Laboratorio Nro. II: NOTACIÓN O GRANDE

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

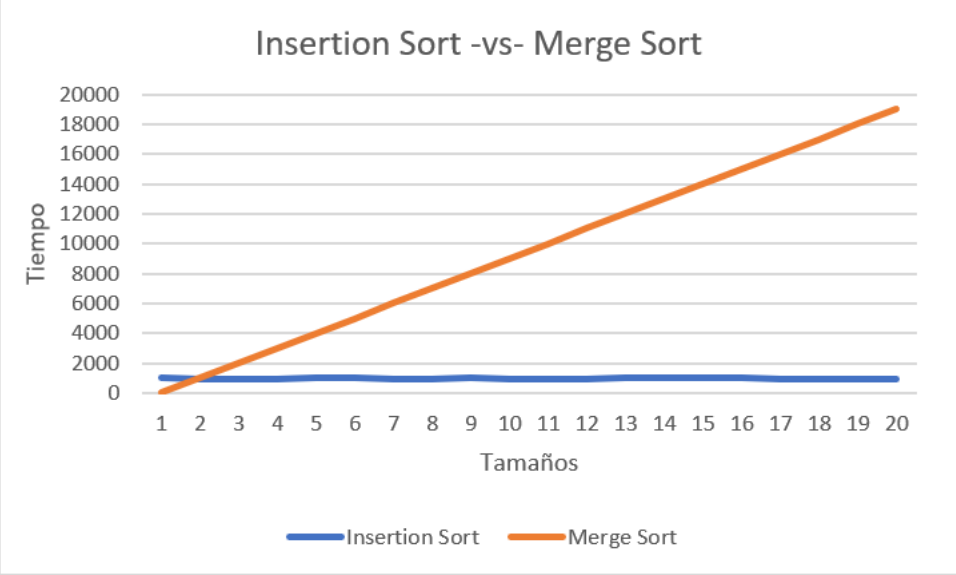
Mariana Gómez Piedrahita
Universidad Eafit
Medellín, Colombia
mgomezp10@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1.

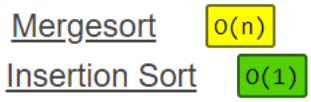
	TIEMPO	
TAMAÑOS	INSERTION SORT	MERGE SORT
1	1001	0
2	1000	1000
3	1000	2002
4	1000	3002
5	1001	4004
6	1008	5008
7	1000	6008
8	1000	7004
9	1002	8003
10	1000	9004
11	1000	10006
12	1000	11026
13	1001	12012
14	1017	13006
151	1001	14008
16	1001	15009
17	1000	16006
18	1000	17010
19	1000	18013
20	1000	19008

2.



3. Es menos eficiente merge sort con respecto a insertion sort para arreglos grandes. Como Podemos observer en la gráfica planteada en el numeral 2, insertion sort se mantiene constante y para bases de datos con millones de elementos sería mucho más rapida y efectiva que el Merge Sort que demora más en su ejecución, evitando mayoritariamente el gasto de memoria innecesario.

Esto podemos verlo al calcular su complejidad en cuanto al espacio de memoria requerido:



Imagenes tomadas de: http://bigocheatsheet.com/

4. El ejercicio maxSpan lo que hace es comparar que numero está más a la izquierda y más a la derecha del arreglo, recorriendo de derecha a izquierda y de izquierda a derecha el arreglo, viendo cual tiene más distancia entre el mismo, por ejemplo:

maxSpan([1, 4, 2, 1, 4, 1, 4]) → 6 En este caso, el algoritmo empieza viendo el numero 1, sigue recorriendo el arreglo y observa que hay otro 1 en la posicion 4 más sin embargo sigue su ejecución y observa otro en la posición 6, el cual está mucho más lejano que el encontrado anteriormente, este llegaria a ser el maxSpan a retornar.

maxSpan([1, 4, 2, 1, 4, 4, 4]) → 6 En este Segundo ejemplo, el algoritmo inicia con el número uno y encuentra que el otro uno más lejano está en la posición 3, pero continua su ejecución con el número 4 y encuentra que el más lejano está en la posicion 6 por lo cuál este pasa a ser el maxSpan a retornar.

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

Este algoritmo funciona inicialmente verificando si la longitude del arreglo es mayor que 0, de ser así se creara una variable llamada maxSpan que se iniciará en 1 prosiguiendo con dos ciclos, uno que iniciara en cero y recorrera hasta la longitude del arreglo y el ciclo anidado que recorrera de derecha a izquierda, iniciando en la longitude del arreglo menos uno, continuando con verificar si el arreglo en la posicion del Segundo ciclo es igual a la del primer ciclo, de ser así se creara una variable aux o Contador en la cual se restaran estas posiciones y se le sumara 1, cuando se ejecute esta parte, si la variable aux es mayor que el maxSpan creado inicialmente se le otorgará este valor al maxSpan y se saldra del ciclo para retornar el valor encontrado. Dado el caso que la longitud del arreglo sea igual 0 se retornara un 0.

Código:

```
public int maxSpan(int[] nums) {
   if (nums.length > 0) {
      int maxSpan = 1;
      for (int i = 0; i < nums.length; i++)
         for (int j = nums.length - 1; j > i; j--)
            if (nums[j] == nums[i]) {
               int aux = (j - i) + 1;
               if (aux > maxSpan) maxSpan = aux;
               break;
            }
      return maxSpan;
   } else return 0;
}
```

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

ARRAY 2	COMPLEJIDAD	EXPLICACIÓN
countEvents	O(n)	El parámetro nums, es un arreglo de numeros enteros, que servira para contar los numeros en el arreglo que sean pares.
Sum28	O(n)	El parámetro nums, es un arreglo de numeros enteros, dentro del cual sumaremos los 2 para ver si da como resultado 8.
Sum13	O(n)	El parámetro nums, es un arreglo de numeros enteros, en este se sumaran los numeros que hay a excepcion del número 13.
FizzArray	O(n)	El parámetro n, es un numero que se tendra como referencia para crear el tamaño de un arreglo, es decir desde 1 hasta n, imprimiendo numeros del 0 a n-1
Lucky13	O(n)	El parámetro nums, es un arreglo de numeros enteros, en el cual se verificara si este posee un 1 o un 3.

ARRAY 3	COMPLEJIDAD	EXPLICACIÓN
canBalance	O(n^2)	El parámetro nums, es un arreglo de numeros enteros, que identificará si la suma de un lado es igual a la del otro o no.
seriesUp	O(n^2)	El parametron n, es un numero que creara un arreglo desde el 1 hasta n con su respective pareja, es decir: {1, 1,2, 1,2,3, ….n}
linearln	O(n)	El parámetro outer y el parámetro inner son un conjunto de arreglos de numeros enteros, los cuales dentor la ejecución debera verificarse si en outer se encuentra inner.
countClumps	O(n)	El parámetro nums es un arreglo de número enteros.

4) Simulacro de Parcial

1. C
2. D
3. B
4. B
5. D
6. A

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

7. T(n)=T(n-1)+C
7.2 O(n)
8. B
9. D
10. C
11. C
12. B
13. A

5) Lectura recomendada (opcional)

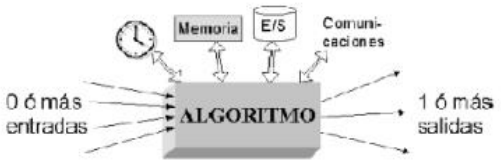
Como su propio nombre lo dice, el tema central del capítulo 2 es la complejidad de los algoritmos, hablando en un principio de la complejidad temporal, definida como el tiempo necesario para ejecutar un algoritmo; así mismo describen como un algoritmo a simple vista, gracias a la notación O, tiene "mayor tiempo de ejecución" que otro; pero examinándolo más a fondo podríamos notar que el que tiene "menor tiempo de ejecución" requiere más tiempo para ejecutar un paso que el otro, lo que lo hace mucho mayor.
Alrededor de todo el capítulo se trata el análisis del mejor caso, promedio y el peor de los casos en diferentes métodos aplicados a los algoritmos, algunos como: método de ordenamiento, algoritmo de búsqueda binaria, ordenamiento por selección directa, algoritmo quick sort, ordenamiento heap sort; entre otros, teniendo como fin común lograr la mayor eficiencia, análisis y facilidad en el uso de algoritmos.

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

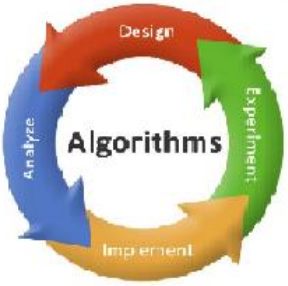
-Algoritmo Quick Sort:
Divide el problema en dos subpoblemas, solucionandolos de manera independiente y después los resultados se unen.

-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