A Comparative Study between Various Sorting Algorithms

Article · March 2015

CITATIONS
23

READS
15,989

1 author:

Jehad Hammad
Al-Quds Open University
26 PUBLICATIONS **192** CITATIONS

SEE PROFILE

A Comparative Study between Various Sorting Algorithms

Jehad Hammad †

† *Faculty of Technology and Applied Sciences, Al-Quds Open University, Palestine*

Summary
The objectives of this paper are to provide a solid foundation for the sorting algorithms and to discuss three of them (bubble sort, selection sort and gnome sort). In this work, the reader will be able to know the definition of algorithms and recognize the major design strategies of algorithms, including Divide and Conquer, Dynamic Programming, The Greedy-Method and Backtracking and Branch-and-Bound. The aim of this comparison between the three algorithms is to know the running time of each one according to the number of input elements. C-Sharp language is used in the implementation. The result of this implementation shows no specific algorithm can solve any problem in absolute.

Key words:
Bubble sort, selection sort, gnome sort, Divide-and-Conquer, Dynamic Programming, The Greedy-Method and Backtracking

1. Introduction

"An algorithm is a complete, step-by-step procedure for solving specific problem. Each step must be unambiguously expressed in terms of a finite number of rules and guaranteed to terminate in a finite number of applications of the rules. Typically, a rule calls for the execution of one or more operations." [2]. The use of algorithms did not begin with the introduction of computers, people using them while they are solving problems. We can describe algorithms as a finite sequence of rules which describes and analyses the algorithms. The major design strategies for the algorithms will be described in section three. Sorting has attracted the attention of many researchers because information is grown rapidly and this requires a stable sorting algorithms in taking into account the vectors of performance, stability and memory space. Each sorting algorithm has its own technique in executing; besides, every problem can be solved with more than one sorting algorithms. In this study, we will compare between the sorting algorithms based on best-case B(n), average-case A(n), and worst-case efficiency W(n) [6] that refer to the performance of the number n of elements. The Big O notation is used to classify algorithms by how they respond to changes in input size. [7, 9]

Manuscript received March 5, 2015
Manuscript revised March 20, 2015

2. Sorting

Sorting is the process of arranging data in specific order which benefits searching and locating the information in an easy and efficiency way. Sorting algorithms are developed to arrange data in various ways; for instance, an array of integers may be sorted from lower to highest or from highest to lower or array of string elements may sorted in alphabetical order. Most of the sorting algorithms, like selection sort, bubble sort and gnome sort algorithms [6], use swapping elements technique until reaching the goal. Each sorting algorithm is chosen based on best-case, average-case and worst-case efficiency. Table 1 shows the order of the best-case, average-case and worst case for some of sorting algorithms.

Table 1: Summary of the best-case, average-case and worst-case[2]

Algorithm	*B(n)*	*A(n)*	*W(n)*
HornerEval	n	n	n
Towers	2^n	2^n	2^n
LinearSearch	1	n	n
BinarySearch	1	logn	logn
Max, Min , MaxMin	n	n	n
InsertionSort	n	n^2	n^2
MergeSort	nlogn	nlogn	nlogn
HeapSort	nlogn	nlogn	nlogn
QuickSort	nlogn	nlogn	n^2
BubbleSort	n	n^2	n^2
SelectionSort	n^2	n^2	n^2
GnomeSort	n	n^2	n^2

3. Major Design Strategies for the algorithms

The main aspect of design algorithms is to produce a solution which gives sufficient run time. Bellow, there is a brief overview of each strategy

3.1 Divide-and-Conquer

First, the sequence to be sorted A is partitioned into two parts, such that all elements of the first part B are less than or equal to all elements of the second part C (divide). Then the two parts are sorted separately by recursive application of the same procedure (conquer). Recombination of the two parts yields the sorted sequence (combine). [2] Figure 1 illustrates this approach.

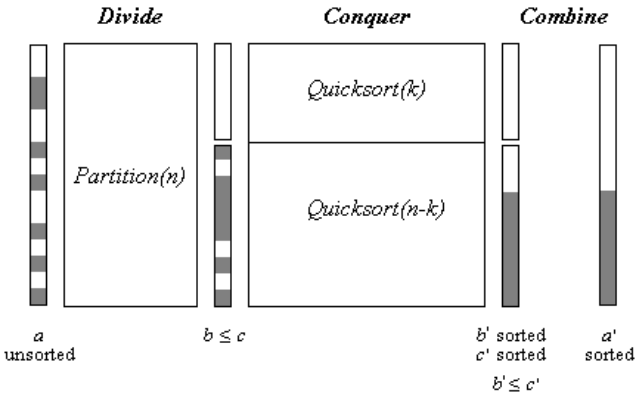

Fig. 1 Divide-and-Conquer Strategy.

3.2 Dynamic Programming

Like divide-and-conquer, dynamic programming technique builds a solution to a problem from solution to sub-problems, and all sub-problems must be solved before proceeding to next sub-problems [2]. For instance, Dynamic programming version of Fibonacci.

Fibonacci(n)
If n is 0 or 1, return 1
Else solve Fibonacci(n-1) and Fibonacci(n-2)
Look up value if previously computed
Else recursively compute
Find their sum and store
Return result

The O(n) for solving Fibonacci is (n-2).

3.3 The Greedy Method

It is a hoping algorithm that solves the problem by choosing an optimal choice for the solution that leads to the goal but that will coast much time. [2] Figure 2 shows a case of greedy method to reach the goal G

3.4 Backtracking and Branch-and-Bound

Backtracking is using depth-first search in the space tree while Branch-and-Bound is using the breadth-first search. Both depend on how to take a decision to reach the solution; for example. in traveling salesman problem(TSP), an algorithm must keep tracking with shortest path and cut-off any tour who's distance exceed the current short one [2].

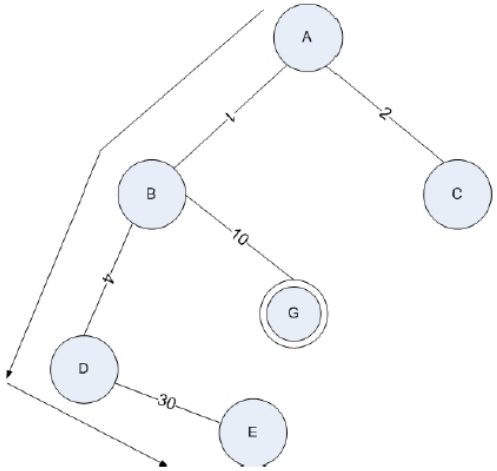

Fig. 2 Greedy Method case.

4. Implemented Sorting Algorithms

The paper in this section will discuss the technique of three sorting algorithms (bubble sort, selection sort and gnome sort).

4.1 Selection sort algorithm

Selection sort is the simplest according to sorting techniques. As shown in figure 3 in order to sort an array of size n, selection find the minimum value, swap it into the first position, then repeat searching and swapping on the remaining (n-1) elements. [1]

We concluded

$$(n-1)+(n-2)+(n-3)+\ \ldots+1=\frac{(n-1)*n}{2}$$

Big O notation O(n2) for the three cases (Best, Average and worst) is the same since selection sort algorithm continue sorting even if the remaining elements in the array is already sorted, in figure 3 after loop3 no need to do more sorting [4, 3]. Selection algorithm is easy for implementation and useful for sorting small data as it does not occupy a large space of memory, but it is inefficient for large list. [5]

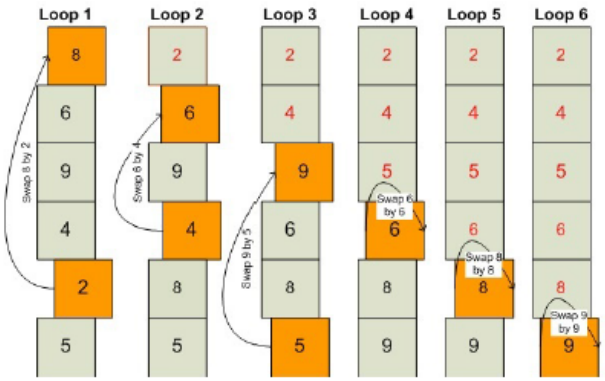

Fig. 3 Greedy Method

4.2 Bubble sort algorithm

Bubble sort starts comparing the last item M in list with the (M-1) and swaps them if needed. The algorithm repeats this technique until the required order as shown in the Pseudo code:

```
For  i = 1:n,
        swapped = false
        for  j = n:i+1,
                if a[j] < a[j-1],
                        swap a[j,j-1]
                        swapped = true
        break if not swapped
end
```

In case of sorted data, bubble sort takes O(n) (best-case) time, since it passes over the items one time. But it takes O(n2) time in the average and worst cases because it requires at least 2 passes through the data.1) Advantage: Simplicity and ease-of-implementation.2) Disadvantage: Code inefficient.[3]

$$O(n2) = n + (n + 1) + (n + 2) + :: + 1$$

4.3 Gnome sort algorithm

Gnome sorting algorithm is similar to bubble sort [4]. The movement of the elements depends on series of swapping and no need for nested loops to reach the final order of the list. The algorithm always finds the first place where tow adjacent elements are in the wrong order and swaps them as shown in Pseudo-code

```
function gnome(a[])
        i := 1
        while i < a[].length
                if (a[i] >= a[i-1])
                i++
        else
                swap a[i] and a[i-1]
                if (i > 1)
                i-
                end if
        end if
        end while
end function
```

If the data is already sorted, gnome sort takes O(n) (best-case) time. This is because it passes over the items one time without swapping though taking O(n2) time in the average and worst cases because of requiring at least 2 passes through the data as bubble sort algorithm. [8]

5. Performance comparison between the implemented sorting Algorithms

In this work, the selection sort, bubble sort and gnome sort algorithms have been implemented by using c-sharp programming language that calculates the executing time at the same input data on the same computer (windows 7 professional 64-bit operating system having Intel Core i5 and 4GB installed memory). Stopwatch timer in c-sharp is used to measure the running time in milliseconds.

The unsorted data-set has the size of 5000,10000,20000 and 30000 also 30000 already sorted data. The experiment is executed in low different technique. The first one is by doing the test four times on every unsorted data for the three sorting algorithms as shown in table 2. The second one is also by doing the test four times on the sorted data for the three algorithms as shown in table 3.

Table 2: Running time for unsorted items.

First Run(Time in Milliseconds)			
N items	***Selection sort***	***Bubble sort***	***Gnome sort***
5000	117	248	119
10000	459	1030	483
20000	1851	3899	1934
30000	3734	9446	4323
Second Run(Time in Milliseconds)			

5000	115	352	119
10000	459	926	461
20000	1851	4236	1858
30000	3696	7516	3116
Third Run(Time in Milliseconds)			
5000	116	251	119
10000	463	1043	465
20000	1579	4099	1810
30000	3110	5594	3391
Fourth Run(Time in Milliseconds)			
5000	116	248	119
10000	457	957	483
20000	1534	3636	1621
30000	3995	8553	4438
Average Run(Time in Milliseconds)			
5000	116	249.75	119
10000	459.5	989	473
20000	1703.75	3967.5	1805.75
30000	3633.75	7777.25	3817

Table 3: Running time for sorted items.

First Run(Time in Milliseconds)			
N Sorted items	***Selection sort***	***Bubble sort***	***Gnome sort***
30000	4979	4054	0
Second Run(Time in Milliseconds)			
30000	2919	2991	0
Third Run(Time in Milliseconds)			
30000	3608	4035	0
Fourth Run(Time in Milliseconds)			
30000	4077	4038	0
Average Run(Time in Milliseconds)			
30000	3670.75	3779.5	0

6. Results and Discussions

We can observe from the table 2 that selection Sort is taking the least time in all cases as shown in figure 4, figure 5, figure 6 and figure 7.

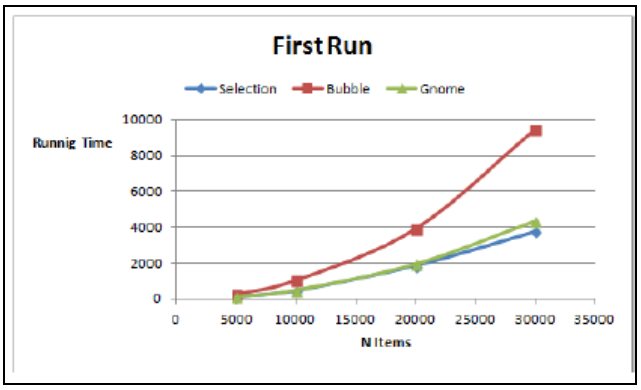

Fig. 4 first run for unsorted items

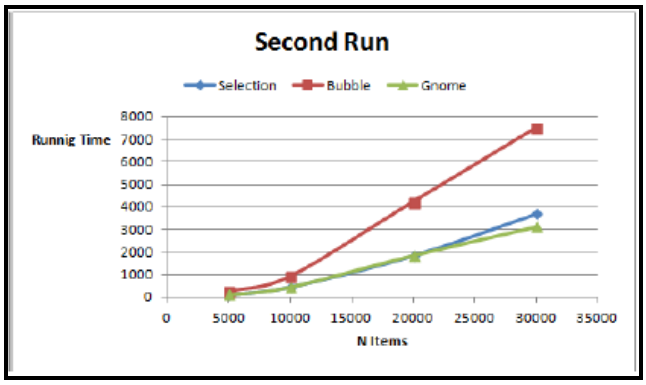

Fig. 5 Second run for unsorted items

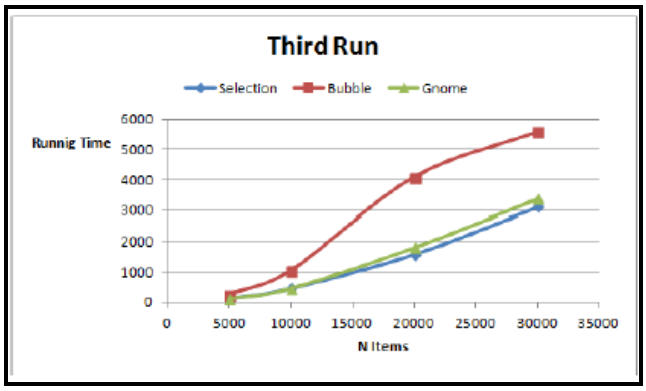

Fig. 6 Third run for unsorted items

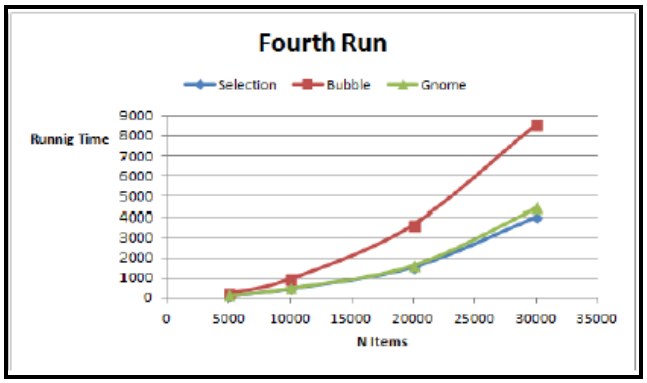

Fig. 7 Fourth run for unsorted items

In figure 8, the average time for selection sort shows that no big different that indicates that O(N2)time for best, average and worst cases are the same. Bubble and gnome, however, show that in best case O(N)time and O(N2) time for average and worst cases.

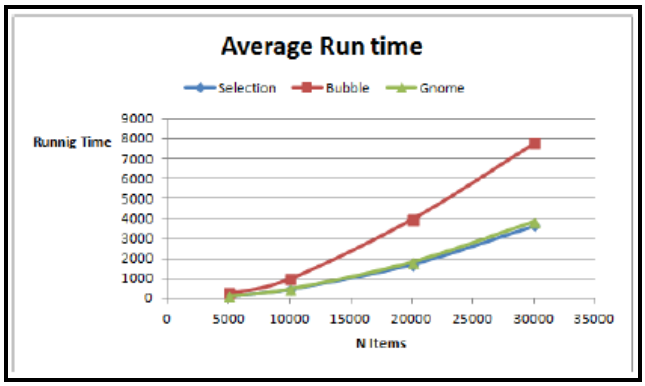

Fig. 8 Average run for unsorted items

Figure 9 shows that the average time for selection sort in terms of sorted or unsorted data is the same which concludes that O(N2) time for best, average and worst cases are the same and bubble sort has the same performance whether the data is sorted or not. Gnome shows good performance in case the data is sorted and the best case is O(1) time.

7. Conclusion

This paper discuss a comparison between three sorting algorithms (selection sort, bubble sort and gnome sort). The analysis of these algorithms are based on the same data and on the same computer. It has been shown that gnome sort algorithm is the quickest one for already sorted data but selection sort is more quick than gnome and bubble in unsorted data. Bubble sort and gnome sort swap the elements if required. In selection sort, however, it continues sorting even if the elements are already sorted. Doing more comparison between more different sorting algorithms is required since no specific algorithm that can solve any problem in absolute.

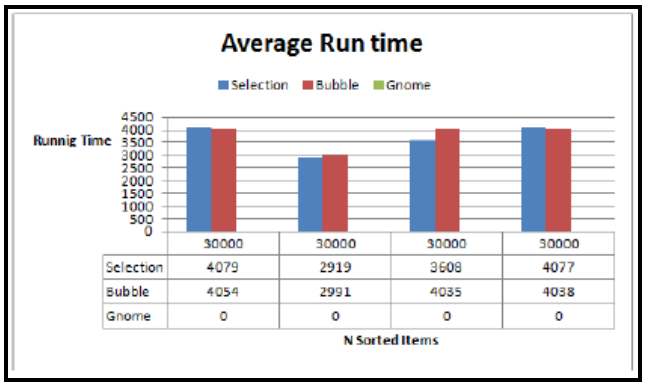

Fig. 9 Average run for sorted items

Acknowledgments

The author would like to express his cordial thanks to Mr. Jumah Zawhera from Al-Quds Open University – Bethlehem Branch (QOU) for his valuable advice as well as to Dr. Yousef Abuzir the Dean of Faculty of Technology and Applied Sciences (FTAS).

References

[1] url:http://www.cs.ucf.edu/courses/cop3502/nihan/spr03/sort.pdf

[2] Kenneth A. Berman and Jerome L. Paul. Algorithms: Sequential, Parallel, and Distributed. Computer Science and Engineering. Thomason course technology, 2002. isbn: 0-534-42057-5.

[3] Md. Khairullah. Enhancing Worst Sorting Algorithms. Computer Science and Engineering. International Journal of Advanced Science and Technology, 2013.

[4] Khalid Suleiman Al-Kharabsheh et al. Review on Sorting Algorithms A Comparative Study. Computer Science. International Journal of Computer Science and Security (IJCSS), 2013.

[5] Ramesh Chand Pandey. Study and Comparison of various sorting algorithms. Computer Science and Engineering. 2008. isbn: 80632019.
[6] Pankaj Sareen. Comparison of Sorting Algorithms (On the Basis of Average Case). Department of Computer Applications. International Journal of Advanced Research in Computer Science and Software Engineering, 2013. isbn: 2277 128X.
[7] url https://classes.soe.ucsc.edu/cmps102/Spring04/TantaloAsymp.pdf
[8] Paul E. Black, "big-O notation", in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. 11 March 2005. Retrieved December 16, 2006.
[9] Sarbazi-Azad, Hamid (2 October 2000). "Stupid Sort: A new sorting algorithm". Newsletter (Computing Science Department, Univ. of Glasgow) (599): 4. Retrieved 25 November 2014.

Jehad A. H. Hammad received the M.S. degree in Computer science from the School of Computer Sciences, University Sains, Malaysia (USM) in 2009. His research interests include Algorithms, Parallel and Distributed Computing Architecture, Advance Network and Data Communication, e-learning, and virtual classes. Currently, Hammad is Faculty member at Al-Quds Open University, Bethlehem branch. He was a supervisor and member of inclusive Education team at the Directorate of Education, Bethlehem, Palestine. The Author is looking forward to further research in the field of Algorithms, Distributed Computing and related studies.