest (quicksort – 0.121) and the slowest (selection sort – 7.839) became even bigger, in this case 64 times bigger.
Conclusion can be made, that selection sort algorithm had the worst performance overall.

With regards to given time assumption, I can say that overall performance of chosen algorithms did not surprise me. Quicksort algorithm which was defined as the most efficient in case of my test fulfilled expectations. Although selection sort algorithm's performance slightly differs from expectation in relation to two other simple comparison-based algorithms.

More surprising output, I have received after 10 in row runs of application. Each run shows different times, in case of most of the algorithms, their differences were not significant, whereas in case of selection algorithm, difference are more meaning.
Below there is a table of 10 runs – times of selection algorithm.



```
 120    240    500    800    1200   2400   4800   12000
.001   0.006  0.025  0.063  0.187  0.349  1.226  7.566
.001   0.002  0.013  0.032  0.077  0.297  1.188  7.472
.001   0.002  0.013  0.032  0.073  0.289  1.175  7.646
.001   0.003  0.012  0.032  0.070  0.304  1.209  7.270
 0.0   0.003  0.013  0.033  0.073  0.287  1.153  7.433
.001   0.003  0.013  0.033  0.077  0.299  1.183  7.402
.001   0.003  0.014  0.034  0.072  0.287  1.163  7.275
.003   0.007  0.024  0.041  0.076  0.311  1.165  7.194
.001   0.003  0.013  0.033  0.073  0.292  1.360  7.584
```

I am not sure why differences occur, the only conclusion that comes to mind is a fact that there are many factors that can affect algorithm performance. Can only assume that randomly generated array might be one of those factors.

REFERENCES:

1. https://en.wikipedia.org/wiki/Sorting
2. https://realpython.com/sorting-algorithms-python/
3. CTA – 03 Analysing Algorithms Part 1
4. https://en.wikipedia.org/wiki/Time_complexity
5. CTA – 07 Sorting Algorithms Part 1
6. https://stackoverflow.com/questions/1517793/what-is-stability-in-sorting-algorithms-and-why-is-it-important
7. https://www.geeksforgeeks.org/bubble-sort/
8. CTA – 08 Sorting Algorithms Part 2
9. https://www.geeksforgeeks.org/insertion-sort/
10. https://www.geeksforgeeks.org/quick-sort/
11. CTA – 09 Sorting Algorithms Part 3
12. https://www.geeksforgeeks.org/bucket-sort-2/