Laboratorio Nro. 02: Notación O grande

Agustín Nieto García
Universidad Eafit
Medellín, Colombia
anietog1@eafit.edu.co

David Immanuel Trefftz Restrepo
Universidad Eafit
Medellín, Colombia
ditrefftzr@eafit.edu.co

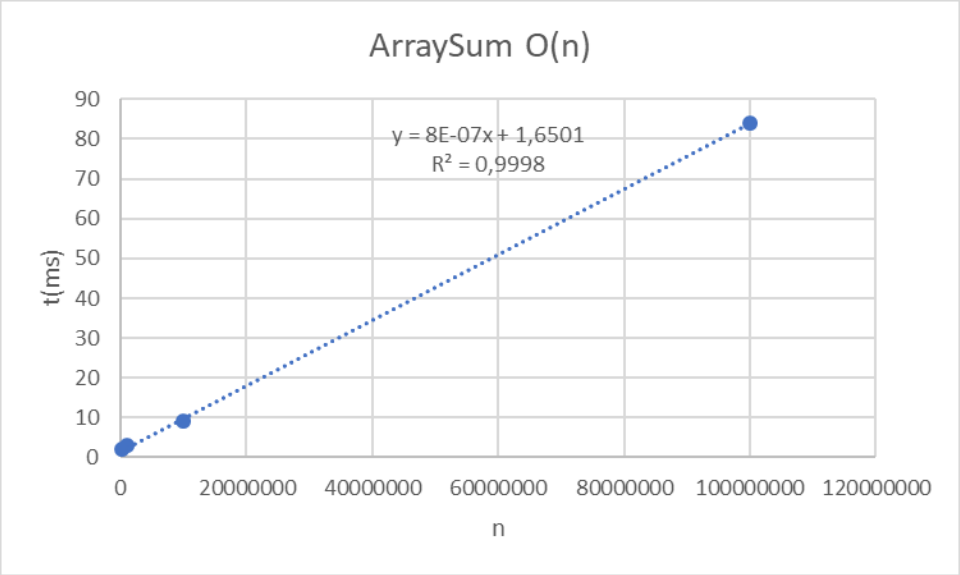
3) Simulacro de preguntas de sustentación de Proyectos

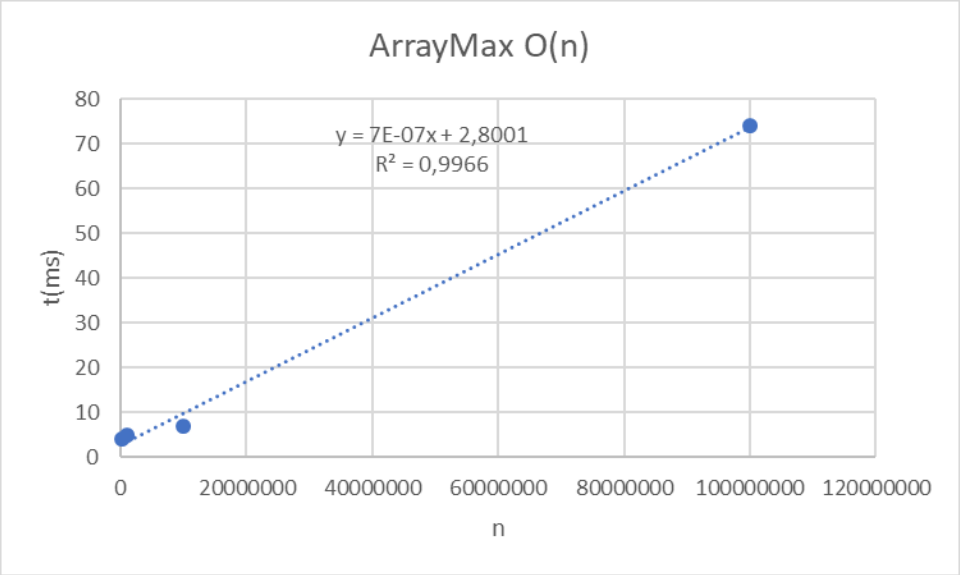
1. Tiempos en milisegundos.

	n=100.000	n=1'000.000	n=10'000.000	n=100'000.000
Array Sum	2ms	3ms	9ms	84ms
Array Max	4ms	5ms	7ms	74ms
Insertion Sort	55ms	2955ms	360654ms	t > 10mins
Merge Sort	35ms	287ms	2114ms	OutOfMemoryError*

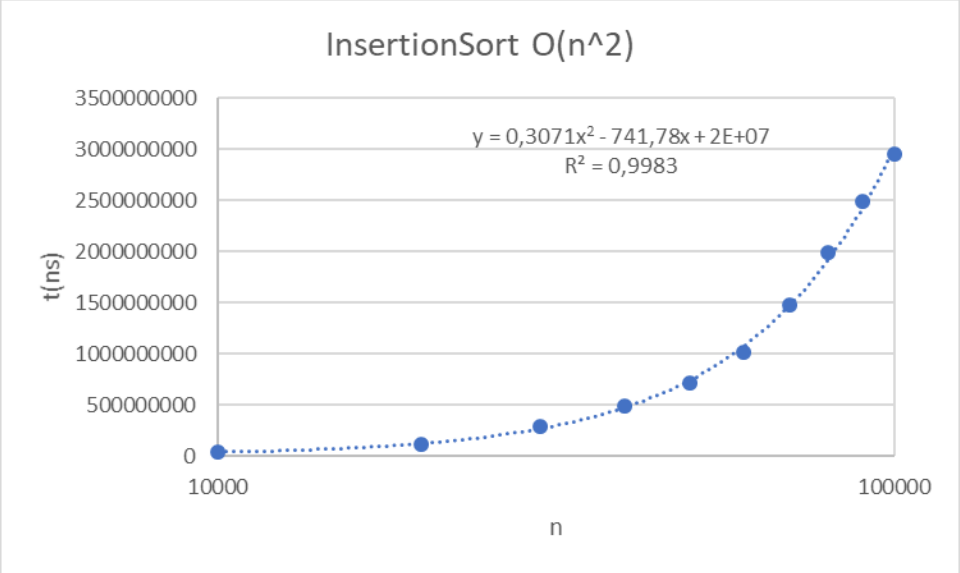
*No pudimos ampliar el heap.

2. Gráficas n vs t.

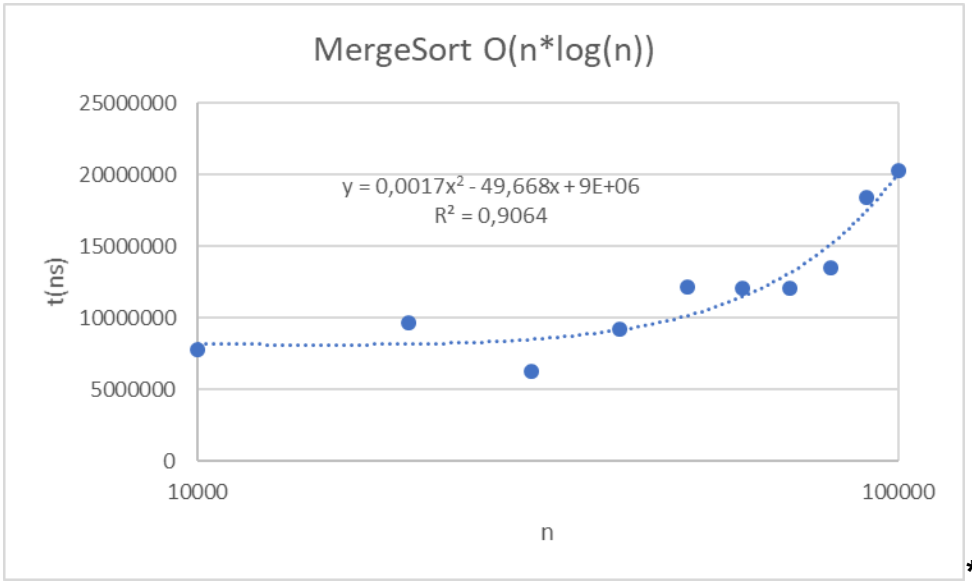




*



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

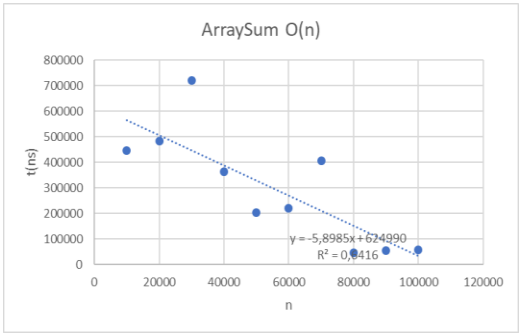


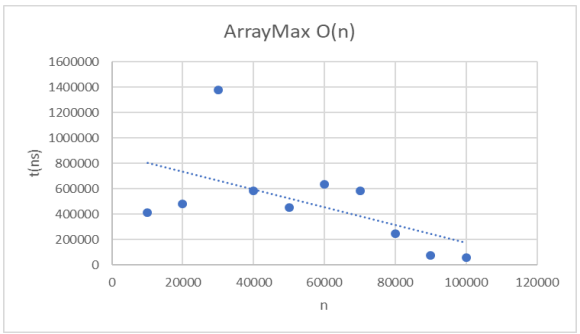
**

*Acerca de ArraySum y ArrayMax, no utilizamos los ejes logarítmicos de Excel en caso de una función lineal, debido a que hace que la gráfica se asemeje a una parábola, lo cual podría generar confusión.

**Acerca de la gráfica de MergeSort, su complejidad no es de n^2, sin embargo, Excel no ofrece la posibilidad de graficar una función compuesta, como lo es n*log(n). Lo que más se asemeja, en este caso, es una función cuadrática.

Acerca de la diferencia de "n" para las diferentes funciones, seleccionamos valores que permitieran obtener una gráfica con sentido. Por ejemplo, no pudimos utilizar las siguientes muestras, debido a que los resultados que arrojaban no tenían sentido.





3. Conclusiones resultados experimentales vs teóricos (t vs notación O).

4. **¿Qué sucede con insertionSort para valores grandes de n?**

   Debido a su complejidad $O(n^2)$, para valores grandes de n es bastante lento, sin embargo, a diferencia de Merge sort, su espacio en memoria no aumenta.

5. **¿Qué sucede con arraySum para valores grandes de n? ¿Por qué los tiempos no crecen tan rápido como Insertion Sort?**

   No sucede nada particular, su complejidad es O(n) y su consumo de espacio es O(1). Así, tiene un comportamiento lineal para todo n. Sin embargo, se puede resaltar el hecho de que para valores grandes de n su velocidad es, por mucho, menor a la de cualquier otra función O(log(n)).

6. **¿Qué tan eficiente es Merge sort con respecto a Insertion sort para arreglos grandes? ¿Qué tan eficiente es Merge sort con respecto a Insertion sort para arreglos pequeños?**

   En cuanto al tiempo, Merge sort es siempre más rápido que Insertion sort, tanto para arreglos grandes como pequeños, la única desventaja en cuanto al rendimiento de Merge sort respecto a Insertion sort para arreglos grandes radica en el hecho de que Merge sort consume O(n) espacio extra en memoria, mientras Insertion sort consume O(1). Así, si no hay mucha memoria, lo mejor sería utilizar Insertion sort, pero si el afán es la velocidad, es preferible el Merge sort.

7. **Explicación de MaxSpan.**

```
public int maxSpan(int[]  nums) {
    int max = 0;
    for (int i = 0; i < nums.length;  i++) {

        int j = nums.length  - 1;
        for (;; j--) {
            if (nums[j]  == nums[i])  {
                break;//n*n*C
            }
        }

        int span = j - i + 1;
```

```
        if (span > max) {
            max = span;
        }
    }
    return max;
}
```

El método maxSpan*, dado un arreglo de enteros, calcula la "amplitud" máxima entre dos valores iguales, inclusivos, dentro de este, por ejemplo, para el arreglo [1, 2, 1, 1, 3] retornaría 4 debido a que entre los dos '1's más lejanos hay 4 elementos, incluyéndolos, para [] retornaría 0 y para uno similar a [1,2,3,4,5] retornaría 1 debido a que cada número tiene una "amplitud" de 1.

¿Cómo lo hace?
Lo principal es la variable que contiene el span máximo, que se inicializa en 0, luego, si el arreglo tiene un tamaño mayor a 0 entra en el primer ciclo, encargado de ir moviéndose hacia la derecha para cambiar el valor con el que se calculará la amplitud. Ahora, dentro de este ciclo, se empieza otro que va desde el último elemento del arreglo hasta uno que tenga igual valor a aquel con el que estamos calculando la amplitud, es decir, hasta que arr[i] == arr[j]. En ese momento calcula una amplitud y si esta es mayor a la máxima, máxima toma su valor, sino, continúa con la siguiente iteración del ciclo principal. Así hasta llegar al último elemento del arreglo. Al final retorna el valor máximo.

*Ejercicio tomado de http://codingbat.com/prob/p189576

8. Complejidades de los ejercicios en CodingBat. *Calculados en el código

Array 2
countEvens → Complejidad O(n)
fizzArray2 → Complejidad O(n)
has12 → Complejidad O(n)
only14 → Complejidad O(n)
post4 → Complejidad O(n)

Array 3
canBalance → Complejidad O(n)
fix45 → Complejidad $O(n^2)$

linearIn → Complejidad O(n)
maxSpan → Complejidad O(n^2)
seriesUp → Complejidad O(n^2)

9. ¿Qué es m y qué es n en los cálculos anteriores?

Array 2

countEvens → n equivale al tamaño del arreglo que se recibe como parámetro.
fizzArray2 → n es el número recibido como parámetro.
has12 → n es el tamaño del arreglo recibido como parámetro.
only14 → n es el tamaño del arreglo recibido como parámetro.*
post4 → n es el tamaño del arreglo recibido como parámetro.*

*Aunque sea repetitivo: sí, eso es. En los 3 casos, en la peor situación, se han de recorrer todas las posiciones del arreglo.

Array 3

canBalance → n es el tamaño del arreglo recibido como parámetro. Siempre se recorre, todas las posiciones del arreglo. En el peor de los casos, sería n + n – 1, que es O(n).
fix45 → n es el tamaño del arreglo recibido como parámetro.
linearIn → Aquí habría n y m, que serían, respectivamente, el tamaño del arreglo "outer" y el tamaño del arreglo "inner". La complejidad depende únicamente de n, ya que es el mayor y, además, porque sin importar el tamaño de m, en el peor de los casos tendríamos que llegar hasta el último elemento de "outer" para saber si "inner" sí está 'dentro', en orden ascendente.
maxSpan → n es el tamaño del arreglo.
seriesUp → n es el número que se ingresa como parámetro. En realidad, el número de operaciones equivalen a las necesarias para llenar el arreglo que se retorna, que es de tamaño n*(n+1)/2, que en cálculo de complejidad equivale a O(n^2).

4) Simulacro de Parcial

1. c
2. d
3. b
4. b
5. d

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