Laboratory No. II: NOTATION OR GRANDE

Camila Barona Cabrera
Universidad Eafit
Medellín, Colombia
cbaronac@eafit.edu.co

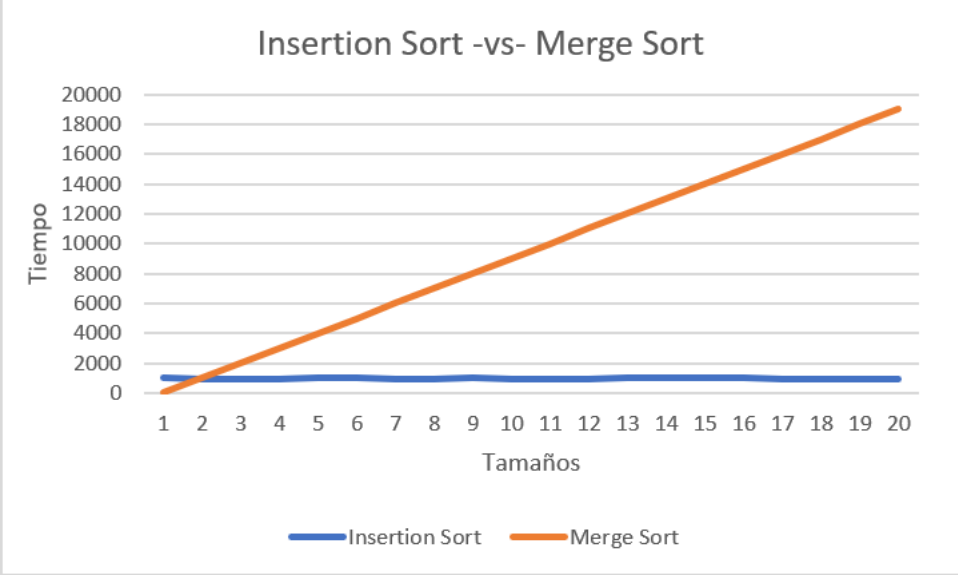
Mariana Gómez Piedrahita
Universidad Eafit
Medellín, Colombia
mgomezp10@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1.

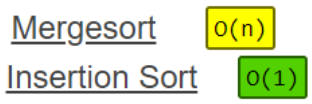
	TIEMPO	
TAMAÑOS	INSERTION SORT	MERGE SORT
1	1001	0
2	1000	1000
3	1000	2002
4	1000	3002
5	1001	4004
6	1008	5008
7	1000	6008
8	1000	7004
9	1002	8003
10	1000	9004
11	1000	10006
12	1000	11026
13	1001	12012
14	1017	13006
151	1001	14008
16	1001	15009
17	1000	16006
18	1000	17010
19	1000	18013
20	1000	19008

2.



2. Merge sort is less efficient with respect to insertion sort for large arrays. As we can see in the graph proposed in numeral 2, insertion sort is kept constant and for databases with millions of elements it would be much faster and more effective than the Merge Sort that delays more in its execution.

You can see this when calculating its complexity in terms of the memory space required:



Images taken from:http://bigocheatsheet.com/

3. The maxSpan exercise what is done is to compare which number is more to the left and more to the right of the arrangement, going from right to left and from left to right the arrangement, seeing which has more distance between them, for example:
maxSpan ([1, 4, 2, 1, 4, 1, 4]) → 6 In this case, the algorithm starts by looking at the number 1, keeps going through the arrangement and observes that there is another 1 in position 4, but nevertheless follows its execution and observe another in position 6, which is much farther than previously found, this would become the maxSpan to return.
maxSpan ([1, 4, 2, 1, 4, 4, 4]) → 6 In this second example, the algorithm starts with number one and finds that the other one farther is in position 3, but continues its execution with the number 4 and find that the farthest is in position 6 for which this happens to be the maxSpan to return.
This algorithm works initially verifying if the length of the array is greater than 0, if so, a variable called maxSpan will be created that will start in 1, continuing with two cycles, one that will start at zero

and travel to the array length and the nested cycle travel from right to left, starting at the length of the array minus one, continuing to verify if the arrangement in the position of the second cycle is equal to that of the first cycle, if so, an aux or counter variable will be created in which they will be subtracted these positions and 1 will be added, when this part is executed, if the aux variable is greater than the maxSpan created initially, this value will be granted to the maxSpan and the cycle will be exited to return the found value. If the length of the array is equal 0, a 0 will be returned.

ARRAY 2	COMPLEXITY	EXPLANATION
countEvents	O(n)	The parameter nums, is an array of integers, which will be used to count the numbers in the array that are even.
Sum28	O(n)	The parameter nums, is an array of integers, within which we add the 2 to see if it results in 8.
Sum13	O(n)	The parameter nums, is an array of integers, in this will be added the numbers that are except for the number 13.
FizzArray	O(n)	The parameter n, is a number that will be used as a reference to create the size of an array, that is, from 1 to n, printing numbers from 0 to n-1
Lucky13	O(n)	The parameter nums, is an array of integers, in which it will be verified if it has a 1 or a 3.

ARRAY 3	COMPLEXITY	EXPLANATION
canBalance	O(n^2)	The parameter nums, is an array of integers, which will identify whether the sum of one side is equal to the other or not.
seriesUp	O(n^2)	The parametron n, is a number that creates an array from 1 to n with its respective pair, that is: {1, 1,2, 1,2,3, ...n}
linearln	O(n)	The outer parameter and the inner parameter are a set of arrays of integers, which must be verified if the outer is located inner.
countClumps	O(n)	The nums parameter is an array of integers.

4) Test simulation

1. C
2. D
3. B

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

4. B
5. D
6. A
7. T(n)=T(n-1)+C

7.2 O(n)

8. B
9. D
10. C
11. C
12. B
13. A

5) Recommended reading (optional)

As its name says, the central theme of Chapter 2 is the complexity of the algorithms, speaking at the beginning of temporal complexity, defined as the time needed to execute an algorithm; they also describe how an algorithm with the naked eye, thanks to the O notation, has "greater execution time" than another; but examining it more thoroughly we could notice that the one that has "less execution time" requires more time to execute one step than the other, which makes it much longer.
Around the whole chapter is the analysis of the best case, average and worst case in different methods applied to the algorithms, some as: ordering method, binary search algorithm, order by direct selection, quick sort algorithm, heap sort ordering; among others, with the common goal of achieving greater efficiency, analysis and ease in the use of algorithms.

Complejidad de los algoritmos

Coste en tiempo y recursos que consume un algoritmo para encontrar la solución

Complejidad temporal

Tiempo necesario para ejecutar un algoritmo

Cota inferior de un problema

Es la complejidad mínima requerida por cualquier algoritmo que pueda aplicarse para resolverlo.

Mejor, promedio y peor de los caos

-Ordenamiento por selección directa:
Se encuentra el número más pequeño y s ehace que este npumero ocupe la posición 1, se repite esto con los numeros restantes hasta que se encuentre el número más grande

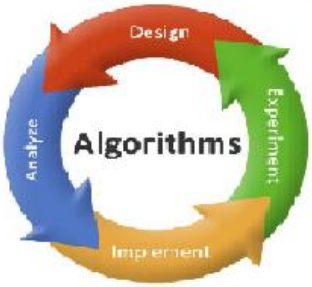
-Algoritmo Quick Sort:
Divide el problema en dos subpoblemas, solucionandolos de manera independiente y después los resultados se unen.

-Inserción directa:
Método de ordenamiento, que cuenta como el mejor de los casos cuando el grupo de elementos ya están ordenados, el peor caso cuando están desordenados y el promedio cuando ya se han ordenado (i-1) datos.

-Algoritmo de búsqueda binaria:
Después de ordenar un conjunto números, el algoritmo empieza desde la parte media de la secuencia, si el punto de prueba es igual al punto medio de la secuencia, finaliza el algoritmo, en caso contrario se busca de izquierda a derecha de la secuencia en forma recurrente

La cota inferior del peor caso del ordenamiento:

-Ordenamiento Heap Sort, cuya complejidad temporal es O(n logn)



La cota inferior del caso promedio del ordenamiento:

-Para determinar la cota inferior de la complejidad temporal del ordenamiento es necesario determinar la mínima longitud de la ruta externa de todos los arboles binarios posibles con n! nodos hoja.

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co