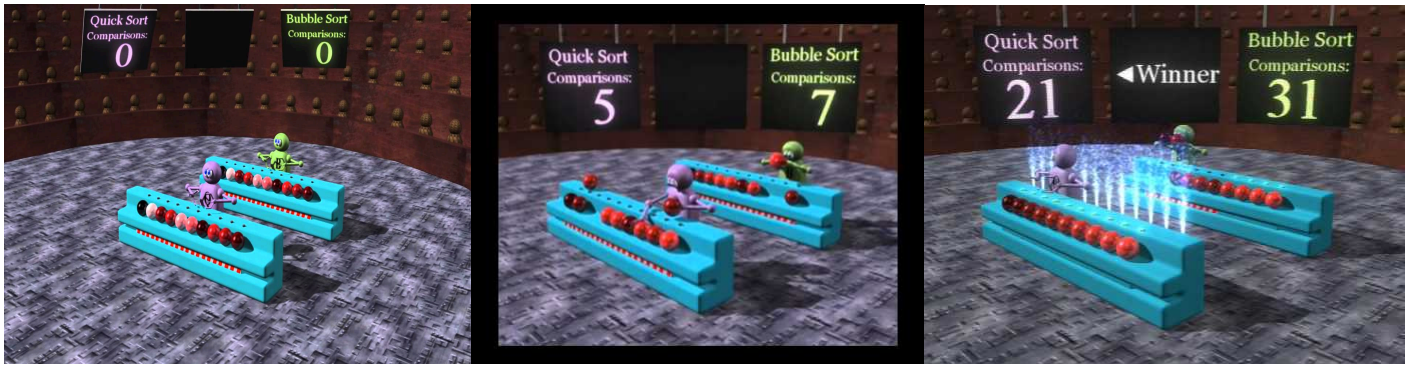
# *Assignment 2 Sorting Algorithms Complexity O() – Quicksort*

Ion Alin Spiridon
14305180




## ***Abstract***

The objective of this exercise is to show the increased efficiency of the Quick sort algorithm over other types of sorting algorithms like Bubble sort on the present case. An array is generated to test the complexity of the two algorithms either with regards to the size of the sequence of numbers or with respect to the type of the sequence: randomly distributed, mostly sorted and sorted but in reversed order.
The proven complexity for the Quick sort algorithm was O(n log n) on average and worst case and O(n) in the best case, while for the Bubble sort algorithm was O(n^2) for worst, average and worst case.

## ***Introduction***

As the technology improved more and more applications started to relay on computers storing and managing data. The data increased in size as fast as the storage capacity increased on modern computers. This created the need for algorithms capable of dealing and manipulating data efficiently. The efficiency of an algorithm can be described as, for example, the amount of processor time an algorithms needs or how many instruction an algorithm has. The complexity in question for this exercise is defined as the type of growth of the function or O~ notation. This evaluates how computing cost grows as size n of data structure increases.

### ***Method***

The first thing to do was to create a C++ program, which was supposed to generate an array of numbers and use a bubble sort algorithm and a quicksort algorithm. The program was also to generate a probe count for the two algorithms in order to be able to estimate their complexity respectively. The results were to be displayed on a table and compared.

***Solution***

The proposed solution is a C++ program, which presents the user with two ways of testing the complexity of the algorithms with respect to the size of the data structure or with respect to the type of the sequence: randomly distributed, mostly sorted and sorted but in reversed order.

For the first type of testing the user is prompted to enter the data manually (or use copy and paste from a data file), then the program runs with no further user input, outputting the sorted array using the bubble sort algorithm with its respective probe count and the array sorted with the quicksort algorithm with its probe count. For the purposes of this exercise the sequence of numbers used was:
64 61 169 113 81 61 206 176 39 100 22 200 128 152 59 165 67 116 165 72 26 149 58 204 188 69 203 94 96 134 83 122 192 85 62 159 35 162 95 92 126 66 66 203 187 18 132 182 181 175 9999

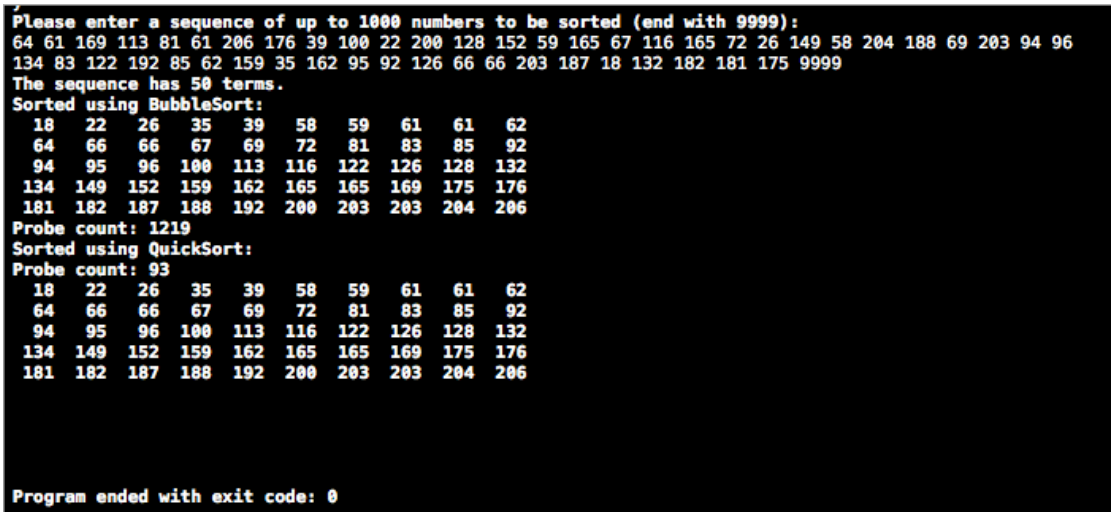



Figure 1 – Testing based on size of data

For the second type of testing the user is presented with the choice of sequence type as randomly distributed, mostly sorted and sorted but in reversed order. Once a choice is made the program uses a type specific hard coded 10 elements array and sorts it using bubble sort and quick sort algorithm, after which displays the results. Figure 2 is an illustration this type of sorting being executed.

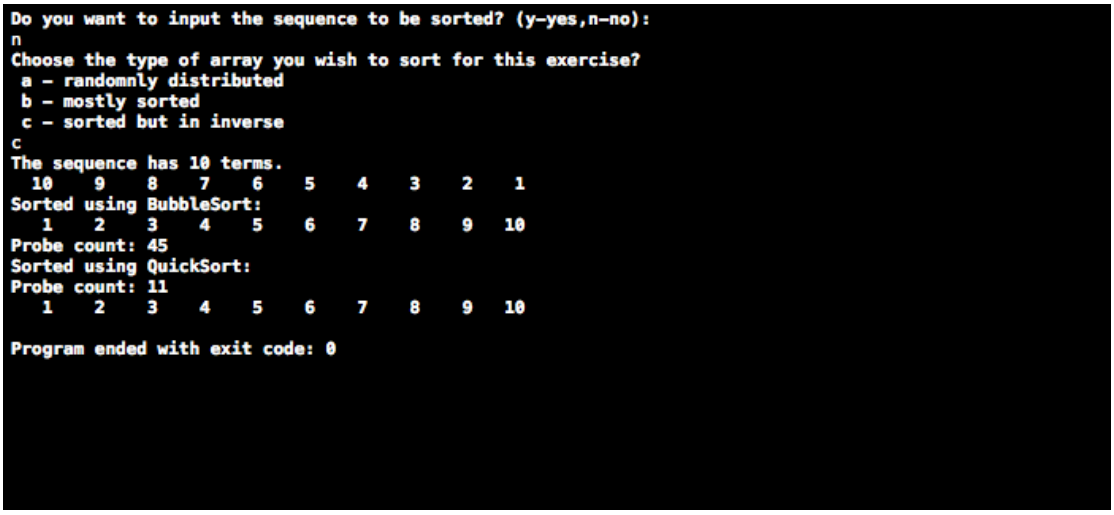



Figure 2 – Testing based on type of sequence

***The bubble algorithm*** used, works by comparing two elements at the time, starting with the first two elements on the array and swapping them if necessary until it reaches the end of the array. The Boolean variable "swapped" is used to check if a swap was made and if that is the case the previously described operation is repeated until no more swaps have been necessary. The integer "j" is used to store the number of passes the algorithm has done already and based on it the last "j-th" elements are not visited again since they are already on the correct order. As a special characteristic of the bubble sort it is worth mentioning that it stops after one pass alone if the given array is already sorted. The expected O~ complexity is O(n^2) for worst and average cases and O(n) on the particular best scenario when the only element out of order in the data sequence is the first element. [1] This algorithm is both stable and adaptive. Stability means that equivalent elements retain their relative positions, after sorting. [2] An **adaptive algorithm** is an **algorithm** that changes its behaviour based on information available at the time it is run. [3] Chart 1 is a graphical representation of the bubble algorithm.

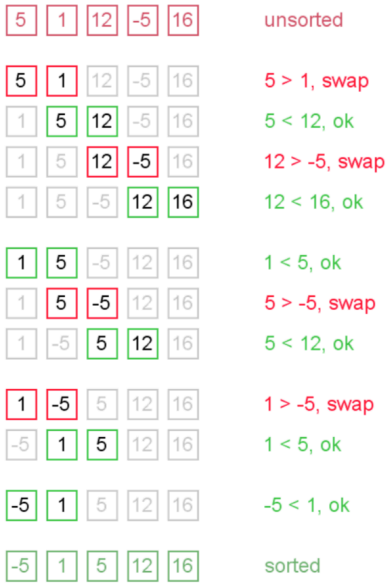



Chart 1 –bubble sort

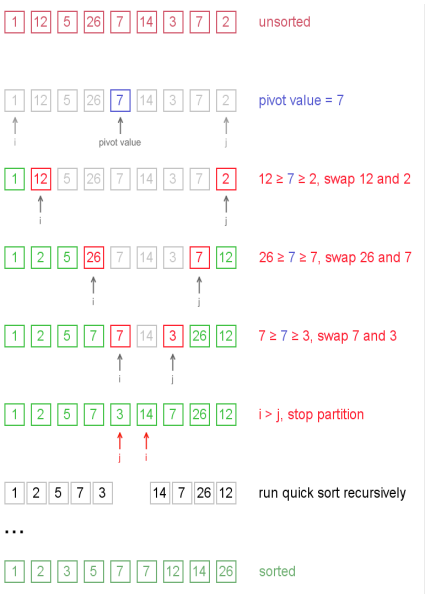



Chart 2 – quick sort

***The quicksort algorithm*** used works by employing a divide and conquers strategy where a pivot element is used to sort all the elements with value bigger than the pivot on its left side on the array, while the elements bigger in value than the pivot are stored on the right side. This was done starting with the left most ("i" indices) and right most element ("j" indices) of the array and employing the middle element as the pivot. Once the array was sorted as described above (algorithm stops when **i** becomes greater than **j**), the algorithm called itself recursively until the entire array was sorted. The expected O~ complexity is O(n^2) for the worst case and O(n log n) for the average case. Best case, mostly sorted data sequence, it performs with O(n) complexity. Although this algorithm is neither stable nor adaptive it outperforms other O(n log n) sorting algorithms for most of the practical data. [4] Chart 2 is a graphical representation of the bubble sort.

## ***Results table***

Next the table containing the results of the exercise is presented:

| ***Complexity based on the type of sequence in the array*** | | | | | | |
|---|---|---|---|---|---|---|
| Algorithm name | Bubble sort | | | Bubble sort | | |
| | Best case - almost sorted | Average case - random elements | Worst case - sorted in | Best case - almost sorted | Average case - random | Worst case - sorted in reverse |
| Big O complexity Theoretical | O(n) | O(n^2) | O(n^2) | O(n) | O(n log n) | O(n^2) |
| Probe count for an array of 10 elements | 9 | 35 | 45 | 7 | 11 | 11 |
| ***Complexity based on the size of the array*** | | | | | | |
| n random elements arrray | 10 random elements | 50 random elements | 100 random elements | 10 random elements | 50 random elements | 100 random elements |
| Probe count | 35 | 1290 | 4940 | 11 | 93 | 212 |

## ***Conclusion***

Both algorithms performed as expected. The bubble sort algorithm has ben outperformed by the quick sort algorithm on every type of test they were subjected to. This exercise has proven that the most efficient sorting algorithm is the quick sort algorithm by a considerable margin and as a result it is recommendable that for general or practical data usage it should be used in order to improve the efficiency.

## ***References***


[1] – 18/11/2016 http://www.algolist.net/Algorithms/Sorting/Bubble_sort
[2] - 18/11/2016 http://homepages.math.uic.edu/~leon/cs-mcs401-s08/handouts/stability.pdf
[3] - 18/11/2016 http://www.mit.edu/~ibaran/mthesis.pdf
[4] - 18/11/2016 http://www.algolist.net/Algorithms/Sorting/Quicksort