Laboratorio Nro. 2: Notación O grande

Alejandro Cano Munera
Universidad Eafit
Medellín, Colombia
acanom@eafit.edu.co

Jorge Luis Herrera Chamat
Universidad Eafit
Medellín, Colombia
jlherrerac@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. Tiempos de ejecución de los algoritmos en milisegundos.

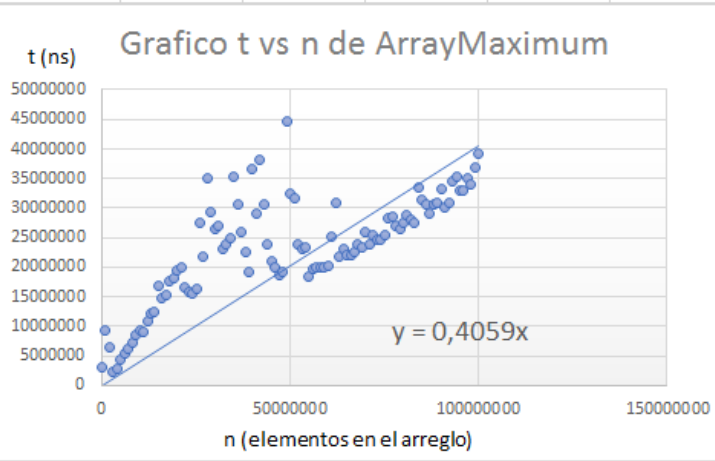
	N = 100.000	N = 1'000.000	N = 10'000.000	N = 100'000.000
Array Sum	4	8	10	74
Array Maximum	1	3	5	40
Insertion Sort	3135	234842	Más de 5 minutos	Más de 5 minutos
Merge Sort	47	327	2779	30623

2. Graficas:

ArraySum: O(n)

Laboratorio 1.
ArraySum

Tabla de valores	
n	t (ns)
100000	3104400
1000000	13502051
10000000	14708747
100000000	60113595

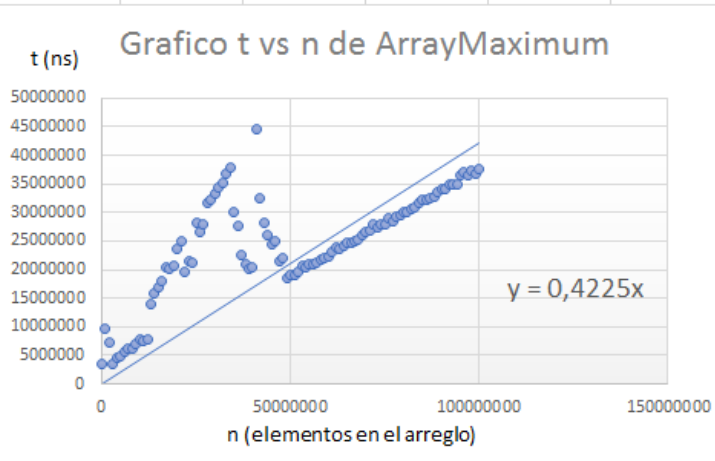


*Se tomaron mas valores para verificar la tendencia en la grafica Los mas representativos aparecen en la tabla

ArrayMaximum: O(n)

Laboratorio 1.
ArrayMaximum

Tabla de valores	
n	t (ns)
100000	3588928
1000000	8745122
10000000	18996901
100000000	59566747



*Se tomaron mas valores para verificar la tendencia en la grafica Los mas representativos aparecen en la tabla

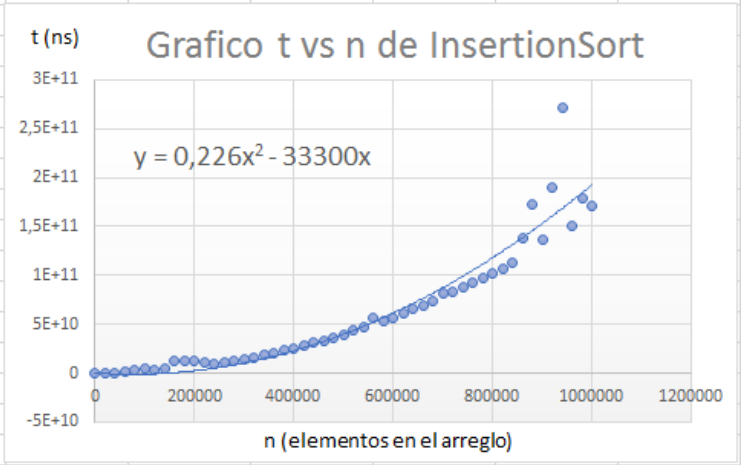
DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

Insertion Sort: O(n^2)

Laboratorio 1.

InsertionSort

Tabla de valores	
n	t (ns)
100000	4342998886
1000000	1,71932E+11
10000000	más de 5 mins
100000000	más de 5 mins



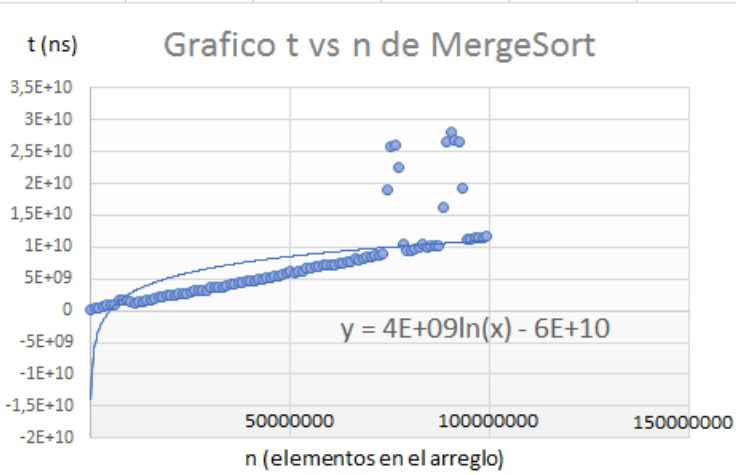
*Se tomaron mas valores para verificar la tendencia en la grafica Los mas representativos aparecen en la tabla

Merge Sort: O(n log(n))

Laboratorio 1.

MergeSort

Tabla de valores	
n	t (ns)
100000	48766407
1000000	335874449
10000000	1372147237
100000000	1,162E+10



*Se tomaron mas valores para verificar la tendencia en la grafica Los mas representativos aparecen en la tabla

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

3. Se concluye que los tiempos obtenidos en el laboratorio y los resultados obtenidos con la notación O son muy similares, ya que las líneas de tendencia en ambos casos siguen el mismo camino, dando cuenta del comportamiento del algoritmo respecto al tiempo, dependiendo del tamaño del arreglo que se esté manejando.

4. Insertion Sort para valores grandes de n, es decir, para arreglos de gran tamaño, se tiende a volver poco eficiente ya que su ejecución conlleva mucho tiempo, cabe resaltar que Insertion Sort es O(n^2), lo que nos da la idea de que su desempeño no es el mejor.

5. En ArraySum el tiempo de ejecución del algoritmo siempre va a depender del tamaño del arreglo, ya que para saber dicha suma debemos recorrer cada posición. Para los valores grandes de n igualmente se tendrían que recorrer todas las posiciones del arreglo, por ende, si el arreglo es muy grande se tiene un tiempo de ejecución mayor. Los tiempos en ArraySum no crecen tan rápido como en Insertion Sort ya que para el funcionamiento de este algoritmo esencialmente se debe recorrer el arreglo sumando los valores allí contenidos, es decir, en el peor de los casos tendríamos O(n), mientras que en Insertion Sort aparte de recorrer todo el arreglo se deben hacer modificaciones moviendo los elementos de posición, por lo que se tiene que en el peor de los casos es O(n^2), que sería recorrer el arreglo hasta el final, y tener que mover de nuevo todos los elementos de este.

6. Merge Sort es mucho más eficiente que Insertion Sort para arreglos de mayor tamaño ya que Merge Sort funciona con el principio de dividir el arreglo (en dos mitades recursivamente), lo cual resulta muy apropiado a la hora de trabajar con grandes volúmenes de datos, ya que se resolverán sub-problemas hasta resolver el problema en si. Sin embargo, Insertion Sort tiende a ser más efectivo para arreglos de un tamaño menor ya que al tratarse de un volumen más pequeño de datos, no representara un gran problema para el ordenador mover los elementos de este cuando se necesite (a comparación de un volumen de datos mayor), es decir, el ordenamiento que se efectúa moviendo los elementos directamente en el arreglo, no se verá muy afectado en tiempo de ejecución con valores pequeños de “n”, ya que los valores que se moverán en el peor de los casos serán relativamente pequeños.

7. **Ejercicio maxSpan:** Cuando se recorre el arreglo, para cada elemento por el que se pase, se vuelve a recorrer todo el arreglo para buscar el último elemento de éste que sea igual al elemento en el que se está recorriendo, entonces se calcula el valor de ese span y se guarda en una variable que contiene el span más grande, y por cada recorrido del arreglo se compara el valor del span con el valor de la variable del span más grande, a fin de mantener la variable con el valor del máximo span. Su complejidad es O(n^2) donde n es el número de elementos del arreglo.

8. **Complejidad de los ejercicios en línea:**
   Array 2:
   1. BigDiff: T(n) = C1*n + C2 es O(n)
   2. CountEvens: T(n) = C1*n + C2 es O(n)
   3. EvenOdd: T(n) = C1*n + C2 es O(n)
   4. FizzBuzz: T(n) = C1*n + C2 es O(n)
   5. HaveThree: T(n) = C1*n + C2 es O(n)

   Array 3:
   1. Fix34: T(n) = C1*n*n + C2 es O(n^2)
   2. Fix45: T(n) = C1*n*n + C2 es O(n^2)
   3. LinearIn: T(n) = C1*n*m + C2 es O(n*m)
   4. SquareUp: T(n) = C1*n + C2 es O(n)
   5. canBalance: T(n) = C1*n*n + C2 es O(n^2)

9. **Significado de las Variables:**
   Array 2:
   1. BigDiff: En este ejercicio n es el número de elementos del arreglo sobre el cual se trabajará.
   2. CountEvens: En este ejercicio n es el número de elementos del arreglo sobre el cual se trabajará.
   3. EvenOdd: En este ejercicio n es el número de elementos del arreglo sobre el cual se trabajará.
   4. FizzBuzz: En este ejercicio n es el número de elementos del arreglo sobre el cual se trabajará.
   5. HaveThree: En este ejercicio n es el número de elementos del arreglo sobre el cual se trabajará.

Array 3:
1. Fix34: En este ejercicio n es el número de elementos del arreglo sobre el cual se trabajará.
2. Fix45: En este ejercicio n es el número de elementos del arreglo sobre el cual se trabajará.
3. LinearIn: En este ejercicio n es el número de elementos del arreglo inner y m es el número de elementos del arreglo outer.
4. SquareUp: En este ejercicio n es el número de elementos del arreglo sobre el cual se trabajará.
5. canBalance: En este ejercicio n es el número de elementos del arreglo sobre el cual se trabajará.

4) Simulacro de Parcial

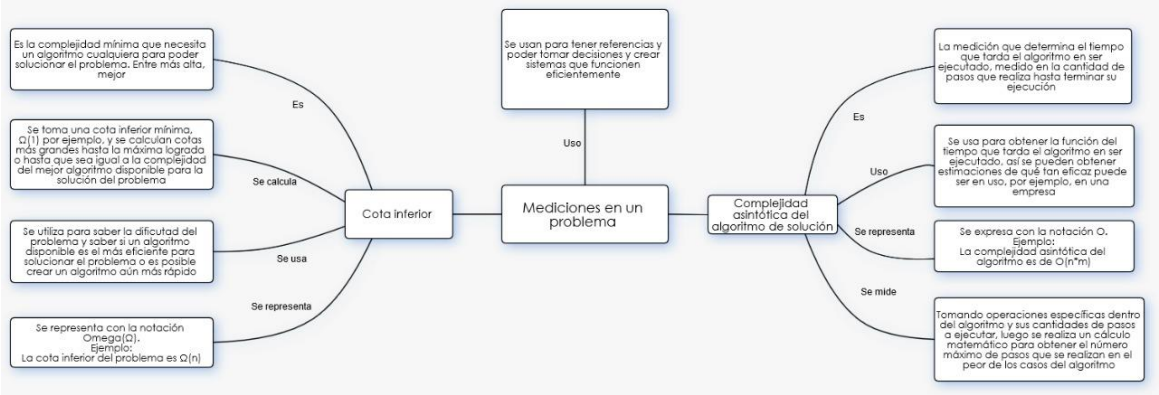
1. C.
2. B.
3. B.
4. B.
5. D.

5. Lectura recomendada

a) Título: Complejidad de los algoritmos y cotas inferiores de los problemas.

b) Resumen: La complejidad temporal de un algoritmo se usa para saber qué tan eficaz es para resolver el problema para el que fue creado. Esta complejidad se basa en operaciones específicas dentro del algoritmo y en un cálculo matemático con el que se define la cantidad de pasos que debe hacer el algoritmo con respecto al tamaño del problema para completar su ejecución, ya que, si se midiese por el tiempo de ejecución, el resultado sería muy subjetivo, puesto que hay muchos factores externos que influirían, como el procesador o el lenguaje usado.
En la resolución de problemas, la complejidad del algoritmo de solución no es lo único que se puede medir. La dificultad de un problema es una medición muy útil y determina la complejidad mínima que necesita un algoritmo para poder solucionar dicho problema en el peor de los casos. A esta dificultad se le llama cota inferior del problema y se expresa en la notación Omega(Ω). Se pueden calcular dos tipos de cotas inferiores, la cota inferior del peor caso y la cota inferior del caso promedio. Las cotas inferiores se calculan mediante análisis teóricos, no por pura suposición.

c) Mapa de Conceptos:



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