

Laboratorio Nro. 2: NOTACION O GRANDE

Omar Alexis Becerra Sierra

Universidad Eafit
Medellín, Colombia
oabecerras@eafit.edu.co

Juan José Tamayo

Universidad Eafit
Medellín, Colombia
jjtamayoa@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

3.1) De acuerdo a lo realizado en el numeral 1.1, completen la siguiente tabla con tiempos en milisegundos, considerando la versión recursiva de los algoritmos:

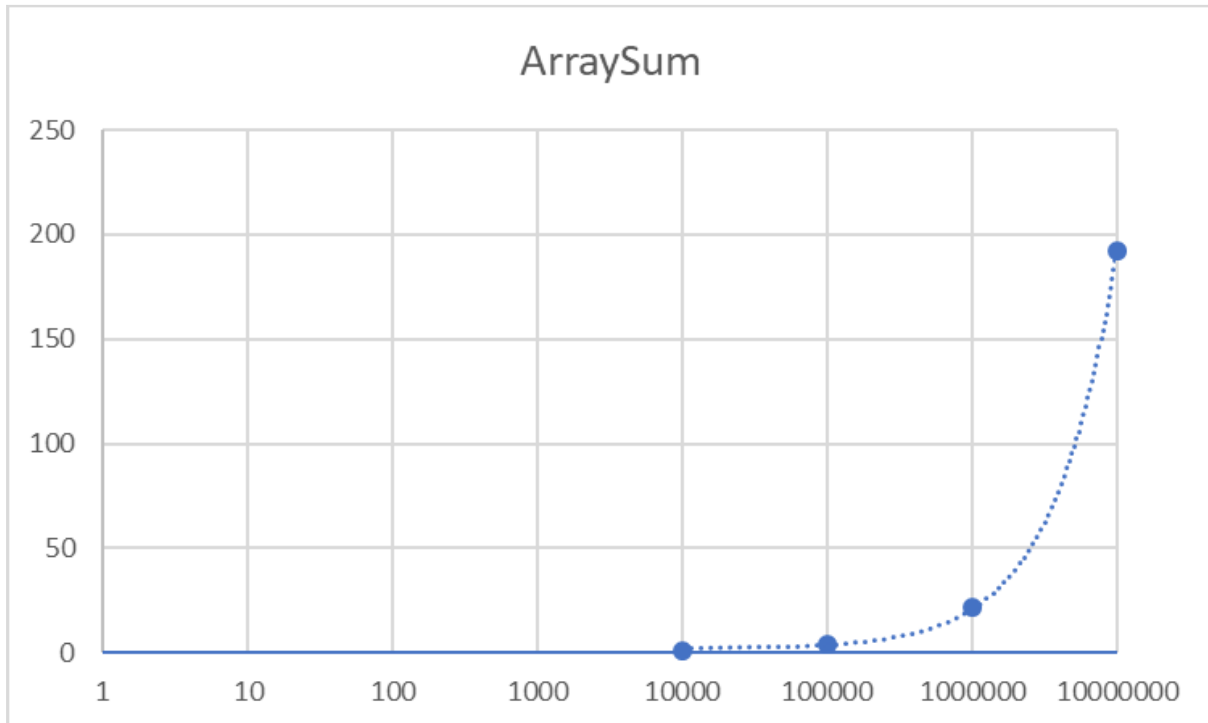
	N = 100.000	N = 1.000.000	N = 10.000.000	N = 100.000.000
R Array sum	1	4	22	192
R Array maximum	1	3	33	243
R Fibonacci	1	4	27	210

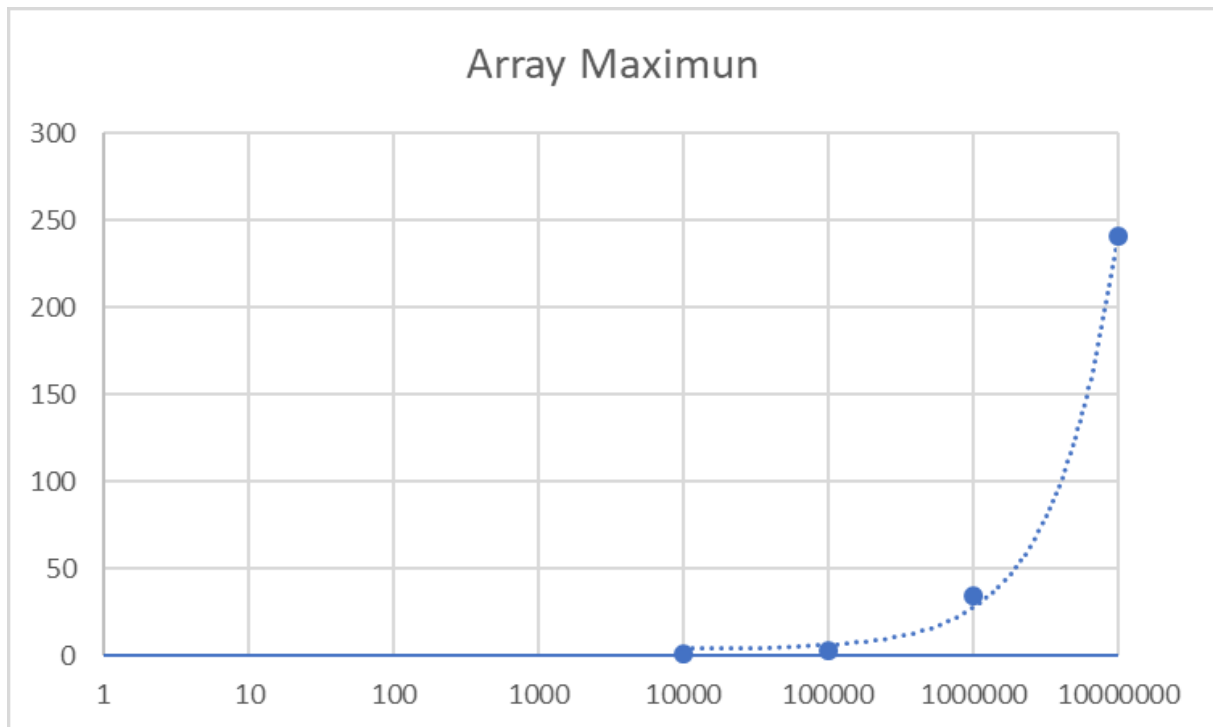
DOCENTE MAURICIO TORO BERMÚDEZ

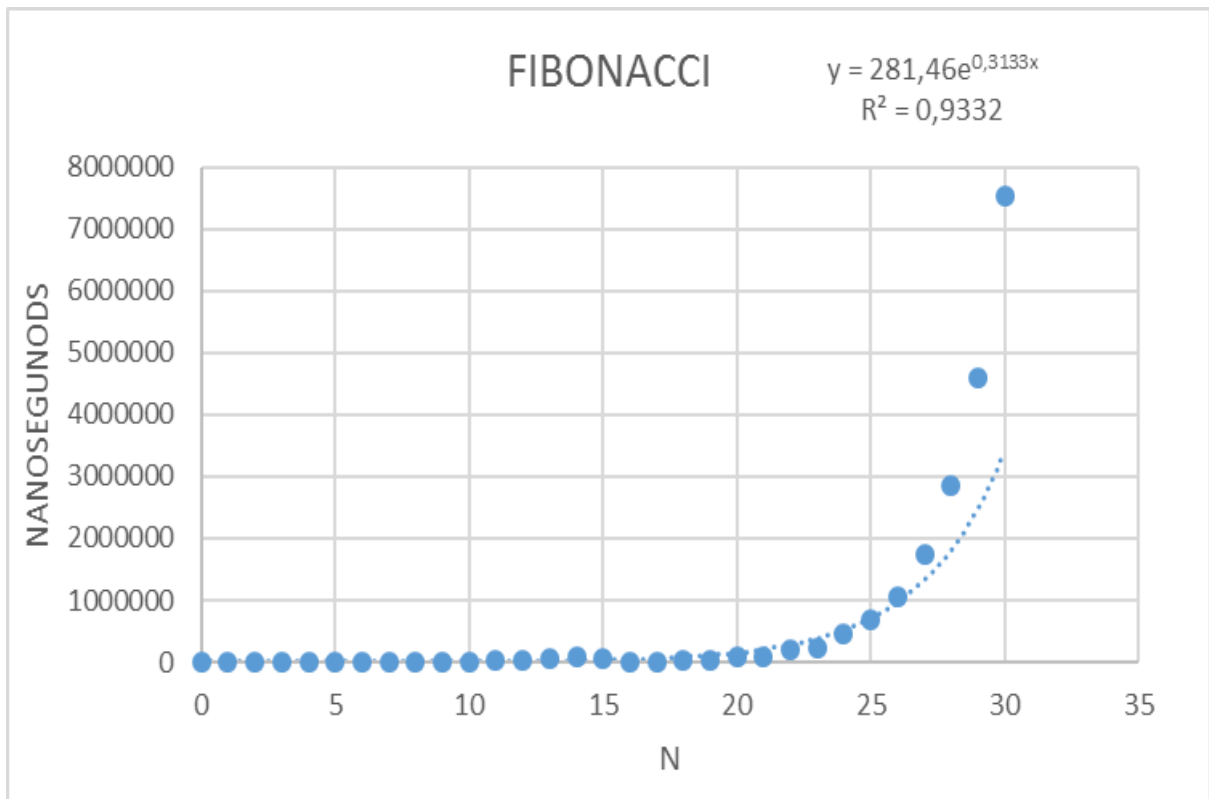
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627

Correo: mtorobe@eafit.edu.co

3.2) Grafiquen los tiempos que tomó en ejecutarse Array Sum, Array Maximum y Fibonacci recursivo, para entradas de diferentes tamaños y para la versión recursiva de los mismos. Si se demora más de un minuto la ejecución, cancela la ejecución y escriba en la tabla “más de 1 minuto”.







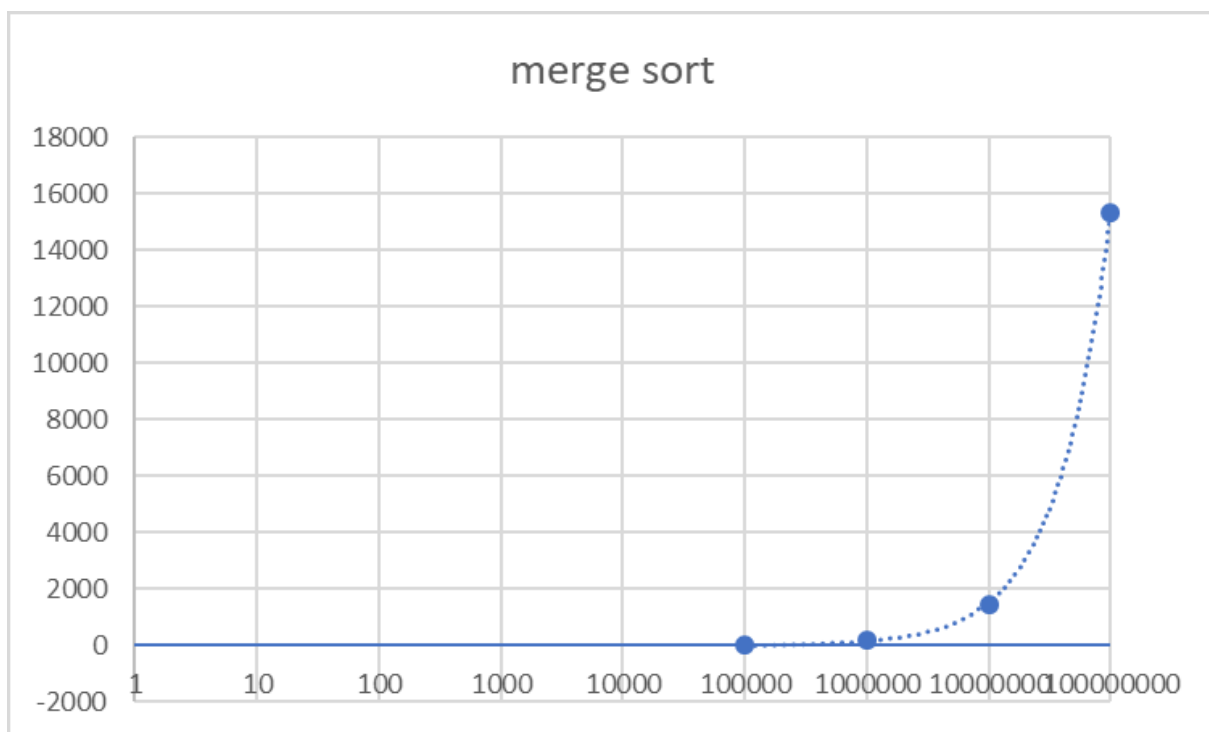
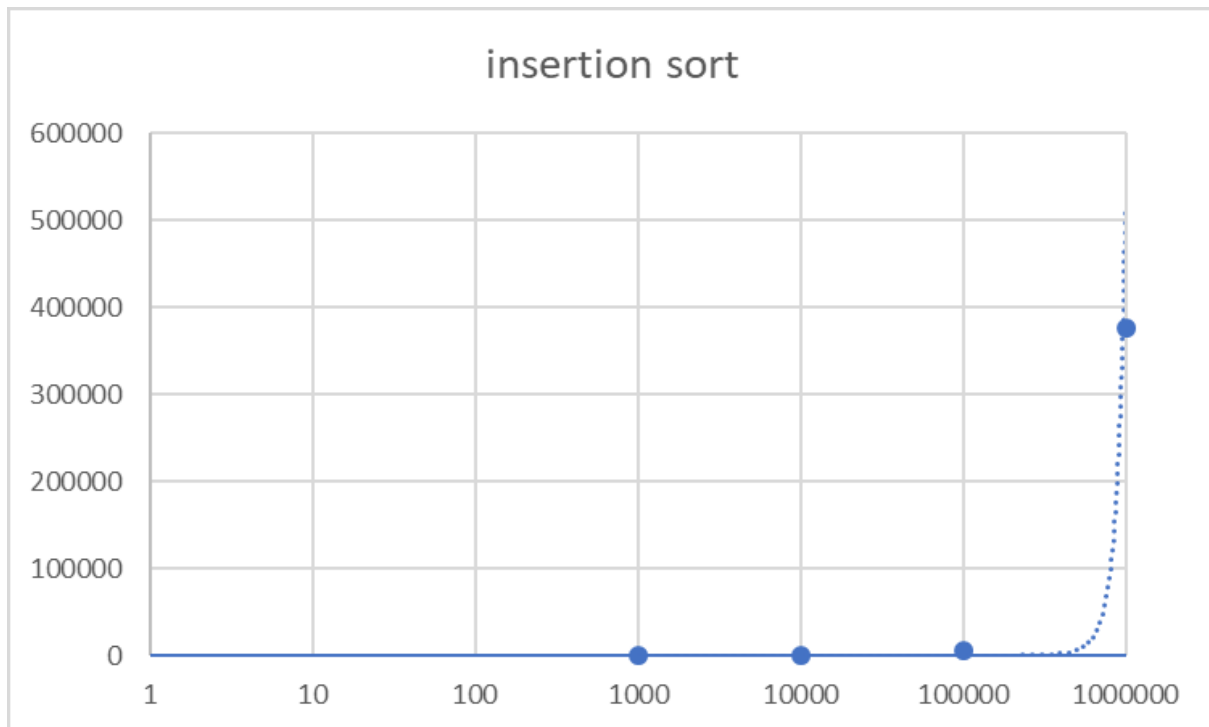
3.3 ¿Qué concluyen respecto a los tiempos obtenidos en el numeral 3,1 del presente y los resultados teóricos?

Tanto “arraysum” y “arrayMax” tenían el mismo resultado teórico el cual fue $O(n)$ y las gráficas nos ayudan a ver el resultado teórico ya que las dos dieron una línea recta además a pesar de que arrayMax daba unos valores un poco más grandes, la gráfica para valores grandes era casi la misma y se conservaba lo visto en la teoría de los que había nos estudiado

3.4 En la vida real, para analizar la eficiencia de los algoritmos de forma experimental, debemos probar con problemas de diferente tamaño. El tamaño del problema lo denominamos N. Como un ejemplo, para un algoritmo de ordenamiento, el tamaño del problema es el tamaño del arreglo.

	N = 100.000	N = 1.000.000	N = 10.000.000	N = 100.000.000
R Array sum	1	4	22	192
R Array maximum	1	3	33	243
Insert sort	4	134	5337	375609
Merge sort	1	179	1440	15313

3.5 Grafiquen los tiempos que tomó en ejecutarse array sum, arrayMax, insertion sort y merge sort, para entradas de diferentes tamaños y para la versión NO recursiva de los mismos. Si se demora más de un minuto la ejecución, cancela la ejecución y escriba en la tabla “más de 5 minutos”. Grafiquen Tamaño de la Entrada (N) Vs. Tiempo de Ejecución. Utilicen una escala logarítmica para poder graficar correctamente.



3.6 ¿Qué concluyen con respecto a los tiempos obtenidos en el numeral 3,4 del presente y los resultados teóricos obtenidos con la notación O?

Pues se puede concluir que los resultados teóricos de Insertionsort son iguales a el resultado dado en las gráficas el cual dio $O(n^2)$ y el Merge sort que dio $O(n \log n)$ son correctos y además se comportan como las funciones que deberían dar.

3.7 Teniendo en cuenta lo anterior, ¿Qué sucede con Insertion Sort para valores grandes de N?

Merge sort es mucho más eficiente para valores grandes, cuando probamos Insertion sort con 10'000.000 la ejecución de este fue muy lenta y demoro un lapso considerable, mientras que cuando probamos 100'000.000 con Mergesort tardo muy poco tiempo, en cuanto a valores pequeños Insertion sort también es más eficiente que Merge sort, pero en un lapso muy corto.

3.8 Teniendo en cuenta lo anterior, ¿Qué sucede con ArraySum para valores grandes de N? y ¿Por qué los tiempos no crecen tan rápido como Insertion Sort?

Porque Arraysum utiliza un solo ciclo para sumar los elementos mientras que insert sort tiene que usar dos ciclos entonces su complejidad seria de n^2
Mientras que la complejidad de sumar un arreglo es n , intuitivamente se puede intuir que la ejecución de insertsort es más tardía que la de ArraySum.

3.9 Teniendo en cuenta lo anterior ¿Qué tan eficiente es Merge sort con respecto a Insertion sort para arreglos grandes?

Acudiendo a la gráfica de complejidad computacional anterior y sabiendo que la complejidad de Insertion sort es n^2 y la de Merge sort es $\log(n)$ Podemos ver que en los valores grandes Insertion sort es menos eficiente que Merge sort

¿Qué tan eficiente es Merge sort con respecto a Insertion sort para arreglos pequeños?

Acudiendo a la grafica de complejidad computacional anterior y sabiendo que la complejidad de Insertionsort es n^2 y la de Mergesort es $\log(n)$ Podemos ver que en los valores pequeños Insertion sort es más eficiente que Mergesort

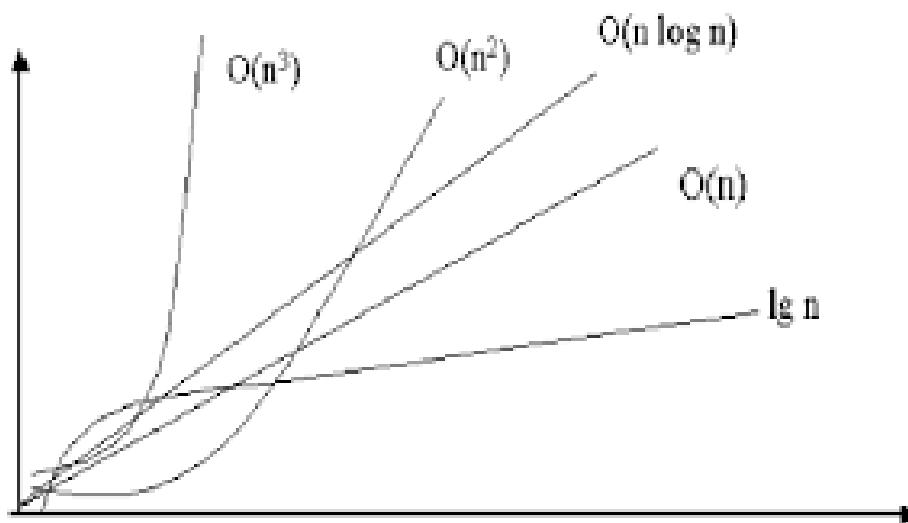



Figure 1:graphic of complied computational

	<p style="text-align: center;">UNIVERSIDAD EAFIT ESCUELA DE INGENIERÍA DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS</p>	Código: ST245
		Estructura de Datos 1

3.10 Expliquen con sus propias palabras cómo funciona el ejercicio maxSpan y ¿por qué?

Primero se mira que el tamaño del arreglo sea mayor a 0, si es cero se retorna 0, en caso de ser mayor a 0 se inicia la variable para contar el Span en 1, se empieza un ciclo desde 0 hasta el tamaño del arreglo, dentro de este ciclo hay otro ciclo que empieza desde el tamaño del arreglo -1 hasta i, si la posición i y la posición j son iguales se crea una variable igual $j - i + 1$, el +1 es porque el span es inclusivo, si la variable es mayor al Span actual se cambia el Span por la variable y se rompe el ciclo, al final se retorna el Span.

3.11 Calculen la complejidad de los ejercicios en línea, numerales 2.1 y 2.2, y agréguenla al informe PDF

ARRAY 2

```
public int countEvens(int[] nums) {  
    int count = 0; // C1  
    for (int i = 0; i < nums.length; i++) { // C2 * n  
        if (nums[i] % 2 == 0) { // C3  
            count++; // C4  
        }  
    }  
    return count; // C5  
}
```

$T(n) = C1 + C3 + C4 + C5 + C2 * n$

$T(n)$ es $O(C1 + C3 + C4 + C5 + C2 * n)$

$T(n)$ es $O(C2 * n)$

$T(n)$ es $O(n)$

```
public int bigDiff(int[] nums) {  
    int max = nums[0]; // C1  
    int min = nums[0]; // C2  
    for (int i = 0; i < nums.length; i++) { // C3 * n  
        if (nums[i] > max) { // C4  
            max = nums[i]; // C5  
        }  
        if (nums[i] <= min) { // C6  
            min = nums[i]; // C7  
        }  
    }  
    return max - min; // C8  
}
```

DOCENTE MAURICIO TORO BERMÚDEZ

Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627

Correo: mtorobe@eafit.edu.co

$T(n) = C1+C2+C4+C5+C3*n$
 $T(n)$ es $O(C1+C2+C4+C5+C3*n)$
 $T(n)$ es $O(C3*n)$
 $T(n)$ es $O(n)$

```
public int centeredAverage(int[] nums) {  
    int max = nums[0]; // C1  
    int min = nums[0]; // C2  
    int sum = 0;  
    for (int i = 0; i < nums.length; i++) { // C3 * n  
        sum += nums[i]; //C4  
        if (nums[i] > max) { // C5  
            max = nums[i]; //C6  
        }  
        if (nums[i] < min) { //C7  
            min = nums[i]; //C8  
        }  
    }  
    return (sum - (max + min)) / (nums.length - 2); //C9  
}
```

$T(n) = C1+C2+C4+C5+C6+C7+C8+C9+C3*n$
 $T(n)$ es $O(C1+C2+C4+C5+C6+C7+C8+C9+C3*n)$
 $T(n)$ es $O(C3*n)$
 $T(n)$ es $O(n)$

```
public int sum13(int[] nums) {  
    int total = 0; // C1  
    for (int i = 0; i < nums.length; i++) { // C2 * n  
        if (nums[i] != 13) { // C3  
            total += nums[i]; // C4  
        } else if (i <= nums.length - 1) { // C5  
            i++; // C6  
        }  
    }  
    return total; // C7  
}
```

$T(n) = C1 + C3 + C4 + C5 + C6 + C7 + C2 * n$

$T(n)$ es $O(C1 + C2 + C4 + C5 + C6 + C7 + C3 * n)$

$T(n)$ es $O(C3 * n)$

$T(n)$ es $O(n)$

```
public boolean lucky13(int[] nums) {  
    for (int i = 0; i < nums.length; i++) { // C1 * n  
        if (nums[i] == 1 || nums[i] == 3) { // C2  
            return false; // C3  
        }  
    }  
    return true; // C4  
}
```

$T(n) = C2 + C3 + C4 + C1 * n$

$T(n)$ es $O(C2 + C3 + C4 + C1 * n)$

$T(n)$ es $O(C1 * n)$

$T(n)$ es $O(n)$

ARRAYS 3

```
public int[] fix34(int[] nums) {  
    for (int i = 0; i < nums.length; i++){ // C1 * n  
        if (nums[i] == 3) {  
            int temp = nums[i + 1]; //C2  
            nums[i + 1] = 4; //C3  
            for (int j = i + 2; j < nums.length; j++){ // C4 * n  
                if (nums[j] == 4) { //C5  
                    nums[j] = temp; //C6  
                }  
            }  
        }  
    }  
    return nums; //C7  
}
```

$T(n) = C2 + C3 + C5 + C6 + C7 + C1 * n + C4 * n$

$T(n)$ es $O(C2 + C3 + C5 + C6 + C7 + C1 * n + C4 * n)$

$T(n)$ es $O(n * n)$

$T(n)$ es $O(n^2)$

```
public int[] fix45(int[] nums) {
    for (int i = 0; i < nums.length; i++) { // C1 * n
        if (nums[i] == 5 && i == 0 || nums[i] == 5 && nums[i - 1] != 4) { //C2
            int pos5 = i; //C3
            for (int j = 0; j < nums.length; j++) { //C4 * n
                if (nums[j] == 4 && nums[j + 1] != 5) { //C5
                    int temp = nums[j + 1]; //C6
                    nums[j + 1] = 5; //C7
                    nums[pos5] = temp; //C8
                    break;
                }
            }
        }
    }
    return nums; //C9
}
```

$T(n) = C2 + C3 + C5 + C6 + C7 + C8 + C9 + C1 * n + C4 * n$

$T(n)$ es $O(C2 + C3 + C5 + C6 + C7 + C8 + C9 + C1 * n + C4 * n)$

$T(n)$ es $O(n * n)$

$T(n)$ es $O(n^2)$

```
public boolean linearIn(int[] outer, int[] inner) {
    int indexInner = 0;
    int indexOuter = 0;
    while (indexInner < inner.length && indexOuter < outer.length) { // C1 * n
        if (outer[indexOuter] == inner[indexInner]) { //C2
            indexOuter++; // C3
            indexInner++; // C4
        } else {
            indexOuter++; //C5
        }
    }
    return (indexInner == inner.length); // C6
}
```

DOCENTE MAURICIO TORO BERMÚDEZ

Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627

Correo: mtorobe@eafit.edu.co

$T(n) = C_2 + C_3 + C_4 + C_5 + C_6 + C_1 * n$ $T(n)$ es $O(C_2 + C_3 + C_4 + C_5 + C_6 + C_1 * n)$ $T(n)$ es $O(n)$ $T(n)$ es $O(n)$

```
public int maxSpan(int[] nums) {  
    if (nums.length > 0) { //C1  
        int maxSpan = 1;  
        for (int i = 0; i < nums.length; i++) { //C2 * n  
            for (int j = nums.length - 1; j > i; j--) { //C3 * n  
                if (nums[j] == nums[i]) { //C4  
                    int count = (j - i) + 1; //C5  
                    if (count > maxSpan) { //C6  
                        maxSpan = count; //C7  
                    }  
                    break;  
                }  
            }  
        }  
        return maxSpan; //C8  
    } else {  
        return 0; //C9  
    }  
}
```

 $T(n) = C_4 + C_3 + C_5 + C_6 + C_7 + C_8 + C_9 + C_2 * n + C_3 * n$ $T(n)$ es $O(C_4 + C_3 + C_5 + C_6 + C_7 + C_8 + C_9 + C_2 * n + C_3 * n)$ $T(n)$ es $O(n * n)$ $T(n)$ es $O(n^2)$

```
public int maxSpan(int[] nums) {  
    if (nums.length > 0) { //C1  
        int maxSpan = 1;  
        for (int i = 0; i < nums.length; i++) { //C2 * n  
            for (int j = nums.length - 1; j > i; j--) { //C3 * n  
                if (nums[j] == nums[i]) { //C4  
                    int count = (j - i) + 1; //C5  
                    if (count > maxSpan) { //C6  
                        maxSpan = count; //C7  
                    }  
                    break;  
                }  
            }  
        }  
        return maxSpan; //C8  
    } else {  
        return 0; //C9  
    }  
}
```

$T(n) = C4 + C5 + C6 + C7 + C8 + C9 + C2 * n + C3 * n$

$T(n)$ es $O(C4 + C5 + C6 + C7 + C8 + C9 + C2 * n + C3 * n)$

$T(n)$ es $O(n * n)$

$T(n)$ es $O(n^2)$

3.12 Expliquen con sus palabras las variables (qué es 'n', qué es 'm', etc.) del cálculo de complejidad del numeral anterior

Las variables como n, m etc. en los cálculos de complejidad anteriores representa una variable que puede afectar el número de instrucciones que ejecuta el programa, ósea es una variable que mientras más grande sea más lento se ejecutara el programa, en todos los programas anteriores, n era el tamaño del arreglo.

1. C
2. B
3. B

4. B
5. D
6. A
- 7.

$$7.1 T(n) = C1 + T(n-1)$$

$$7.2 T(n) = O C1 + T(n-1)) == O(n)$$

5) Lecture recommended (optional)

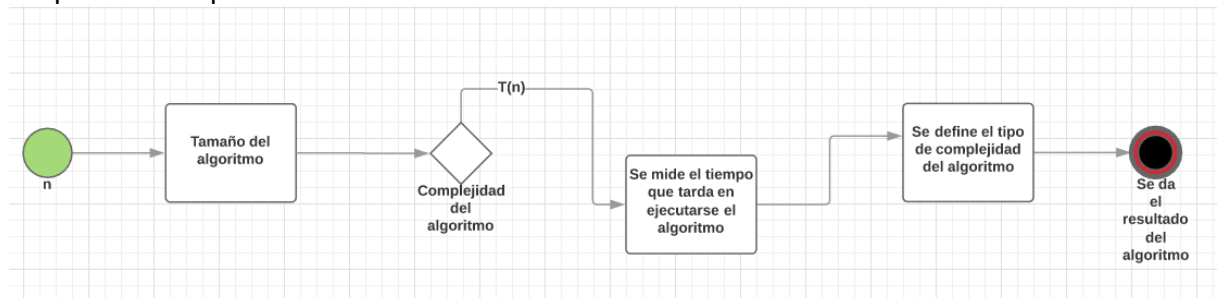
a) Título

Capítulo 3: Notación asintótica

b) Ideas principales

Se puede pensar que n representa el tamaño del ejemplar sobre el cual es preciso que se aplique un algoritmo dado, y en $t(n)$ como representante de la cantidad de un recurso dado que se invierte en ese ejemplar por una implementación particular de este algoritmo. Considérese ahora una función f : tal como $f(n) = 11$. Decimos que $t(n)$ está en el orden de $f(n)$ si $t(n)$ está acotada superiormente por un múltiplo real positivo de $f(n)$ para todo n suficientemente grande. Matemáticamente, esto significa que existe una constante real y positiva c y un entero umbral tal que $t(n) \leq cf(n)$, siempre que $n \geq n_0$. Por ejemplo, considérense las funciones $f(n)$ y $t(n)$ definidas anteriormente. $355 t(n) = 27,12$ Tomando $c = 42$ (o cualquier valor mayor) y $n_0 = 1$, concluiremos por tanto que $t(n)$ es del orden de $f(n)$ por cuanto $t(n) \leq cf(n)$ siempre que $n \geq n_0$. peor 2711 2 12 microsegundos para resolver un caso de tamaño n , podríamos simplificar diciendo que el tiempo está en el orden de n^2 . Le indicará mediante $O(f(n))$ —que se lee «O del conjunto de todas las funciones t : tales que $t(n) \leq cf(n)$ para todo $n \geq n_0$ para una constante positiva real c y para un umbral entero. En otras palabras: Aun cuando es natural utilizar el símbolo «e» de teoría de conjuntos para denotar que es del orden de tal como en « n^2 e $O(11^3)$ », le advertimos que la notación tradicional es $n^2 = O(11^3)$.

c) Mapa de conceptos



DOCENTE MAURICIO TORO BERMÚDEZ

Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627

Correo: mtorobe@eafit.edu.co