Optimal Solutions to the Sort Algorithms of Database Structure

Rifat Osmanaj

Dr.c , Lecturer, AAB University, Prishtina

Hysen Binjaku

Lecturer, Prof.Asoc. Dr. Europian Universityof Tirana, Tirana

Abstract

Sorting is much used in massive data applications, insurance systems, education, health, business, etc. To the sorting operation that sorts the data as desired, quick access to the required data is achieved. Typically sorted data are organized in strings as file elements or tables. The most common case is when the tabular data is processed in the main memory of the computer. The paper presents the algorithms currently used for sorting objects that are involved in static and dynamic data structures. Then the selection of the data set on which particular algorithms will be applied will be made and the advantages and disadvantages of each of the algorithms in question will be seen.Thereafter, it is determined the efficiency of the sorting algorithm work and it is considered what is determinative when selecting the appropriate algorithm for sorting.

Keywords: optimal, solutions, algorithms, database, structure

1. Introduction

Solving problems in life, however simple it may be, requires different actions. The set of all sorts of actions with a certain queue, in solving a problem, is called Algorithm. It is a well-defined calculator procedure that takes some values or values sets as inputs and outputs value or value sets as outputs.

Thus, an algorithm represents a sequence or sequence of computing steps that convert the entry to exit results. Sorting represents a fundamental operation in computer science (many programs use it as an intermediate step) and therefore a large number of sorting algorithms have been developed. The algorithm is correct if, for each incoming instance, the output is correct or correct.

The sequential list (vector, matrix, or multidimensional field) represents static data structures because their size does not change during the execution of the program.While linked lists represent dynamic data structures, because the nodes are set or deleted dynamically during the execution of the program.The number of nodes is likely to increase while there is free memory on the computer, but can even be reduced.

Algorithms to solve the same problem often change dramatically in their efficiency. Differences in algorithms can be far greater and more important than differences due to software and hardware. Two kinds of algorithms for sorting, one of the weakest bubble sort varieties, and the other with very high speed (quick sort) efficiency are used in the paper.

2. BUBBLE SORT

The bubble sort makes the alignment of the string elements in such a way that at first glance impresses for a very fast method. In fact the bubble sort is one of the simplest methods used in computer science for data sorting. The algorithm takes its name based on the movement of the smallest element of the "bubble" list to the heading of the list, similar to the movement of air bubbles in the water.

The bubble sort is an algorithm that works by repeating the steps in the list to be sorted by comparing each pair of the list and changing the locations of the elements that are not properly sorted.

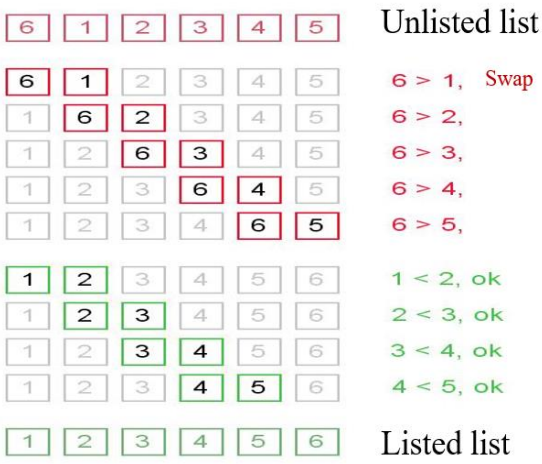
The bubble sort can, however, be used efficiently to rank the lists on which most of the elements are in the right place (the lists are almost sorted), unless the number of elements is too small.

For example, if only one element is not listed the bubble variety will take 2n time, if two elements are not sorted the bubble variety will take time 3n.

In the case of bubble sort avarage case (average) and worst case (the worst case) are: O (n2).

The bubble sort uses only element comparisons and is therefore referred to as a comparator. Also the bubble variety is stable and adaptive.

Example of bubble sort:



Listing Code:

```
//  Bubble Sort
#include <iostream>
using namespace std;
int compare(int, int);
void sort(int
, const int);
void swap(int *, int *);
int compare(int x, int y)
{
    return(x > y);
}
void swap(int *x, int *y)
{
```

```
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
void sort(int table
, const int n)
{
    for(int i = 0; i < n; i++)
    {
  for(int j = 0; j < n-1; j++)
  {
 if(compare(table[j], table[j+1]))
     swap(&table[j], &table[j+1]);
  }
    }
}
int quantity;
int* tab;
int main()
{
cout << "\nNumber of elements: ";
cin >> quantity;
tab = new int [quantity];
cout << "\nLists number: \n\n";
for (int i = 0; i < quantity; i++)
{
   int x = i;
                    cout << "Numri " << ++x << ": ";
   cin >> tab[i];
}
cout << "\n\n List before sort: ";
for (int i = 0; i < quantity; i++)
```

```
{
    cout << tab[i] << " ";
}
cout << "\n\n List after sort: ";
sort(tab, quantity);
for(int i = 0; i < quantity; i++)
{
    cout << tab[i] << " ";
}
cout<<endl;
cout<<endl;
return 0;
}
```

2.1.1. Measuring the execution time

The execution time is one of the main tools that determines the functioning of the algorithm. The execution time of the Bubble Sort algorithm depends on the number of elements in the list: the order of 27777 elements lasts more than the order of 17777 elements, the execution time in relation to the number of elements of the list is a quadratic function.

The execution time measurements for the Bubble Sort algorithm are performed on some computers, but in this case, the results of 3 computers that have different results compared to others will be reviewed.

Computer 1:

Intel(R) Celeron(R) M CPU
530 @ 1.73GHz
1.73 GHz, 768 MB of RAM

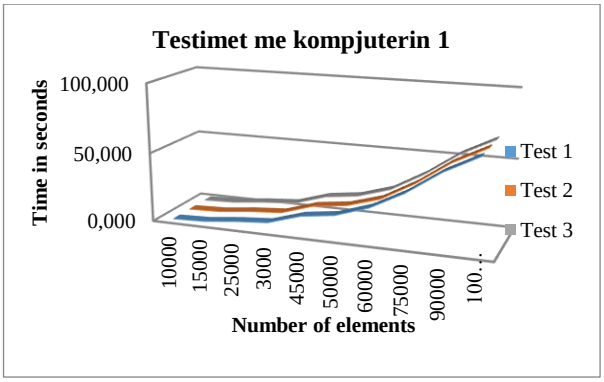
After three tests we have the following results:

Number of elements	Computer 1 Time in seconds		
	Test 1	Test 2	Test 3
10000	0.656	0.703	0.703
15000	1.468	1.484	1.516
25000	4.110	4.157	4.094
30000	5.969	5.953	5.938
45000	13.360	13.437	13.344
50000	16.593	16.469	16.437
60000	24.532	23.656	23.797
75000	37.219	37.125	37.375

90000	53.703	53.828	53.906
100000	66.234	66.125	66.156

Computer 1 tests

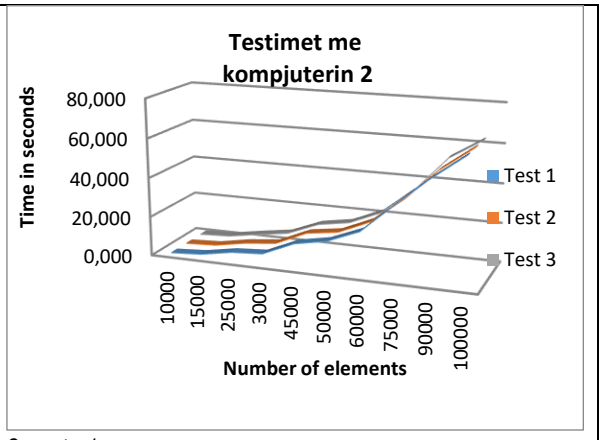
For computer features 1 of the empirical results it is seen that there are small differences in the tests 1,2,3



Computer 2:

Intel(R)
Pentium(R) 4 CPU 3.00GHz
3.00 GHz, 504 MB of RAM

After three tests we have the following results:



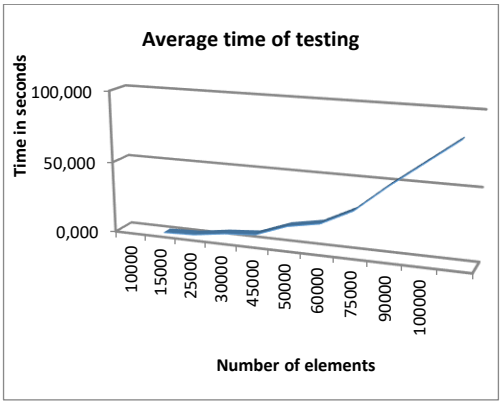
Number of elements	Computer 1 Time in seconds		
	Test 1	Test 2	Test 3

10000	0.625	0.625	0.688
15000	1.656	1.406	1.422
25000	4.531	4.297	4.078
30000	5.610	5.985	6.484
45000	12.609	13.093	12.895
50000	15.687	15.391	15.578
60000	22.219	22.391	22.437
75000	36.703	35.343	35.547
90000	50.703	51.109	52.969
100000	62.953	63.063	63.015

For computer features 1 of the empirical results it is seen that there are small differences in the tests 1,2,3

Finally, the average of the tests on these computers is calculated, which is the average of all tests:

Number of elements	Average time of testing
10000	0.778
15000	1.653
25000	4.884
30000	6.783
45000	15.074
50000	19.285
60000	30.445
75000	49.068
90000	65.995
100000	82.925



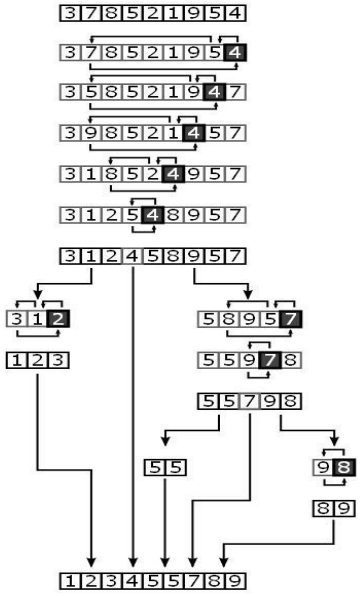
3. QUICK SORT

The fast order has the complexity O (n log n) in the average case, while in the worst case the complexity of this algorithm is O (n2).

In practice, quick sorting is significantly faster than other O complex (n log n) algorithms, because the algorithm's crossovers can be applied efficiently in larger architectures, and in different data in practice , to make the design of elections that minimize the probability of the need for quadratic time. The quick sort has enabled maximum utilization of the memory hierarchy, creating great advantages in using virtual memory.

The Quick Sort works according to the divide and conquer method, dividing the list into the sublists. Lists with only one element or with 0 elements do not need to be sorted.The quick sort is part of the group of comparative algorithms, which is efficient but unstable.

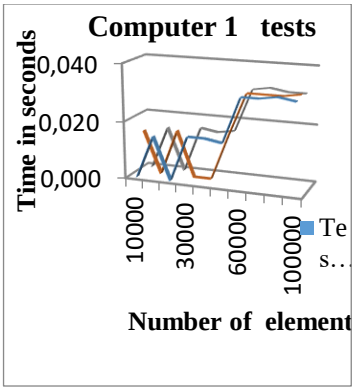
The three most popular varieties of quick sort are: Balanced quicksort, External quicksort, and Multikey quicksort.



Example:

3.1. Execution time

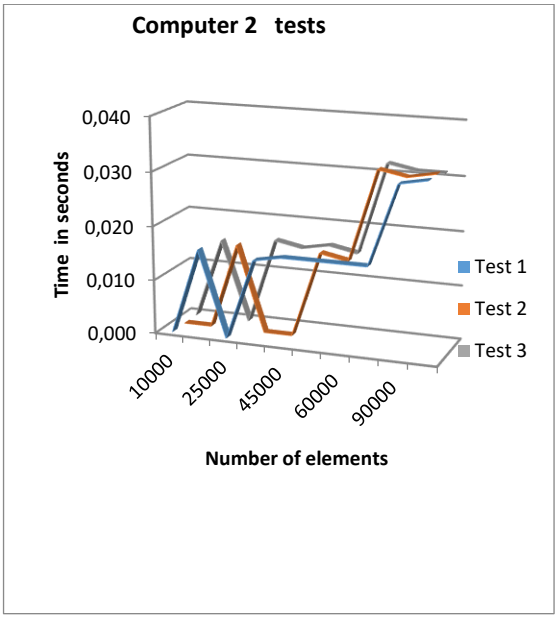
Quick Sort is a fast algorithm, faster compared to the previous methods, but is also quite complex and highly recursive. Next we will do the execution time measurements for vectors filled with random number numbers. The execution time measurements for the Quick Sort algorithm are performed on two computers.



Computer 1: Intel(R) Celeron (R) M CPU 530 @ 1.73 GHz, 768 MB of RAM After three tests we have the following results:

Number of elements	Computer 1 Time in seconds		
	Test 1	Test 2	Test 3
10000	0.000	0.015	0.000
15000	0.015	0.000	0.015
25000	0.000	0.016	0.000
30000	0.016	0.000	0.016
45000	0.016	0.000	0.015
50000	0.015	0.016	0.016
60000	0.031	0.031	0.031
75000	0.031	0.031	0.032
90000	0.032	0.031	0.031
100000	0.031	0.032	0.031

For computer features 1 of the empirical results it is seen that there are small differences in the tests 1,2,3



Computer 2: Intel (R) Pentium (R) 4 CPU 3.00 GHz, 3.00 GHz, 504 MB of RAM After three tests we have the following results:

Number of elements	Computer 2 Time in seconds		
	Test 1	Test 2	Test 3
10000	0.000	0.000	0.000
15000	0.016	0.000	0.015
25000	0.000	0.016	0.000
30000	0.015	0.000	0.016
45000	0.016	0.000	0.015
50000	0.016	0.016	0.016
60000	0.016	0.015	0.015
75000	0.016	0.032	0.032
90000	0.031	0.031	0.031
100000	0.032	0.032	0.031

For computer features 2 of the empirical results it is seen that there are small differences in the tests 1,2,3

Comparing the results achieved on these computers (above), the average of tests for each computer is derived, and then compared to each other.

From the tables it is seen that for the same number of elements there are differences in execution time for different computers due to the characteristics of the computers.

Finally, the average of the tests on these computers is calculated, which is the average of all tests:

Below is the average of the tests on these computers for one (**bubble sort**) and the other (**quick sort**):

Number of elements	Average time of testing
10000	0.778
15000	1.653
25000	4.884
30000	6.783
45000	15.074
50000	19.285
60000	30.445
75000	49.068
90000	65.995
100000	82.925
Number elements	Average time of testing
10000	0.005
15000	0.009
25000	0.009
30000	0.012
45000	0.014
50000	0.019
60000	0.026
75000	0.033
90000	0.033
100000	0.035

Bubble Sort **Quick Sort**

From the above tables it is seen that for the same number of elements there are major differences at the execution time for different algorithms for sorting.

As seen, the quick sorting algorithm is much more advanced than the bubble algorithm.

4. CONCLUSION

Informally, an algorithm 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. An algorithm is thus a sequence of computational steps that transform the input into the output. In general we used more techniques of algorithm design and analysis so that you can develop