**ESTRUCTURA DE DATOS 1**
**Código ST0245**

# Laboratorio Nro. X
# Complejidad de algoritmos

**Alejandro Villada Toro**
Universidad Eafit
Medellín, Colombia
avilladat@eafit.edu.co

**Cristian Alzate Urrea**
Universidad Eafit
Medellín, Colombia
calzateu@eafit.edu.co

## 3) Simulacro de preguntas de sustentación de Proyectos

### 3.1

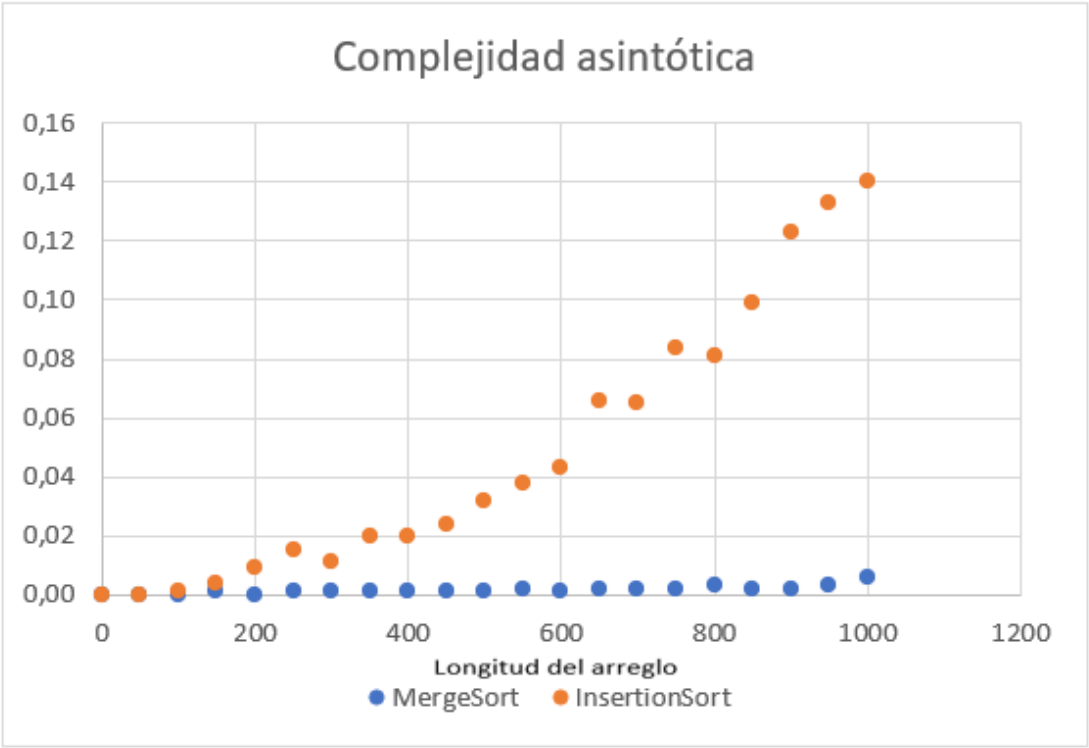
La siguiente tabla muestra los tiempos de ejecución para 20 tamaños diferentes de los algoritmos de *InsertionSort* y *MergeSort.*

| Tamaños | MergeSort | InsertionSort |
|---|---|---|
| 0 | 0,0000000000000000000 | 0,0000000000000000000 |
| 50 | 0,0000000000000000000 | 0,0000000000000000000 |
| 100 | 0,0000000000000000000 | 0,0009970664978027340 |
| 150 | 0,0009965896606445310 | 0,0039901733398437500 |
| 200 | 0,0000000000000000000 | 0,0089757442474365200 |
| 250 | 0,0010006427764892500 | 0,0149598121643066000 |
| 300 | 0,0009939670562744140 | 0,0110077857971191000 |
| 350 | 0,0009975433349609370 | 0,0199103355407714000 |
| 400 | 0,0009973049163818360 | 0,0199451446533203000 |
| 450 | 0,0009987354278564450 | 0,0239737033843994000 |
| 500 | 0,0009651184082031250 | 0,0318772792816162000 |
| 550 | 0,0020310878753662100 | 0,0379359722137451000 |
| 600 | 0,0009980201721191400 | 0,0428476333618164000 |
| 650 | 0,0019934177398681600 | 0,0658242702484130000 |
| 700 | 0,0019948482513427700 | 0,0648262500762939000 |
| 750 | 0,0019586086273193300 | 0,0837774276733398000 |
| 800 | 0,0030286312103271400 | 0,0808246135711669000 |
| 850 | 0,0019948482513427700 | 0,0987372398376464000 |
| 900 | 0,0019960403442382800 | 0,1226286888122550000 |
| 950 | 0,0029911994934082000 | 0,1326940059661860000 |
| 1000 | 0,0059840679168701100 | 0,1405749320983880000 |

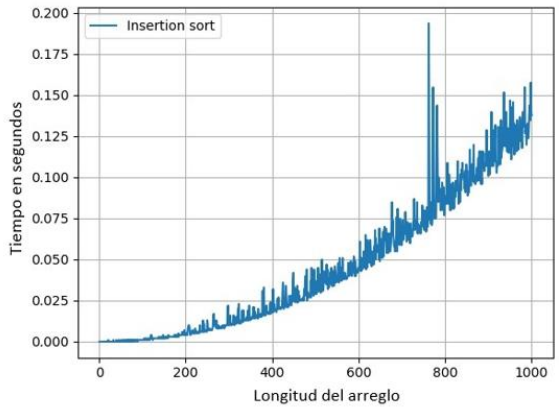
**PhD. Mauricio Toro Bermúdez**
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

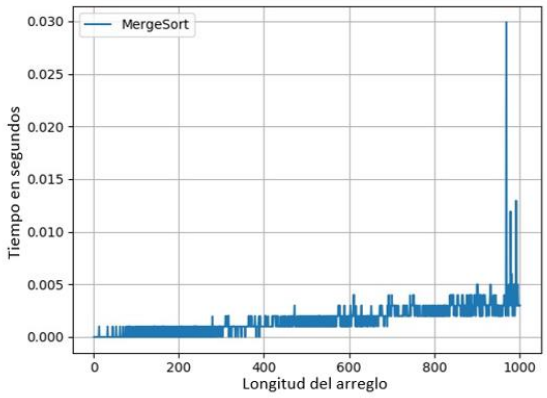
### 3.2

Si se grafica la tabla anterior se obtiene un diagrama como el siguiente:




Yendo un poco más a lo grande, y utilizando la librería *matplotlib.pyplot,* se hizo la gráfica para mil tamaños distintos de ambos algoritmos, quedando así:




Como claramente se puede ver, la gráfica de *InsertionSort* sigue la tendencia de una función polinómica de grado dos; y por otro lado, la gráfica de *MergeSort* sigue la tendencia de una función logarítmica.

**PhD. Mauricio Toro Bermúdez**
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

**3.3**

No es apropiado ya que existen métodos mucho más eficientes, como por ejemplo *mergeSort*, el cual tiene una complejidad mucho menor, comparada con la de *insertionSort.*

**3.4**

En la complejidad del algoritmo *MergeSort* aparece un logaritmo, porque en el momento de realizar el llamado recursivo, el tamaño del problema que en este caso es la longitud del arreglo, es divido entre dos. El hecho de que se vaya dividiendo es lo que hace que aparezca el logaritmo.

**3.5**

Tendría sentido que *insertionSort* fuera más rápido que *mergeSort,* cuando el arreglo tiene todos los elementos iguales.

**3.6**

El algoritmo del método *maxSpan* será explicado en punto *3.7,* donde además se presenta su complejidad.

**3.7**

**Ejercicios de Array 2 en CodingBat**

**CountEvens**

El método *CountEvens* calcula el número de enteros pares que posee un arreglo dado. Tiene un solo parámetro y es precisamente el arreglo con el que se realizará la operación. Su funcionamiento es el siguiente: Primero inicializa un entero *c* como cero, que nos servirá de contador, luego con un ciclo revisa cada elemento del arreglo, si el residuo de dividir el elemento entre dos es igual a cero, se suma uno a *c*, de no ser así simplemente pasa al siguiente elemento; finalmente retorna *c.*

La complejidad del tiempo está dada por la función T(n), donde n es la longitud del arreglo y $C_i$ son cantidades constantes de operaciones.

$C_1=1$ $C_2=5$ $C_3=1$

$T(n) = C_1 + C_2*n + C_3$

Aplicando la notación O

$T(n)$ es $O(C_1 + C_2*n + C_3)$

Aplicando reglas de suma y producto

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

**BigDiff**

El metodo *BigDiff* calcula la diferencia entre el mayor y el menor elemento de un arreglo de enteros dado. Tiene un solo parámetro y es precisamente el arreglo con el que se realizará la operación. El método tiene el siguiente funcionamiento: primero inicializa un entero *max* como el mínimo valor que pueden tomar los enteros, y un entero *min* como el máximo valor que pueden tomar los enteros; estas dos variables servirán para almacenar el mayor y el menor elemento en el arreglo. Luego con un ciclo se repasa cada elemento del arreglo, *max* se vuelve el máximo entre sí mismo

**PhD. Mauricio Toro Bermúdez**
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

y el elemento del arreglo actual, *min* se vuelve el mínimo entre sí mismo y el elemento del arreglo actual. Finalmente se retorna la diferencia entre *max* y *min.*

La complejidad del tiempo está dada por la función T(n), donde n es la longitud del arreglo y $C_i$ son cantidades constantes de operaciones.
$C_1$=4 $C_2$=1 $C_3$=2 $C_4$=2 $C_5$=2
$T(n) = C_1 + C_2*n + C_3*n + C_4*n + C_5$
Aplicando la notación O
T(n) es O ($C_1 + C_2*n + C_3*n + C_4*n + C_5$)
Aplicando reglas de suma y producto
T(n) es O (n).

**CenteredAverage**
El método *centeredAverage* calcula el promedio de los elementos de un arreglo de enteros ignorando el mayor y menor elemento. Tiene un solo parámetro y es precisamente el arreglo con el que se realizará la operación. Funciona de la siguiente manera: primero inicializa un entero *max* como el mínimo valor que pueden tomar los enteros, y un entero *min* como el máximo valor que pueden tomar los enteros; estas dos variables servirán para almacenar el mayor y el menor elemento en el arreglo. Luego con un ciclo se repasa cada elemento del arreglo, *max* se vuelve el máximo entre sí mismo y el elemento del arreglo actual, *min* se vuelve el mínimo entre sí mismo y el elemento del arreglo actual. Después se inicializa un entero *c* como la suma de *max* y *min* multiplicada por menos uno. Posteriormente, con otro ciclo, se le suman todos los elementos del arreglo a *c*. Finalmente retorna *c* dividido la longitud del arreglo menos dos.

La complejidad del tiempo está dada por la función T(n), donde n es la longitud del arreglo y $C_i$ son cantidades constantes de operaciones.
$C_1$=4 $C_2$=1 $C_3$=2 $C_4$=2 $C_5$=3 $C_6$=3 $C_7$=4
$T(n) = C_1 + C_2*n + C_3*n + C_4*n + C_5 + C_6*n + C_7$
Aplicando la notación O
T(n) es O ($C_1 + C_2*n + C_3*n + C_4*n + C_5 + C_6*n + C_7$)
Aplicando reglas de suma y producto
T(n) es O (n).

**Sum13**
El método *sum13* suma los elementos de un arreglo de enteros dado, ignorando aquellos elementos que sean igual a 13 y los inmediatamente siguientes a estos. Tiene un solo parámetro y es precisamente el arreglo con el que se realizará la operación. Su funcionamiento es el siguiente: primero inicializa un entero *c* como cero, que nos servirá para guardar los datos de la suma; luego con un ciclo se revisa cada elemento del arreglo, si el elemento es distinto de 13, lo suma a *c,* en caso contrario le suma uno al iterador del ciclo. Para terminar retorna *c.*

**PhD. Mauricio Toro Bermúdez**
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

La complejidad del tiempo está dada por la función T(n), donde n es la longitud del arreglo y $C_i$ son cantidades constantes de operaciones.
$C_1$=1 $C_2$=3 $C_3$=6 $C_4$=2 $C_5$=1
$T(n) = C_1 + C_2*n + C_3*n + C_4*n + C_5$
Aplicando la notación O
$T(n)$ es $O(C_1 + C_2*n + C_3*n + C_4*n + C_5)$
Aplicando reglas de suma y producto
$T(n)$ es $O(n)$.

**Has22**
El método *has22* nos dice si hay un dos seguido por otro dos en alguna de las posiciones de un arreglo de enteros dado. Tiene un solo parámetro y es precisamente el arreglo con el que se realizará la operación. Su funcionamiento es el siguiente: primero, por medio de un ciclo, revisa cada uno de los elementos del arreglo exceptuando el último, si el elemento en la posición actual y el elemento en la siguiente son iguales a dos retorna *true*, de lo contrario, pasa al siguiente elemento, si al llegar al penúltimo elemento sigue sin cumplirse la condición retorna *false*.

La complejidad del tiempo está dada por la función T(n), donde n es la longitud del arreglo y $C_i$ son cantidades constantes de operaciones.
$C_1$=11 $C_2$=1
$T(n) = C_1*(n-1) + C_2$
Aplicando la notación O
$T(n)$ es $O(C_1*n - C_1 + C_2)$
Aplicando reglas de suma y producto
$T(n)$ es $O(n)$.

**Ejercicios de Array3 en CodingBat**

**CountClumps**
En este método un grupo está conformado por dos o más números consecutivos. El objetivo de la función en contar cuantos grupos hay en un arreglo. Como parámetro recibe un arreglo de enteros, y a partir de allí se ejecuta. Su funcionalidad es la siguiente: primero inicializa una variable de tipo entero llamada c, la cual va a almacenar el número de grupos. Después, recorre todo el arreglo con un for, y va comprobando si el elemento está en la posición cero y el elemento que está después de él son iguales; o alternativamente, si el elemento analizado y el que está después son iguales, y si el elemento que está antes es diferente. En caso de cumplirse alguna de esas dos opciones, la variable c será aumentada en una unidad. Cuando termina el ciclo, retorna el valor de la variable, que indica el número de grupos.

**PhD. Mauricio Toro Bermúdez**
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

La complejidad en el tiempo está dada por la función T(n), donde n es el tamaño del arreglo, y Ci son las operaciones constantes.
$C_1 = 1, C_2 = 4, C_3 = 6, C_4 = 11, C_5 = 1$
$T(n) = C_1 + C_2*n + C_3*n + C_4*n + C_5$
Aplicando la notación O:
$T(n)$ es $O(C_1 + C_2*n + C_3*n + C_4*n + C_5)$
Aplicando reglas de suma y producto:
$T(n)$ es $O(n)$

**Fix34**

El algoritmo Fix34 recibe un arreglo de números, y devuelve otro arreglo con los mismos elementos, pero reorganizándolos de tal forma que cuando aparezca un 3 se va a buscar un 4 en cualquier posición del arreglo para colocarlo después del primer número. Este método recibe un solo parámetro, el cual es el arreglo para analizar. Su funcionamiento es el siguiente: mediante un ciclo for, recorre todas las posiciones de este hasta encontrar un 3; una vez localizado dicho número, se busca un 4 en el arreglo con otro for, con la condición de que no puede estar después de un tres; luego de encontrar dicho elemento, crea una variable temporal llamada temp que almacena el número después del tres; posteriormente, a la posición que está después del tres le asigna un 4, y a la posición en que estaba el cuatro le asigna el elemento que estaba después del tres, que está contenido en temp. Este procedimiento lo realiza hasta llegar al final de la matriz con el primer for.

La complejidad en el tiempo está dada por la función T(n), donde n es el tamaño del arreglo, y Ci son las operaciones constantes.
$C_1 = 2, C_2 = 3, C_3 = 2, C_4 = 7, C_5 = 3, C_6 = 4, C_7 = 2$
$T(n) = C_1 * n + C_2 * n + (C_3 * n) *n/2 + C_4 * n * n/2 + C_5 * n/2 * n/2 + C_6 * n/2 * n/2 + C_7 * n/2 * n/2$
Aplicando la notación O:
$T(n)$ es $O(C_1 * n + C_2 * n + (C_3 * n) *n/2 + C_4 * n * n/2 + C_5 * n/2 * n/2 + C_6 * n/2 * n/2 + C_7 * n/2 * n/2)$
Aplicando reglas de suma y producto:
$T(n)$ es $O(n^2)$

**Fix45**

El algoritmo Fix45 recibe un arreglo de números, y devuelve otro arreglo con los mismos elementos, pero reorganizándolos de tal forma que cuando aparezca un 4 se va a buscar un 5 en cualquier posición del arreglo para colocarlo después del primer número. En este arreglo es posible encontrar por primera vez un 5 antes que un tres. Este método recibe un solo parámetro, el cual es el arreglo para analizar. Su funcionamiento es el siguiente: mediante un ciclo for, recorre todas las posiciones de

**PhD. Mauricio Toro Bermúdez**
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

este hasta encontrar un 4; una vez localizado dicho número, se busca un 5 en el arreglo con otro for, con la condición de que no puede estar después de un cuatro o si está en la posición 0; luego de encontrar dicho elemento, crea una variable temporal llamada temp que almacena el número después del cuatro; posteriormente, a la posición que está después del cuatro le asigna un 5, y a la posición en que estaba el cinco le asigna el elemento que estaba después del cuatro, que está contenido en temp. Este procedimiento lo realiza hasta llegar al final de la matriz con el primer for.

La complejidad en el tiempo está dada por la función T(n), donde n es el tamaño del arreglo, y Ci son las operaciones constantes.
$C_1 = 2, C_2 = 3, C_3 = 2, C_4 = 9, C_5 = 3, C_6 = 4, C_7 = 2$
$T(n) = C_1 * n + C_2 * n + (C_3 * n) * n/2 + C_4 * n * n/2 + C_5 * n/2 * n/2 + C_6 * n/2 * n/2 + C_7 * n/2 * n/2$
Aplicando la notación O:
$T(n)$ es $O(C_1 * n + C_2 * n + (C_3 * n) * n/2 + C_4 * n * n/2 + C_5 * n/2 * n/2 + C_6 * n/2 * n/2 + C_7 * n/2 * n/2)$
Aplicando reglas de suma y producto:
$T(n)$ es $O(n^2)$

**LinearIn**
Este método se encarga de decir si todos los elementos de cierto arreglo están contenidos en otro arreglo. Los parámetros de la función son dos matrices de enteros, llamadas outer e inner; la primera es la matriz que se va a recorrer buscando los elementos de la segunda. El funcionamiento de este algoritmo es el siguiente: mediante un ciclo for se va a tomar cada elemento de inner y con otro ciclo igual se va a buscar en outer; para dicho procedimiento se inicializa una variable de tipo int llamada test, la cual va a ser incrementada cuando se encuentre el elemento tomado con el primer ciclo; si al final, del segundo for la variable no ha sido incrementada, retorna falso, ya que indica que ese elemento no se encuentra. Si finaliza el primer for y no se ha retornado falso, entonces se retorna verdadero, ya que se evidencia que todos los elementos de inner están contenidos en outer.

La complejidad en el tiempo está dada por la función T(m,n), donde m y n son los tamaños de los arreglos outer e inner, respectivamente, y Ci son las operaciones constantes.
$C_1 = 2, C_2 = 1, C_3 = 2, C_4 = 5, C_5 = 2, C_6 = 1$
$T(m,n) = C_1*n + (C_2)*n + (C_3*m)*n + (C_4)*m*n + (C_5)*n + C_6$
Aplicando la notación O:
$T(n)$ es $O(C_1*n + (C_2)*n + (C_3*m)*n + (C_4)*m*n + (C_5)*n + C_6)$
Aplicando reglas de suma y producto:
$T(m,n)$ es $O(m*n)$

**PhD. Mauricio Toro Bermúdez**
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

**MaxSpan**

Este algoritmo recibe un arreglo de enteros, y lo recorre elemento a elemento, buscando la aparición más lejana del número en consideración; una vez encontrado, cuenta el número de elementos que se encuentran entre ellos. Esa operación la realiza con cada número, y al final retorna la secuencia más larga. El funcionamiento está dado de la siguiente forma: primero se inicializa en cero una variable llamada m, la cual va a almacenar la secuencia más extensa. Luego, con un for toma cada número del arreglo, y con un segundo ciclo se mueve desde el final hasta el inicio de la matriz, cuando encuentra el elemento buscado, resta la posición de elemento encontrado con la del elemento inicial, y asigna este valor a una variable llamada mt; una vez finalizado el ciclo, compara si el valor de m es menor que mt, en caso de que sea verdadero, entonces le asigna el valor de mt a m. Este proceso lo realiza hasta finalizar el primer ciclo. Al final retorna el valor que se encuentra en m, que es la secuencia más larga.

La complejidad en el tiempo está dada por la función T(n), donde m es el tamaño del arreglo, y Ci son las operaciones constantes.

$C_1 = 1, C_2 = 2, C_3 = 1, C_4 = 2, C_5 = 4, C_6 = 2, C_5 = 4, C_6 = 2, C_7 = 1, C_8 = 2, C_9 = 3$

$T(n) = C_1 + C_2*n + C_3*n + (C_4*n(n-1)/2)*n + (C_5)*n(n-1)/2*n + (C_6)*n(n-1)/2*n + (C_7)*n(n-1)/2*n + C_8*n + C_9$

Aplicando la notación O:

$T(n)$ es $O(C_1 + C_2*n + C_3*n + (C_4*n(n-1)/2)*n + (C_5)*n(n-1)/2*n + (C_6)*n(n-1)/2*n + (C_7)*n(n-1)/2*n + C_8*n + C_9)$

Aplicando reglas de suma y producto:

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

**3.8**

Las variables n y m son aquellas con las que se representa y entiende el tamaño del problema, es decir, son las variables que aumentan o disminuyen la complejidad del algoritmo ya sea en tiempo o en memoria.

## *4) Simulacro de Parcial*

***4.1*** *c*
***4.2*** *d*
***4.3*** *b*
***4.4*** *b*
***4.5*** *a*
a
***4.6*** *100000 segundos*
***4.7*** 1, 2, 3 y 4

**PhD. Mauricio Toro Bermúdez**
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

***4.8*** a
***4.9*** a
**4.10** c
**4.11** c
**4.12** b
**4.13** c
**4.14** a ó c

## *5) Lectura recomendada (opcional)*

Mapa conceptual

## 6) Trabajo en Equipo y Progreso Gradual (Opcional)

***6.1*** *Actas de reunión*
***6.2*** *El reporte de cambios en el código*
***6.3*** *El reporte de cambios del informe de laboratorio*

**PhD. Mauricio Toro Bermúdez**
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473