ESTRUCTURA DE DATOS 1
Código ST0245

Laboratory practice No. 2: Big O Notation

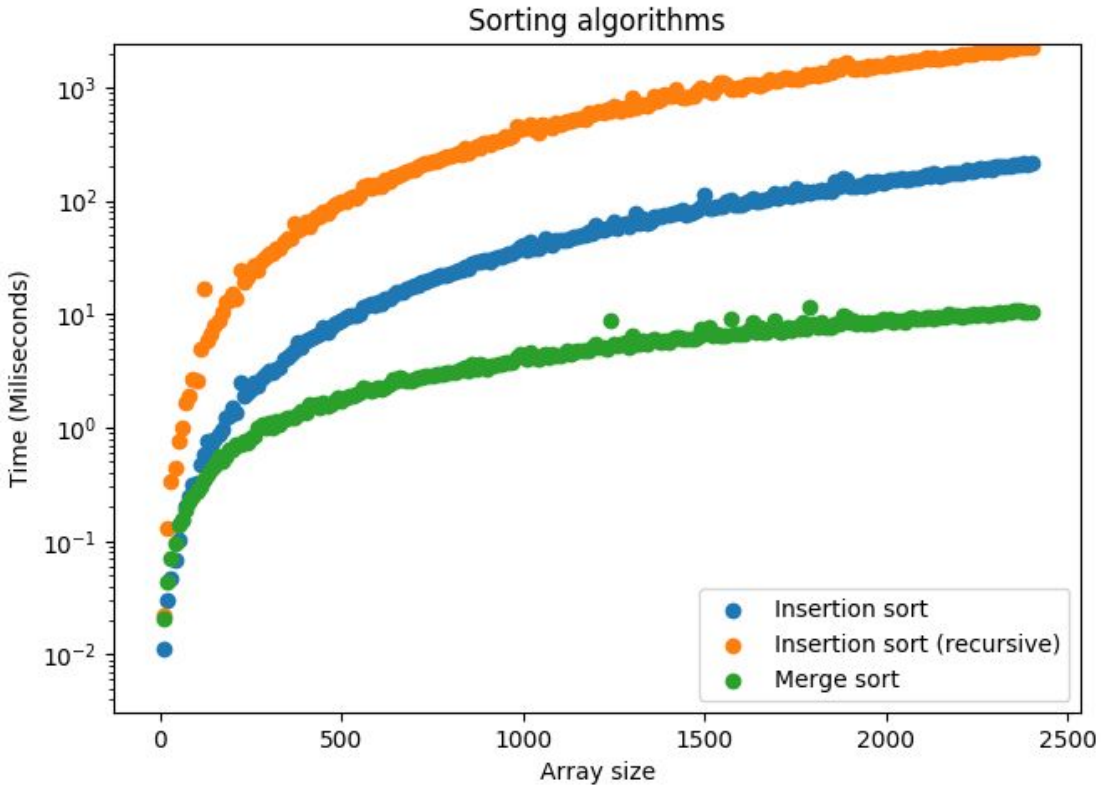
Juan Pablo Ossa Zapata
Universidad Eafit
Medellín, Colombia
jpossaz@eafit.edu.co

Mauricio Jaramillo Uparela
Universidad Eafit
Medellín, Colombia
mjaramillu@eafit.edu.co

3) Practice for final project defense presentation

3.1 and 3.2

The following is a scatter plot for the runtime of three different sorting algorithm implementations.



Note the fact that the recursive implementation of insertion sort is orders of magnitude more complex in both time and space (the later one not being present in the diagram, but definitely

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

considerable: the stack size for the python interpreter had to be increased massively, as well as the maximum recursive depth). This is probably due to the fact that the recursive insertion does not reduce the size of the problem significatively with each recursive step, but instead uses recursion as an alternative to simple iteration. This is contrasted to the implementation of merge sort, which effectively reduces the size of the problem by one half, mitigating the negative effects in performance that arise from the nature of recursive execution.

3.3

According to the data presented, we can see that merge sort is orders of magnitude more efficient than insertion sort. However, the difference between both efficiencies can't be expressed as a single quantity, since the complexity of insertion sort divided by the complexity of merge sort is not a constant, but a function of *n*.

3.4

The use of insertion sort in a video game would be unwise if the amount of elements is considerably high. That is, assuming that merge sort (or other, less complex algorithm) can be implemented and used instead. However, we must recognise that some computational systems (even some designed running video games) don't work with a stack frame of memory, therefore not allowing the implementation of recursive algorithms natively.

Furthermore, the fact that a videogame would have millions of elements is not by itself a contundent reason for insertion sort to not be used. Even if the sorting time is considerably high, the game experience won't be affected as long as the time required to deliver each update remains constant. For instance, a game would still be playable if it presents 15 frames of video every second, potentially doing intensive computations in the background, as long as this frame speed remains consistent (not speeding up and slowing down as the game progresses). Therefore, insertion sort should be avoided in video games where the array being sorted has different size before each sort, since tiny changes in array size could produce noticeable changes in the performance of the game overall. However, it can still be used in games where the array to sort maintains a fixed size during execution.

3.5

The complexity of merge sort is the same for all cases, regardless of the data being passed to the algorithm. Since the best case of insertion sort is less complex than all cases of merge sort, we can conclude that insertion sort will be faster than merge sort in its best case. That is, when the data is already in full or partial order.

3.6

A variable Span is created in the code where the number of elements between two occurrences of the same number will be stored. Then, the array is scrolled in order from position 0 to the last position looking for any number. Then you go through the same array starting from the last number up to 0, looking for the farthest appearance of the previously mentioned number. Having these two apparitions of the same number, we look for how many positions there are between them, for this, we subtract the position of first appearance to the position of the last appearance and we add 1. Then this number is compared with the last maxSpan achieved, to save in the variable again the largest that will give us the answer**.**

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

3.7 Complexity of exercises 2.1

Array2

countEvens: O(n)
centeredAverage: O(n)
sum13: O(n)
has22: O(n)
fizzArray: O(n)

Array3

maxSpan: O(n^2)
fix34: O(n^2)
fix45: O(n^2)
linearIn: O(n^2)
maxMirror: O(n^2)

3.8 Explaining Variables

Array2

countEvens: n is the size of the array.
centeredAverage: n is the size of the array.
sum13: n is the size of the array.
has22: n is the size of the array.
fizzArray: n is the size of the array.

Array3

maxSpan: n is the size of the array.
fix34: n is the size of the array.
fix45: n is the size of the array.
linearIn: n is the size of the array.
maxMirror: n is the size of the array.

4) Practice for midterms

4.1 *c*
4.2 *b*
4.3 *Optional*
4.4 *b*
4.5 *d*
4.6 *a*
4.7 *Optional*
4.8 *a*
4.9 *d*
4.10 *Optional*
4.11 *c*
4.12 *b*
4.13 *Optional*
4.14 *b*

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473