بسم الله الرحمن الرحيم

Palestine Polytechnic University

College of Information Technology and Computer Engineering

Design and analysis of algorithms

Constructor name:

Faisal khamayseh

Student name:

Islam Alama

Lama Halayka

Lina Alama

Introduction

Most often in programming, when an algorithm is developed to solve a particular problem it is not regarded as a complete success. Often it emerges the question of the performance analysis for the created algorithm: complexity and portability on different systems.

Complexity is the measure by which an algorithm can be evaluated in comparison with other algorithms that solve the same problem. The aim of the complexity assess is to provide an information about the performances of an algorithm before transpose it in a programming language and without considering the particularities set of the incoming data.

Evaluating the complexity of the algorithm will assume the estimation of:

The time complexity:

Number of executions of a certain statement. Usually this statement is the most significant for the particular problem to solve, referred as the base statement.

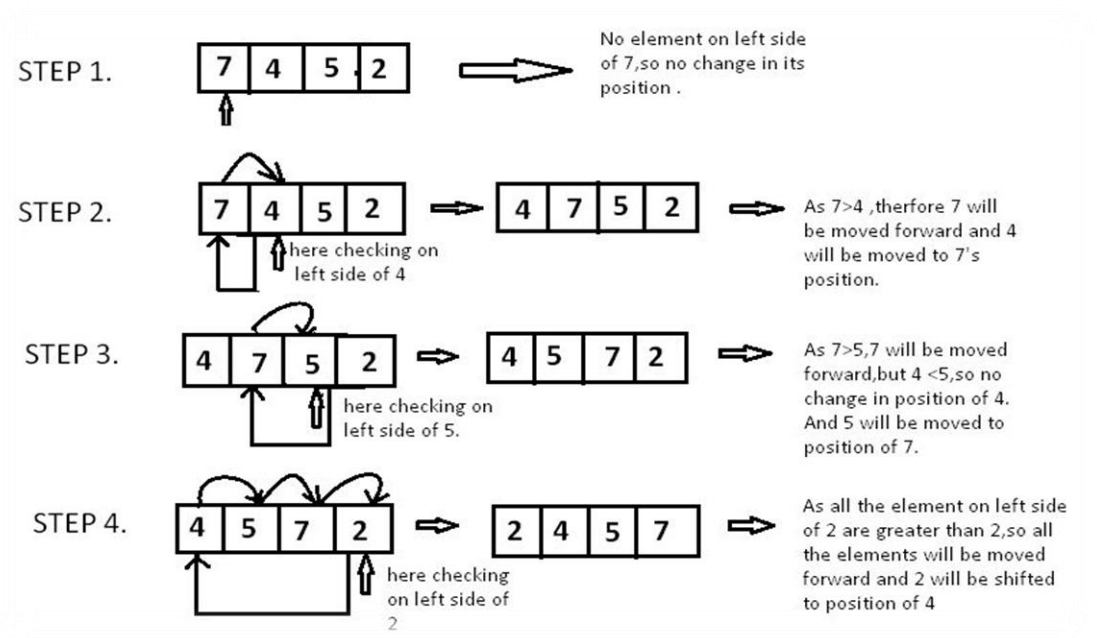
In sorting problems, the base statement is the comparison between items of the array.

The time complexity is a theoretical measure, usually it needs complex theories to be computed, therefore, in some cases, it is estimated using the run-time complexity. This means the algorithm is implemented in any programming language and the run-time is computed on a particular machine.

Report of the sorting algorithms

Insertion sort algorithm:

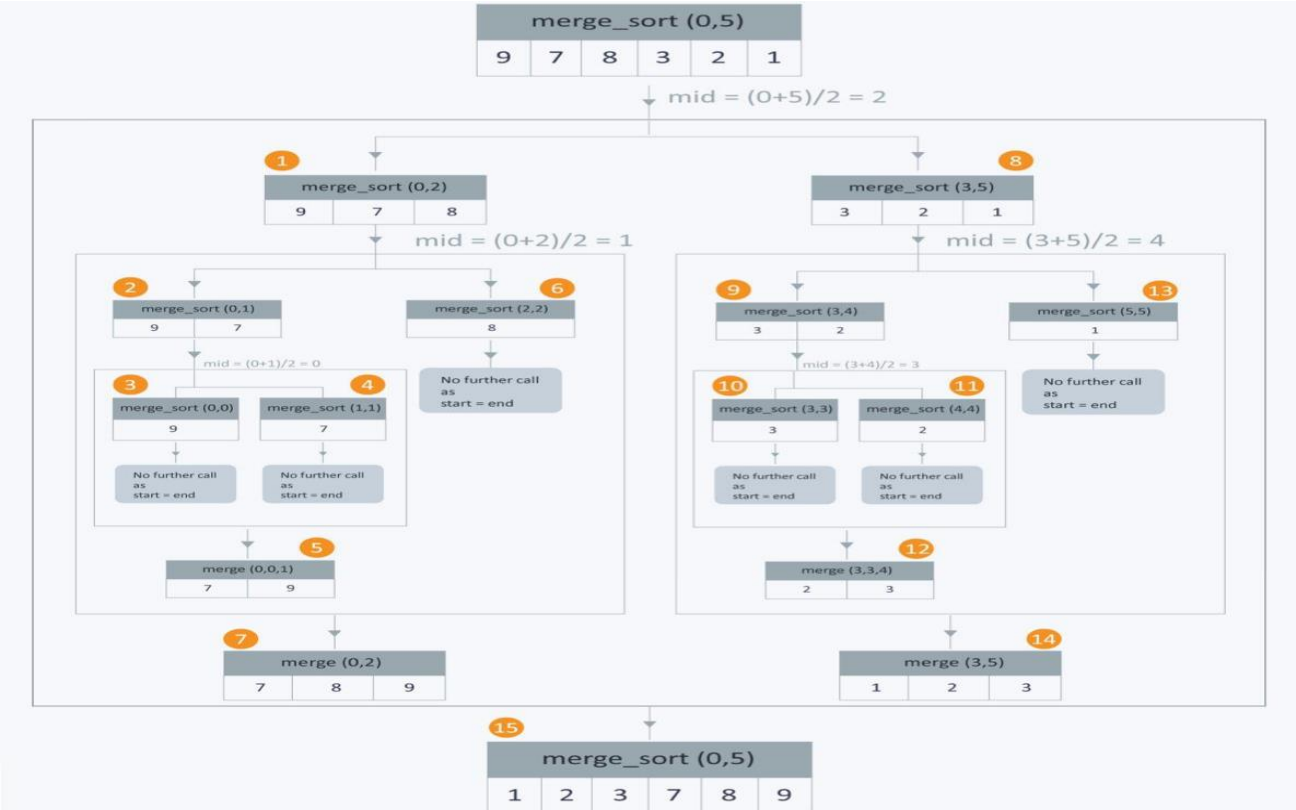
At this point there is no information about the inventor of this algorithm. Like Quicksort, Direct insertion sort is a comparison sort algorithm. It is a O(n2) algorithm. The algorithm is most inefficient when it has to sort an array that is sorted descending. In this case, like the previous sort methods, direct insertion sort algorithm has a maximum polynomial time O(n2). The method is to move one element at a time into the correct position by sliding it to the right. The first part of the array, until the first unsorted element, is the sorted part. This algorithm is easy to understand, as well.



Size	Insertion time
20000	440
40000	1530
60000	3944
80000	6188
100000	9617

merge sort algorithm:

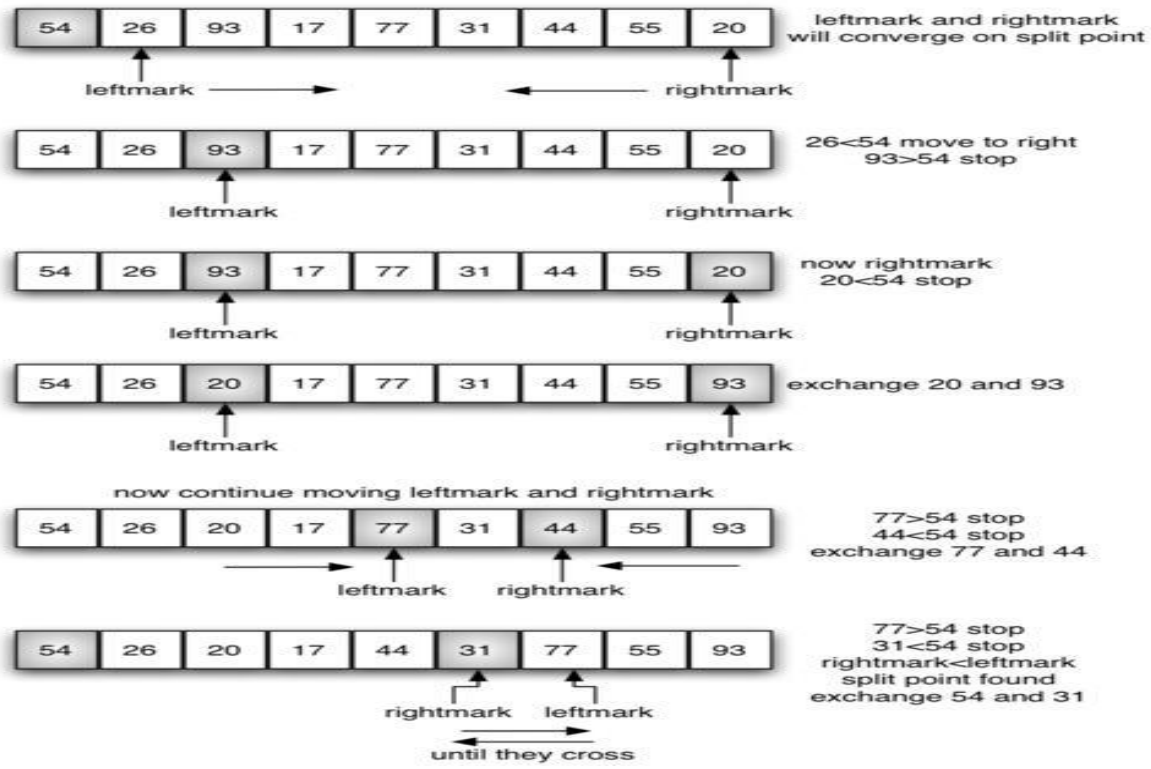
It is an efficient, general-purpose, comparison-based sorting algorithm. Most implementations produce a stable sort, which means that the implementation preserves the input order of equal elements in the sorted output. Merge sort is a divide and conquer algorithm that was invented by John von Neumann in 1945. A detailed description and analysis of bottom-up merge sort appeared in a report by Goldstine and von Neumann as early as 1948.



Size	Merge time
20000	6
40000	10
60000	17
80000	20
100000	25

Quick sort algorithm:

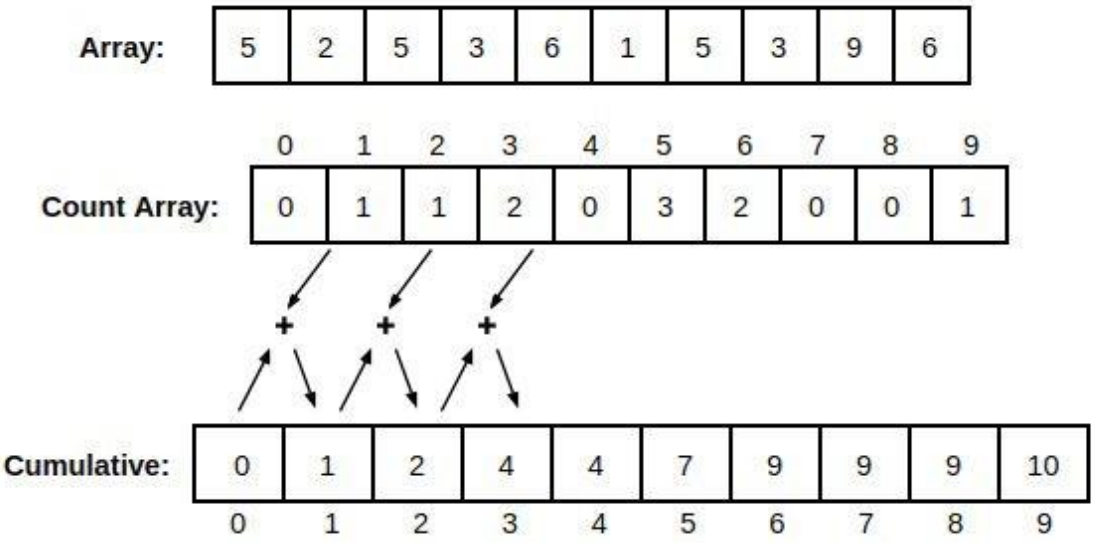
Quicksort is one of the most efficient sorting algorithms; it has the theoretical mean time complexity of O(n*log n). Charles Antony Richard Hoare (born January 11, 1934) developed the algorithm in 1959. The algorithm was published in 1962. In 1975 Robert Sedgewick wrote his Ph.D. thesis about this sorting algorithm. When implemented properly, Quicksort can be even three times faster in comparison with other sorting algorithm because of its outstanding performance. Precisely because of these performances, it is the most widely used algorithm. Nevertheless, like any type of sorting algorithm, sometimes it is not so efficient. The worst case scenario happens when the array is ordered descending, when it will be O (n2). It should be noted that this behavior is rarely met. The algorithm is based on a Divide et Impera method, it successively divides the array in two parts: the items in the left being less than a threshold, whilst the items in the right being greater than it.



Size	Quick time
20000	5
40000	8
60000	15
80000	18
100000	24

Counting sort algorithm:

It is a sorting technique based on keys between a specific range. It works by counting the number of objects having distinct key values (kind of hashing). Then doing some arithmetic to calculate the position of each object in the output sequence.



size	Counting time
20000	0
40000	1
60000	2
80000	2
100000	2

Sample of output

Size of 20000

time1 insertion Sort : 440

time1 merge Sort: 6

time1 counting Sort: 0

time1 quick Sort: 5

****

Size of 40000

time1 insertion Sort : 1530

time1 merge Sort: 10

time1 counting Sort: 1

time1 quick Sort: 8

****

Size of 60000

time1 insertion Sort : 3944

time1 merge Sort: 17

time1 counting Sort: 2

time1 quick Sort: 15

****

Size of 80000

time1 insertion Sort : 6188

time1 merge Sort: 20

time1 counting Sort: 2

time1 quick Sort: 18

****

Size of 100000

time1 insertion Sort : 9617

time1 merge Sort: 25

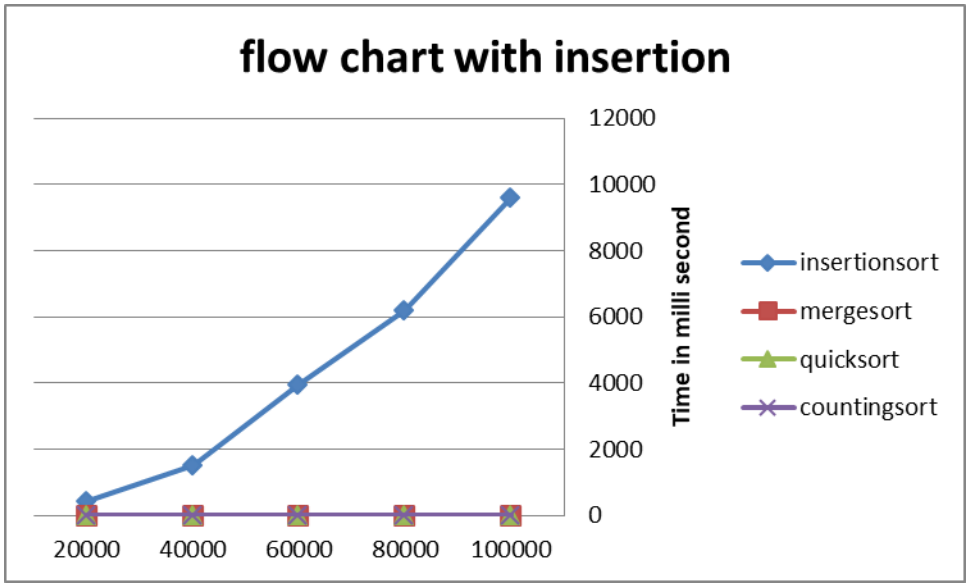
time1 counting Sort: 2

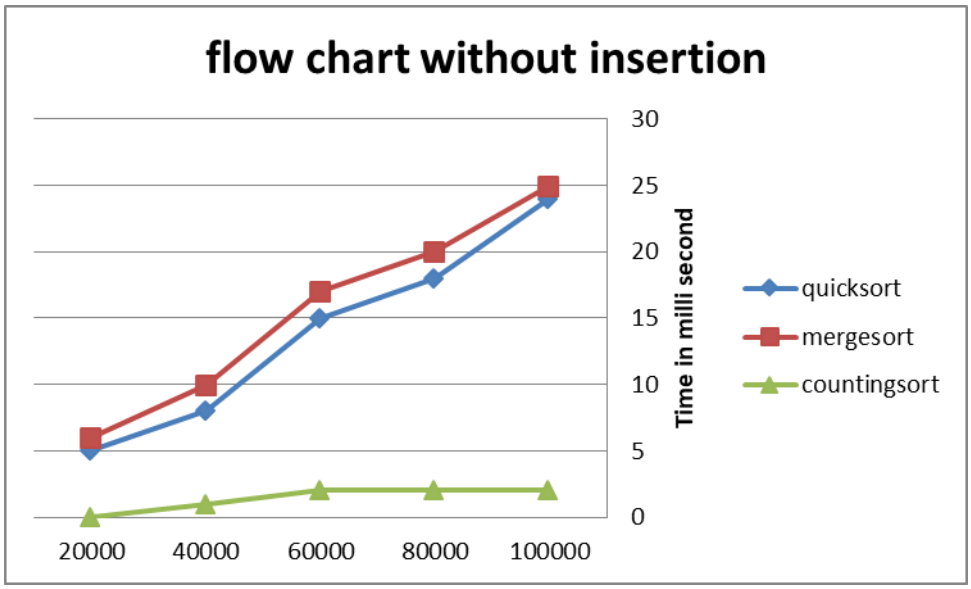
time1 quick Sort: 24

Excel sheet:

	20000	40000	60000	80000	100000
Insertion sort	440	1530	3944	6188	9617
Quick sort	5	8	15	18	24
Merge sort	6	10	17	20	25
Counting sort	0	1	2	2	2

Graph:


flow chart with insertion
12000
10000
8000
6000
4000
2000
0
Time in milli second
20000
40000
60000
80000
100000
insertionsort
mergesort
quicksort
countingsort


flow chart without insertion
30
25
20
15
10
5
0
Time in milli second
20000
40000
60000
80000
100000
quicksort
mergesort
countingsort

- **In theoretical analysis, using - Line by Line - analysis , the time complexity is as shown below :**

	Insertion sort	Merge sort	Quick sort	Counting sort
Worst case	**O(n^2)**	**O(nlgn)**	**O(n^2)**	**O(n)**
Average case	**O(n^2)**	**O(nlgn)**	**O(nlgn)**	**O(n)**
Best case	**O(n)**	**O(nlgn)**	**O(nlgn)**	**O(n)**

In the practical analysis, the sample output in page 7 these value in average case

Conclusion:

After studying the result we noticed that each type of sorting method needs a different time, so the insertions sort was the slowest so obtained the largest time , while the count sort was noticeably faster due to the number of loops and many things in code so the time is very small , the difference between the insertion sort and the rest of the methods are pretty straightforward, but the merge sort and quick sort the time is almost the same .

- **Insertion sort The larger the size →the more time it takes(quadratic increase).**
- **Merge sort the larger the size →rise of time small.**
- **Quick Algorithm same the merge but the merge take more time (the difference in the time constant) But this does not agree with the theoretical results that show that merge faster than quick The reason is due to several factors It might be one of them :**

1. **the often-quoted runtime of sorting algorithms – also play an important role on current hardware. Quicksort in particular requires little additional space and exhibits good cache locality, and this makes it faster than merge sort in many cases**
2. **In addition, it's very easy to avoid quicksort's worst-case run time of O(n2) almost entirely by using an appropriate choice of the pivot?– such as picking it at random (this is an excellent strategy).**
3. **many modern implementations of quicksort (in particular libstdc++'s std::sort) are actually introsort, whose theoretical worst-case is O(nlogn), same as merge sort. It achieves this by limiting the recursion depth, and switching to a different algorithm (heapsort) once it exceeds logn.**

    - **Counting sort when increasing size→A very small increase in time or almost constant .**

Reference:

https://www.geeksforgeeks.org

http://www.cplusplus.com/

https://www.programiz.com/