Sorting Algorithms

Kenneth Alvarado-Méndez, José Andres Rive-Tencio
ce.kalvarado@gmail.com jose7rivera77@gmail.com
Academic area of Computer Engineering
Instituto Tecnológico de Costa Rica

***Abstract*—One of the fundamental issues in computer science is ordering a list of items. Although there is a huge number of sorting algorithms, sorting problem has attracted a great deal of research; because efficient sorting is important to optimize the use of other algorithms. This paper aims to demonstrate the different run-times between some of the most popular sorting algorithms.**

***Palabras clave*—Sort, Node, List, Swaps, Time Complexity.**

I. Introduction

IN computer science, a sorting algorithm is an algorithm that puts elements of a list in a certain order. Efficient ordering is important to optimize the use of other algorithms (such as search and merge) that require ordered lists for fast execution. The following table shows the run-time pattern from the sorting algorithm depending on the n-elements given as inputs.

Table I: Run-time pattern

Algorithm	Worst	Average	Best
Bubble Sort	$O(n^2)$	$O(n^2)$	O(n)
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Quick Sort	$O(n^2)$	O(n log n)	O(n log n)
Insertion Sort	$O(n^2)$	$O(n^2)$	O(1)
Merge Sort	O(n log n)	O(n log n)	O(n log n)

This research pretends to demonstrate that tries to demonstrate experimentally the execution time of the algorithms, and thus compare them with the theory. Will be compared with the following table.

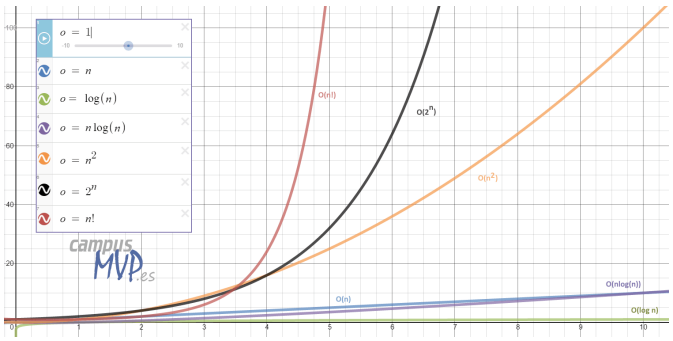

Figure 1: Graph for the best case

1) Best Case: Each sorting algorithm has its own worst, best, and average case, however, in this project will be used as a best case a sorted list, worst case a list inverse sort, and average case a random list propose by the teacher. As a result, some graphs will not match with the theory.

II. Sorting Algorithms

A. BubbleSort

Is a simple sorting algorithm that repeatedly steps through the list to be sorted, compares each pair of adjacent items and swaps them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. Although the algorithm is simple, it is too slow and impractical for most problems

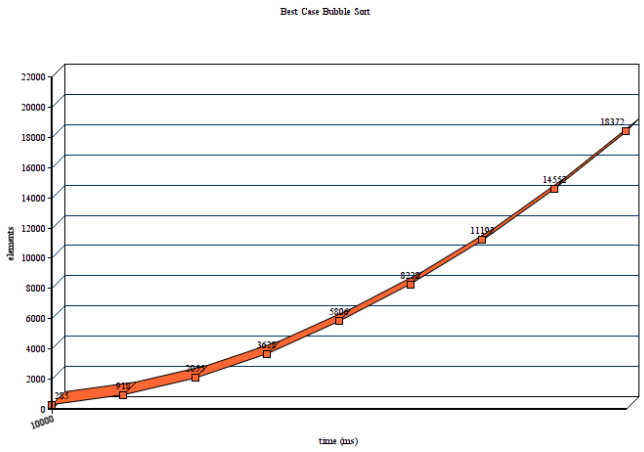

Figure 2: Graph for the best case

1) Best Case: As you can see comparing with the theoretical graph, the best case isn't aproximately O(n), this may be due to the fact that the proposed cases are not the correct ones for the algorithm.

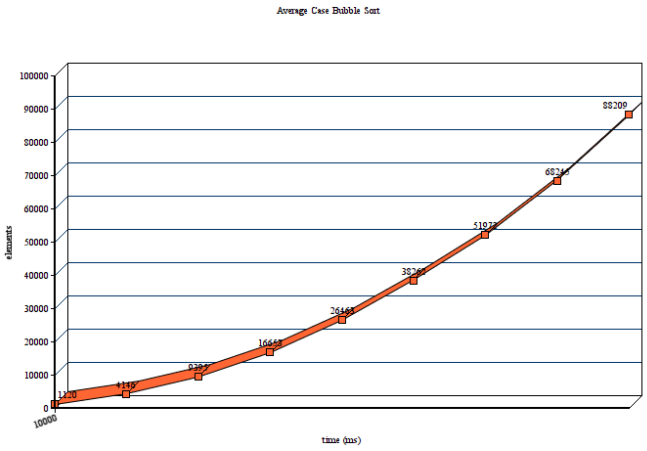

Figure 3: Graph for the average case

2) Average Case: As you can see comparing with the theoretical graph, the average case is aproximately O(n^2)

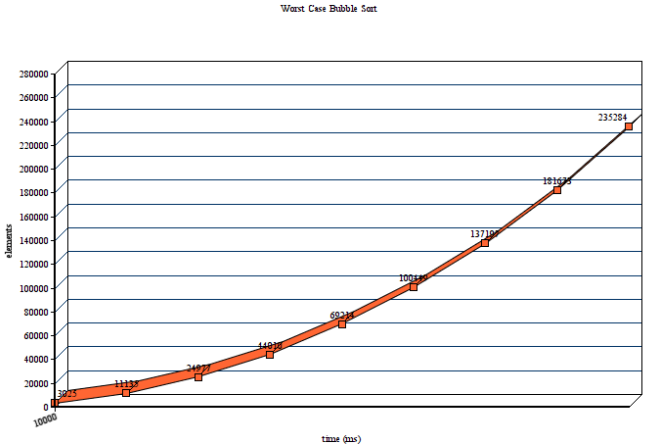

Figure 4: Graph for the worst case

3) Worst Case: As you can see comparing with the theoretical graph, the worst case is aproximately O(n^2)

B. SelectionSort

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

- The subarray which is already sorted.
- Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

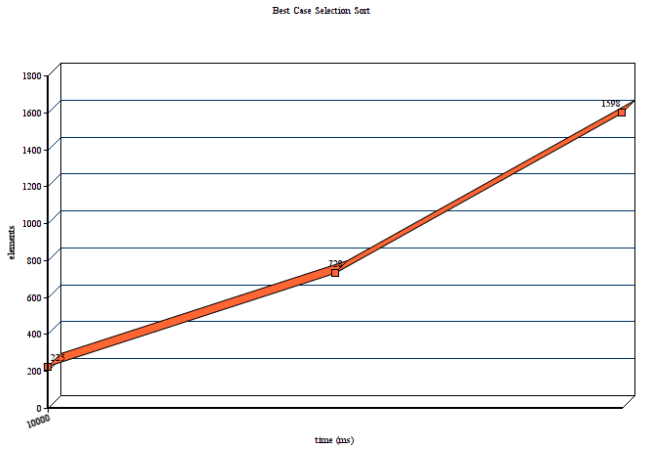

Figure 5: Graph for the best case

1) Best Case: As you can see comparing with the theoretical graph, the best case is aproximately O(n^2)

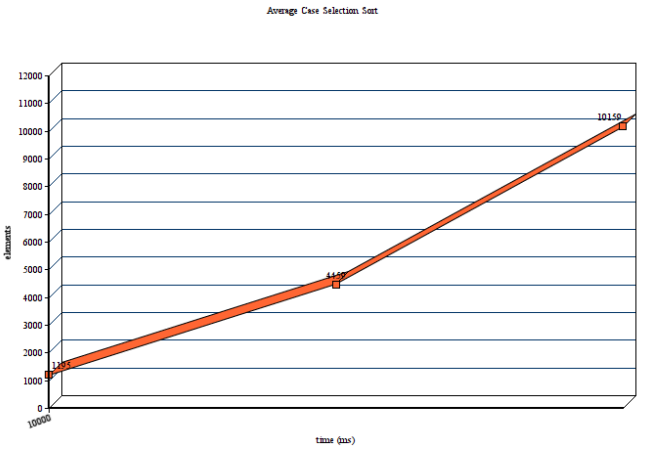

Figure 6: Graph for the average case

2) Average Case: As you can see comparing with the theoretical graph, the average case is aproximately O(n^2)

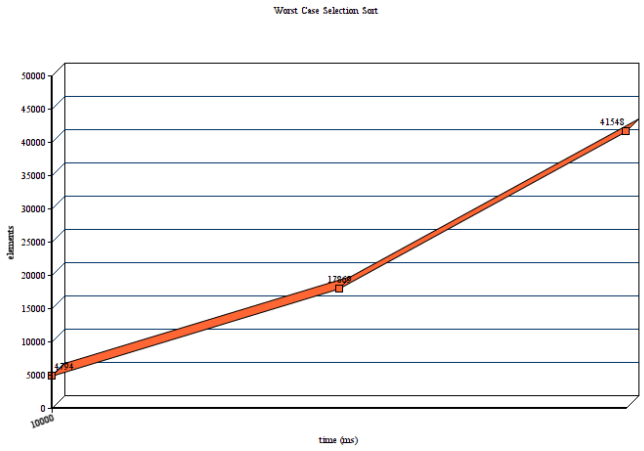

Figure 7: Graph for the worst case

3) Worst Case: As you can see comparing with the theoretical graph, the worst case is aproximately O(n^2)

C. Insertion Sort

Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be inserted in this sorted sub-list, has to find its appropriate place and then it has to be inserted there.

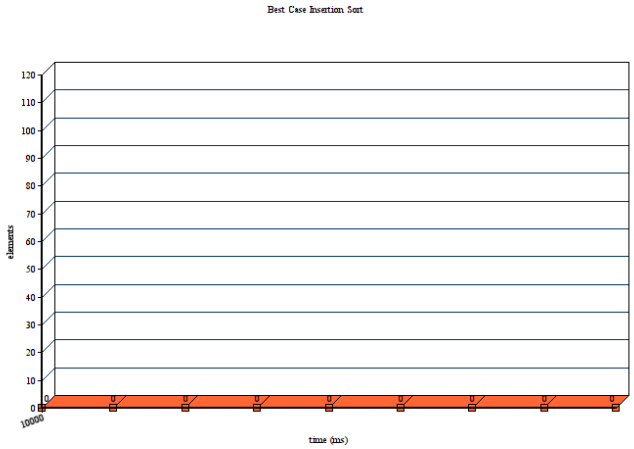

Figure 8: Graph for the best case

1) Best Case: As you can see comparing with the theoretical graph, the best case is aproximately O(1), it's a straight line.

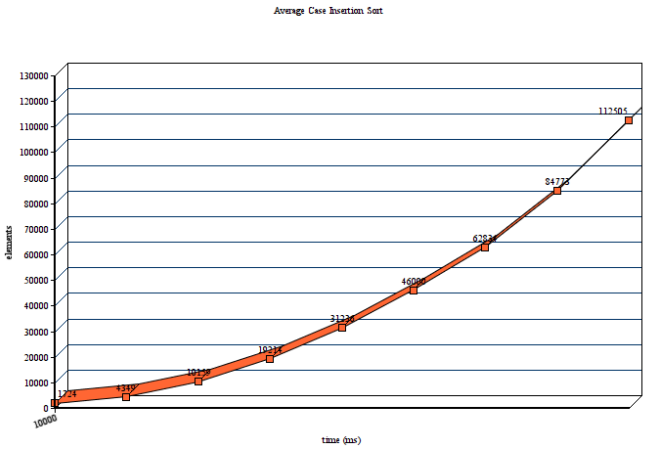

Figure 9: Graph for the average case

2) Average Case: As you can see comparing with the theoretical graph, the average case is aproximately O(n^2)

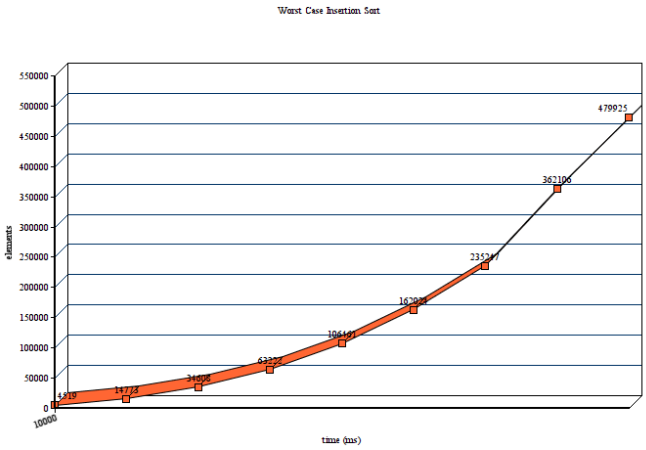

Figure 10: Graph for the worst case

3) Worst Case: As you can see comparing with the theoretical graph, the worst case is aproximately O(n^2)

D. Merge Sort

Divide the unsorted list into n sublists, each containing 1 element (a list of 1 element is considered sorted). Repeatedly merge sublists to produce new sorted sublists until there is only 1 sublist remaining. This will be the sorted list.

1) Best Case: As you can see comparing with the theoretical graph, the best case is aproximately O(n log n)

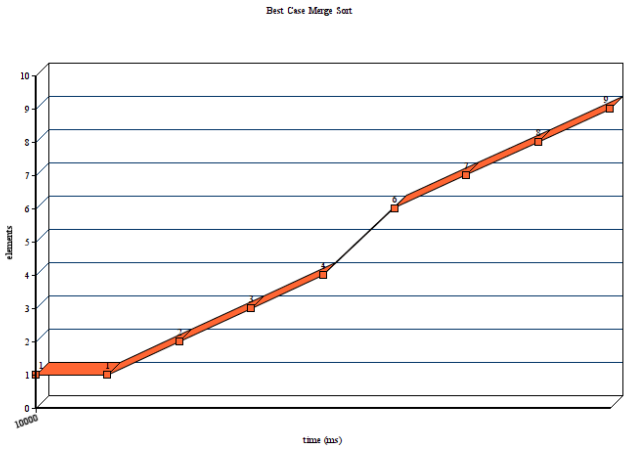

Figure 11: Graph for the best case

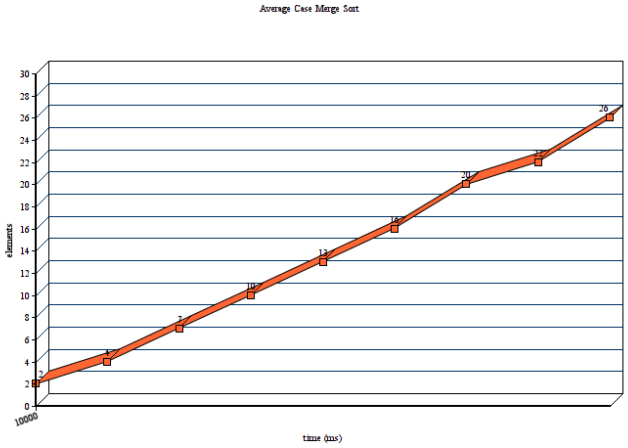

Figure 12: Graph for the average case

2) Average Case: As you can see comparing with the theoretical graph, the average case is aproximately O(n log n)

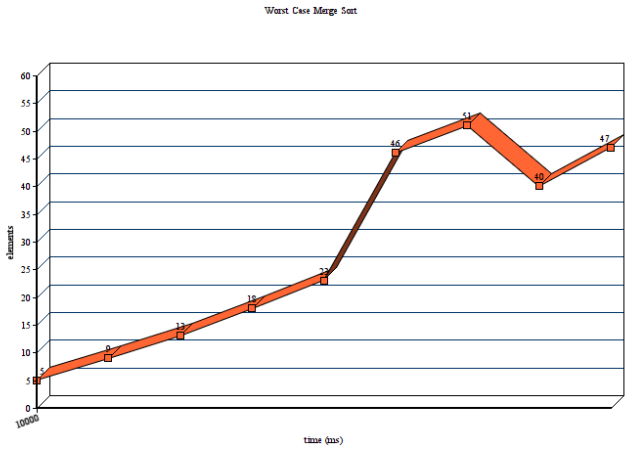

Figure 13: Graph for the worst case

3) Worst Case: As you can see comparing with the theoretical graph, the worst case isn't aproximately O(n log n), this may be due to the fact that the proposed cases are not the correct ones for the algorithm.

E. Quick Sort

Quicksort first divides a large array into two smaller sub-arrays: the low elements and the high elements. Quicksort can then recursively sort the sub-arrays. Basically, it picks an element, called a pivot, from the array.Then reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. Recursively applies the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

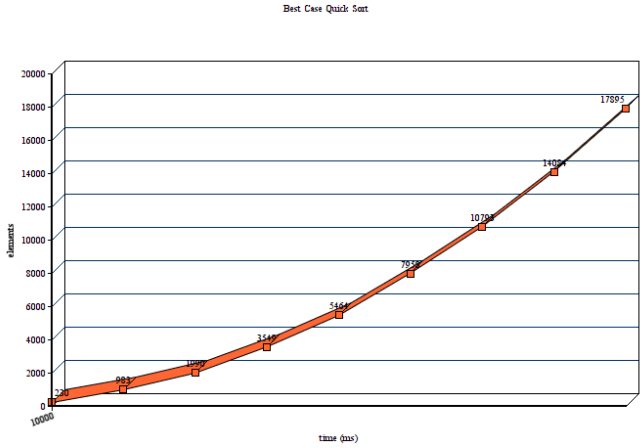

Figure 14: Graph for the best case

1) Best Case: As you can see comparing with the theoretical graph, the best case is aproximately O(n log n)

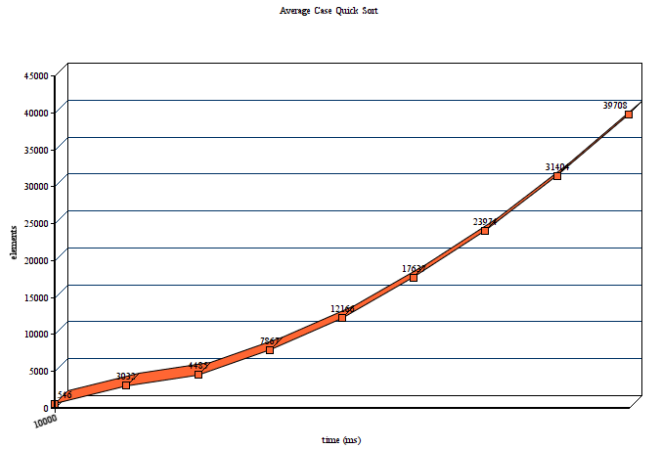

Figure 15: Graph for the average case

2) Average Case: As you can see comparing with the theoretical graph, the average case is aproximately O(n log n)

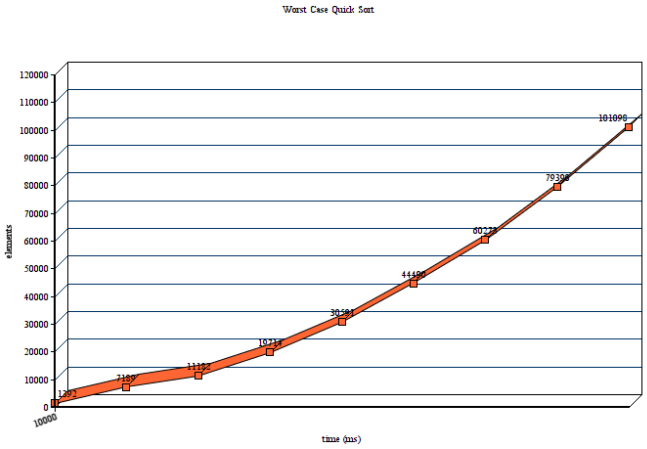

Figure 16: Graph for the worst case

3) Worst Case: As you can see comparing with the theoretical graph, the worst case is aproximately O(n^2)

III. Referencias

References

[1] [Brian W. Kerngham] *The C Programming Language* consultado el 14/11/2017