Laboratorio Nro. 2
Complejidad

Carlos Gustavo Vélez Manco
Universidad Eafit
Medellín, Colombia
cgvelezm@eafit.edu.co

John Esteban Castro Ramírez
Universidad Eafit
Medellín, Colombia
jecastror@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

3.1

En la siguiente tabla podemos ver los tiempos que tomó el algoritmo Insertion Sort para 20 tamaños del problema diferentes.

Dimensión del problema (n)[Longitud del arreglo]	Tiempo de complejidad (ms)
1000	0.60697794
1600	1.370368958
2200	2.116821289
2800	2.551743984
3400	5.44758296
4000	6.883194923
4600	9.699492455
5200	13.01599765
5800	16.39926004
6400	19.01372576
7000	23.37230015
7600	27.86166906
8200	31.3277545
8800	34.7318573
9400	38.54732394
10000	46.09465528
10600	52.79272628
11200	57.44562721
11800	61.75443792
12400	70.48964953

Figura 1. Tabla de tiempos para el algoritmo Insertion Sort

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 en la siguiente tabla podemos ver los tiempos que tomó el algoritmo Merge Sort para 20 tamaños del problema diferentes.

Dimensión del problema (n)[Longitud del arreglo]	Tiempo de complejidad (ms)
1000	0.016887665
1600	0.001047134
2200	0.01675868
2800	0.024930477
3400	0.024612665
4000	0.040594578
4600	0.045144081
5200	0.086394072
5800	0.045660973
6400	0.069361687
7000	0.057685852
7600	0.082159281
8200	0.067579746
8800	0.067028522
9400	0.0527668
10000	0.150695801
10600	0.098007917
11200	0.054582119
11800	0.135091543
12400	0.197421789

Figura 2. Tabla de tiempos para el algoritmo MergeSort

Cabe aclarar, que ambas tomas de tiempo las hicimos con la misma longitud del arreglo y con los arreglos generados de igual manera, para así realizar una comparación más fácilmente entre la complejidad en tiempo de los dos algoritmos; además generamos el arreglo de tal forma que quedarán los valores en él de mayor a menor para que así los algoritmos tuvieran el peor caso en el que deben organizar todos los valores.

3.2

En las siguientes gráficas podemos observar los tiempos que tomó el el algoritmo Insertion Sort, según la tabla del punto anterior:

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

Figura 3. Gráfica en Excel para el algoritmo Insertion Sort

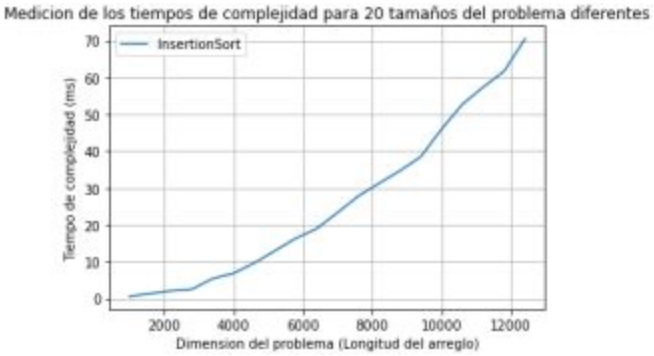


Figura 4. Gráfica en Python para el algoritmo Insertion Sort

En las siguientes gráficas podemos observar los tiempos de complejidad para el algoritmo MergeSort según los tiempos tomados de la tabla del punto anterior:

Figura 5. Gráfica en Excel para el algoritmo MergeSort

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

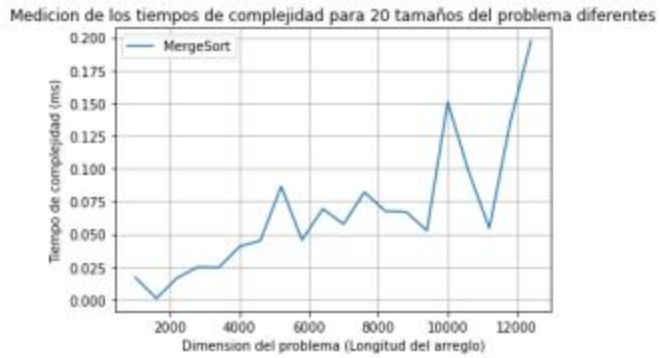


Figura 6.Gráfica en Python para el algoritmo MergeSort

Además, realizamos una gráfica comparativa entre ambos algoritmos

Figura 7. Gráfica comparativa entre los tiempos de complejidad de los algoritmos Insertion Sort y Merge Sort.

Como es evidente, el algoritmo Insertion Sort toma mucho más tiempo que Merge Sort, lo que se preveía según el cálculo de complejidad para el peor caso: Insertion Sort O(n^2) y Merge Sort O(nlogn) y claramente, nlogn es mucho más pequeño que n^2.

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, no es apropiado usar InsertionSort para un videojuego con millones de elementos en una escena y demandas de tiempo real en la renderización 3D, ya que como vemos según la gráfica del punto anterior y los cálculos de complejidad para el tiempo, el algoritmo tendrá una complejidad de O(n^2) tanto para el caso promedio como para el peor caso (el tiempo de complejidad es muy malo para grandes cantidades de datos), es decir, el algoritmo solo funciona rápidamente en el mejor de los casos que es cuando el arreglo ya está ordenado o los datos son iguales y en un videojuego esto no necesariamente sucederá, al contrario, la mayoría de veces estamos en el caso promedio o el peor de los casos y con millones de datos esto tardará demasiado tiempo, por ejemplo sólo con 12400 elementos vemos que tardó 70.5 ms según la tabla anterior, por lo que con millones de datos, que es una cantidad mucho más grande que la mencionada tardará muchísimo tiempo más y no cumplirá la demanda de renderización en tiempo real que exige el usuario en su videojuego, por lo que no es apropiado.

3.4

Aparece un logaritmo para el peor de los casos del algoritmo Merge Sort ya que este utiliza la recursión para dividir los arreglos por mitades y esta división se da en términos de tiempo para el peor de los casos es O(log(n)) hasta llegar a donde la longitud del arreglo dividido es 1 el cual es su caso de parada. Luego, para la vuelta atrás el algoritmo ordena cada subarreglo para fusionarlos a medida que va subiendo hasta el arreglo original y esto toma O(n), por lo tanto su complejidad en términos de tiempo es de O(nlog(n))

[Ejercicio opcional] (¿Cómo deben ser los datos para que Insertion Sort sea más rápido que Merge Sort?)

Para que Insertion Sort sea más rápido que Merge Sort, los datos en el arreglo deben estar ordenados o iguales, ya que este sería el mejor caso para el algoritmo y esto lo comprobamos tomando los tiempos experimentales, obteniendo las siguientes gráficas:

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

Figura 8. Gráfica para 20 tiempos experimentales de Insertion Sort y Merge Sort con datos iguales

Figura 9. Gráfica para 20 tiempos experimentales de Insertion Sort y Merge Sort con datos ordenados

Como podemos ver, en ambas gráficas Insertion Sort es más rápido que Merge Sort. Ahora, el caso promedio sería cuando los datos del arreglo son generados de manera aleatoria e igualmente realizamos la gráfica experimental obteniendo los siguientes resultados:

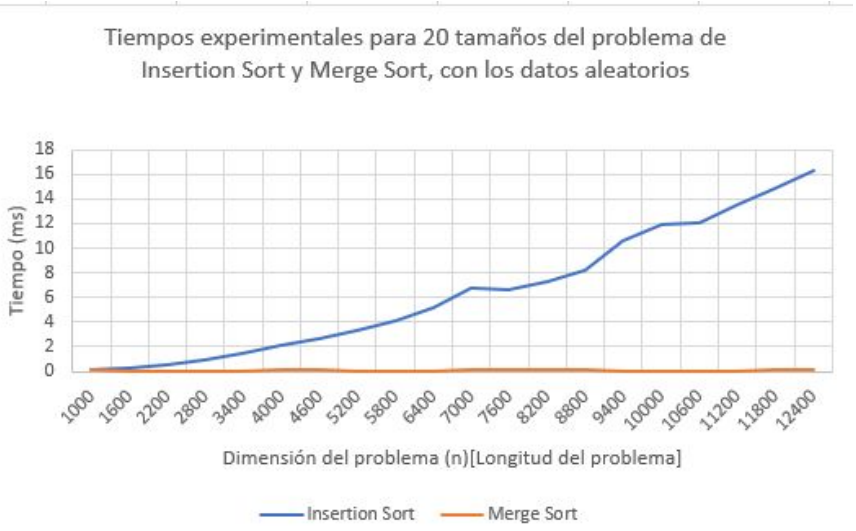


Figura 10. Gráfica para 20 tiempos experimentales de Insertion Sort y Merge Sort con datos aleatorios

Como podemos ver, insertion sort es más lento que merge sort en este caso.

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 cuando los arreglos están posicionados de mayor a menor ya pudimos ver según puntos anteriores que insertion sort nuevamente es más lento que merge sort. Por tanto, como dijimos anteriormente los datos deben ser iguales u ordenados y esto lo comprobamos también con la tabla obtenida de https://www.bigocheatsheet.com/

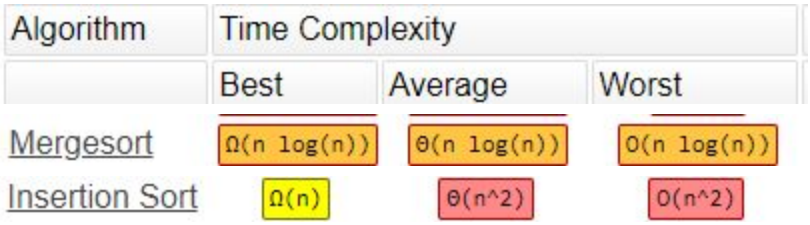


Figura 11. Complejidades de los algoritmos Merge Sort e Insertion Sort en cada escenario.

Así, podemos ver que en el mejor caso O(n) es menor que O(nlog(n)), comprobando que insertion sort es más rápido que merge sort solo en el mejor de los casos.

[Ejercicio opcional] (Explicación MaxSpan)

El algoritmo MaxSpan funciona de la siguiente manera, el arrojará el rango o intervalo más grande que se encuentre en el arreglo entre dos números iguales (tomando el valor más a la derecha y más a la izquierda del arreglo); es decir, por ejemplo si se le ingresa el arreglo [1,2,3,4,1,5,6,7,4,0] la consola nos devolverá un 6, ya que como podemos ver es la longitud del máximo intervalo que podemos encontrar entre dos números iguales, que en este caso es el 4 en la tercera posición y el 4 en la octava posición; además podemos ver que el intervalo mayor no es entre los 1s ya que este tiene una longitud de 5.

En nuestro caso, lo que hicimos para implementar el algoritmo fue primeramente crear una variable temporal y una variable que almacena este valor máximo, ambas inicializadas en 0; luego implementamos 2 ciclos que recorran el arreglo de números ingresado, en el primer ciclo lo que hacemos es evaluar el valor de cada posición comparándolo con todo el arreglo y con el segundo ciclo empezar a comparar el valor almacenado en el primer ciclo con las demás posiciones, es decir; por ejemplo si tenemos el arreglo [1,2,3,1,0] en el primer ciclo entramos con el valor de la posición 0, es decir el 1, y con el segundo ciclo empezamos a recorrer el arreglo de nuevo posición por posición encontrando los valores que también sean 1, es decir iguales al valor almacenado del arreglo en la posición i, para que entren al condicional y así cambiar los valores de las variables 'temporal' y 'máximo'; en este caso encontramos un 1 en la posición 0 y así, temporal=i-j+1=0-0+1=1 por tanto, el máximo por el momento está en 1; luego estos ciclos van a hacer lo mismo iterativamente hasta que i llegue al valor de la longitud del arreglo en cuestión, devolviendo así por consola el valor máximo que tenga almacenado, que es el valor que deseamos encontrar justamente.

A continuación se muestra un ejemplo de cómo funciona el código paso a paso para la entrada [1,2,0,1,3]

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

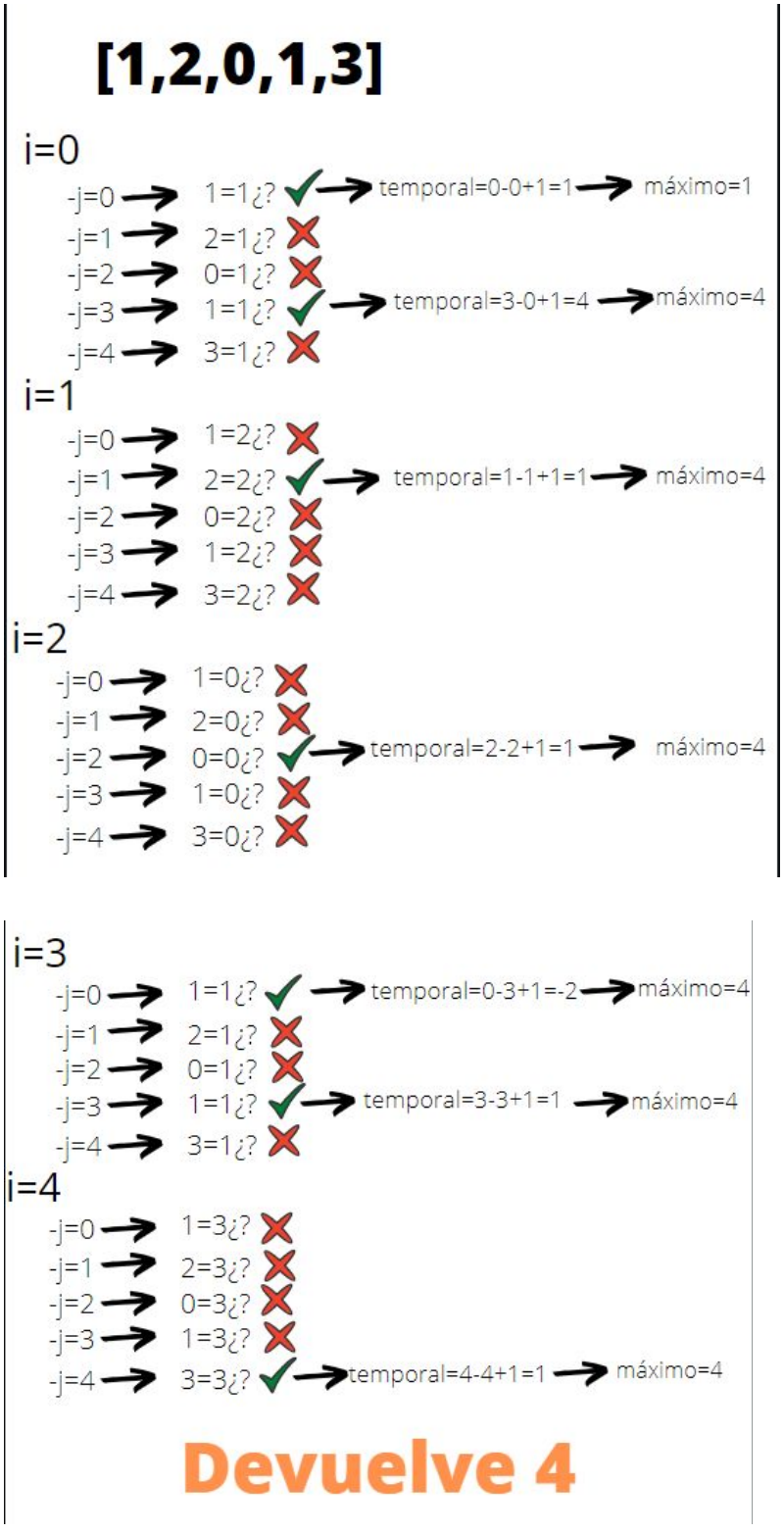


Figura 12. Paso a paso de nuestra solución para el ejercicio MaxSpan.

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

3.5

ARRAY 2.

- **CountEvents:**
  Para calcular la complejidad de este algoritmo, lo que hicimos fue calcular línea por línea la complejidad del mismo, para luego realizar una serie de operaciones aplicando algunas de las reglas de la complejidad para llegar a la misma.
  Línea 2=C1
  Línea 3=C2*n
  Línea 4=C3*n
  Línea 5=C4*n
  Línea 6=C5

  El peor caso para este algoritmo es que deba de ejecutarse desde la línea 2 hasta la línea 6, es decir que ingrese al ciclo y en ese caso la ecuación de complejidad queda:

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

  Luego, aplicamos los siguientes pasos:

  - $T(n) = [n * (C2 + C3 + C4)] + C$ **----->(Factor común y regla de la suma)**
  - $T(n) = (n * C') + C$ **------>(Regla de la suma)**
  - $T(n) = n * C'$ **------>(Regla de la suma)**
  - $T(n) = n$ **----->(Regla del producto)**
  - $T(n)\ es\ O(n)$ **------>(Notación O)**

  Donde n es la longitud del arreglo, así, podemos ver que el algoritmo es medianamente bueno para grandes dimensiones del problema.

- **FizzArray:**
  Para calcular la complejidad de este algoritmo, lo que hicimos fue calcular línea por línea la complejidad del mismo, para luego realizar una serie de operaciones aplicando algunas de las reglas de la complejidad para llegar a la misma.
  Línea 2=C1
  Línea 3=C2*n
  Línea 4=C3*n
  Línea 5=C4

  El peor caso para este algoritmo es que deba de ejecutarse desde la línea 2 hasta la línea 5, es decir que ingrese al ciclo y en ese caso la ecuación de complejidad queda:

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

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

Luego, aplicamos los siguientes pasos:

1. $T(n) = [n * (C2 + C3)] + C$ **----->(Regla de la suma y factor común)**
2. $T(n) = (n * C') + C$ **------->(Regla de la suma)**
3. $T(n) = n * C'$ **------->(Regla de la suma)**
4. $T(n) = n$ **----->(Regla del producto)**
5. $T(n)\ es\ O(n)$ **----->(Notación O)**

Donde n es la longitud del arreglo, así, podemos ver que el algoritmo es medianamente bueno para grandes dimensiones del problema.

- **Lucky13:**
Para calcular la complejidad de este algoritmo, lo que hicimos fue calcular línea por línea la complejidad del mismo, para luego realizar una serie de operaciones aplicando algunas de las reglas de la complejidad para llegar a la misma.
Línea 2=C1*n
Línea 3=C2*n
Línea 4=C3*n
Línea 5=C4

El peor caso para este algoritmo es que deba de ejecutarse desde la línea 2 hasta la línea 4, es decir que ingrese al ciclo y además, al condicional y en ese caso la ecuación de complejidad queda:

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

Luego, aplicamos los siguientes pasos:

1. $T(n) = n * (C1 + C2 + C3)$ **----->(Factor común)**
2. $T(n) = n * C$ **------>(Regla de la suma)**
3. $T(n) = n$ **------>(Regla del producto)**
4. $T(n)\ es\ O(n)$ **----->(Notación O)**

Donde n es la longitud del arreglo, así, podemos ver que el algoritmo es medianamente bueno para grandes dimensiones del problema.

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

- **Sum28:**
  Para calcular la complejidad de este algoritmo, lo que hicimos fue calcular línea por línea la complejidad del mismo, para luego realizar una serie de operaciones aplicando algunas de las reglas de la complejidad para llegar a la misma.
  Línea 2=C1
  Línea 3=C2*n
  Línea 4=C3*n
  Línea 5=C4*n
  Línea 6=C5
  Línea 7=C6
  Línea 8=C7

  El peor caso para este algoritmo es que deba de ejecutarse desde la línea 2 hasta la línea 7, es decir que ingrese al ciclo, en este caso la ecuación es:

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

  Luego, aplicamos los siguientes pasos:

  1. $T(n) = [n * (C2 + C3 + C4)] + C$ **-->(Regla de la suma y factor común)**
  2. $T(n) = (n * C') + C$ **----->(Regla de la suma)**
  3. $T(n) = n * C'$ **------>(Regla de la suma)**
  4. $T(n) = n$ **----->(Regla del producto)**
  5. $T(n)\ es\ O(n)$ **----->(Notación O)**

  Donde n es la longitud del arreglo, así, podemos ver que el algoritmo es medianamente bueno para grandes dimensiones del problema.

- **IsEveryWhere:**
  Para calcular la complejidad de este algoritmo, lo que hicimos fue calcular línea por línea la complejidad del mismo, para luego realizar una serie de operaciones aplicando algunas de las reglas de la complejidad para llegar a la misma.
  Línea 2=C1*(n-1)
  Línea 3=C2*(n-1)
  Línea 4=C3*(n-1)
  Línea 5=C4

  El peor caso para este algoritmo es que deba de ejecutarse desde la línea 2 hasta la línea 4, es decir que ingrese al ciclo y al mismo tiempo al condicional, en este caso la ecuación es:

  $$T(n) = C1 * (n-1) + C2 * (n-1) + C3 * (n-1)$$

  Luego, aplicamos los siguientes pasos:

  1. $T(n) = [(n-1) * (C1 + C2 + C3)]$ **---->(Factor común)**

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

2. $T(n) = (n-1) * C$ **------>(Regla de la suma)**
3. $T(n) = n - 1$ **----->(Regla del producto)**
4. $T(n) = n$ **----->(Regla de la suma)**
5. $T(n)\ es\ O(n)$ **----->(Notación O)**

Donde n es la longitud del arreglo, así, podemos ver que el algoritmo es medianamente bueno para grandes dimensiones del problema.

- **TripleUp:**
Para calcular la complejidad de este algoritmo, lo que hicimos fue calcular línea por línea la complejidad del mismo, para luego realizar una serie de operaciones aplicando algunas de las reglas de la complejidad para llegar a la misma.
Línea 2=C1*(n-2)
Línea 3=C2*(n-2)
Línea 4=C3*(n-2)
Línea 5=C4

El peor caso para este algoritmo es que deba de ejecutarse desde la línea 2 hasta la línea 4, es decir que ingrese al ciclo y al mismo tiempo al condicional, en este caso la ecuación es:

$$T(n) = C1 * (n-2) + C2 * (n-2) + C3 * (n-2)$$

Luego, aplicamos los siguientes pasos:

1. $T(n) = [(n-2) * (C1 + C2 + C3)]$ **---->(Factor común)**
2. $T(n) = (n-2) * C$ **---->(Regla de la suma)**
3. $T(n) = n - 2$ **--->(Regla del producto)**
4. $T(n) = n$ **---->(Regla de la suma)**
5. $T(n)\ es\ O(n)$ **---->(Notación O)**

Donde n es la longitud del arreglo, así, podemos ver que el algoritmo es medianamente bueno para grandes dimensiones del problema.

ARRAY 3.

- **Fix34:**
Para calcular la complejidad de este algoritmo, lo que hicimos fue calcular línea por línea la complejidad del mismo, para luego realizar una serie de operaciones aplicando algunas de las reglas de la complejidad para llegar a la misma.
Línea 2=C1*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

Línea 3=C2*n*n
Línea 4=C3*n*n
Línea 5=C4*n*n
Línea 6=C5*n*n
Línea 7=C6*n*n
Línea 8=C7

El peor caso para este algoritmo es que deba de ejecutarse desde la línea 2 hasta la línea 8, es decir que ingrese a los 2 ciclos y al mismo tiempo al condicional, en este caso la ecuación es:

$$T(n) = C1 * n + C2 * n^2 + C3 * n^2 + C4 * n^2 + C5 * n^2 + C6 * n^2 + C7$$

Luego, efectuamos los siguientes pasos:

1. $T(n) = (C1 * n) + [n^2 * (C2 + C3 + C4 + C5 + C6)] + C7$ **--->(Factor común)**
2. $T(n) = (C1 * n) + (C * n^2) + C7$ **--->(Regla de la suma)**
3. $T(n) = (C1 * n) + (C * n^2)$ **---->(Regla de la suma)**
4. $T(n) = n + n^2$ **---->(Regla del producto)**
5. $T(n) = n^2$ **--->(Regla de la suma, se toma el valor máximo, en este caso n^2 es mayor que n)**
6. $T(n)\ es\ O(n^2)$ **--->(Notación O)**

Donde n es la longitud del arreglo, así, podemos ver que el algoritmo es malo para grandes dimensiones del problema.

- **Fix45:**
Para calcular la complejidad de este algoritmo, lo que hicimos fue calcular línea por línea la complejidad del mismo, para luego realizar una serie de operaciones aplicando algunas de las reglas de la complejidad para llegar a la misma.
Línea 2=C1*n
Línea 3=C2*n*n
Línea 4=C3*n*n
Línea 5=C4*n*n
Línea 6=C5*n*n
Línea 7=C6*n*n
Línea 8=C7

El peor caso para este algoritmo es que deba de ejecutarse desde la línea 2 hasta la línea 8, es decir que ingrese a los 2 ciclos y al mismo tiempo al condicional, en este caso la ecuación es:

$$T(n) = C1 * n + C2 * n^2 + C3 * n^2 + C4 * n^2 + C5 * n^2 + C6 * n^2 + C7$$

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

Luego, efectuamos los siguientes pasos:

1. $T(n) = (C1 * n) + [n^2 * (C2 + C3 + C4 + C5 + C6)] + C7$ **--->(Factor común)**
2. $T(n) = (C1 * n) + (C * n^2) + C7$ **--->(Regla de la suma)**
3. $T(n) = (C1 * n) + (C * n^2)$ **---->(Regla de la suma)**
4. $T(n) = n + n^2$ **---->(Regla del producto)**
5. $T(n) = n^2|$ **--->(Regla de la suma, se toma el valor máximo, en este caso n^2 es mayor que n)**
6. $T(n)\ es\ O(n^2)$ **--->(Notación O)**

Donde n es la longitud del arreglo, así, podemos ver que el algoritmo es malo para grandes dimensiones del problema.

- **MaxSpan:**

Para calcular la complejidad de este algoritmo, lo que hicimos fue calcular línea por línea la complejidad del mismo, para luego realizar una serie de operaciones aplicando algunas de las reglas de la complejidad para llegar a la misma.
Línea 2=C1
Línea 3=C2
Línea 4=C3*n
Línea 5=C4*n*n
Línea 6=C5*n*n
Línea 7=C6*n*n
Línea 8=C7*n*n
Línea 9=C8
El peor caso para este algoritmo es que deba de ejecutarse desde la línea 2 hasta la línea 9, es decir que ingrese a los 2 ciclos y al mismo tiempo al condicional, en este caso la ecuación es:

$$T(n) = C1 + C2 + C3 * n + C4 * n^2 + C5 * n^2 + C6 * n^2 + C7 * n^2 + C8$$

Luego, efectuamos los siguientes pasos:

1. $T(n) = (C3 * n) + [n^2 * (C4 + C5 + C6 + C7)] + C$ **--->(Factor común y regla de la suma)**
2. $T(n) = (C3 * n) + (C' * n^2) + C$ **--->(Regla de la suma)**
3. $T(n) = (C3 * n) + (C' * n^2)$ **--->(Regla de la suma)**
4. $T(n) = n + n^2$ **--->(Regla del producto)**
5. $T(n) = n^2|$ **-->(Regla de la suma, se toma el valor máximo, en este caso n^2 es mayor que 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

6. $T(n)\ es\ O(n^2)$ **--->(Notación O)**

Donde n es la longitud del arreglo, así, podemos ver que el algoritmo es malo para grandes dimensiones del problema.

- **LinearIn:**
Para calcular la complejidad de este algoritmo, lo que hicimos fue calcular línea por línea la complejidad del mismo, para luego realizar una serie de operaciones aplicando algunas de las reglas de la complejidad para llegar a la misma.
Línea 2=C1
Línea 3=C2*n
Línea 4=C3*n*m
Línea 5=C4*n*m
Línea 6=C5*n*m
Línea 7=C6*n*m
Línea 8=C7
Línea 9=C8
Línea 10=C9

El peor caso para este algoritmo es que deba de ejecutarse desde la línea 2 hasta la línea 10, es decir que ingrese a los 2 ciclos y al mismo tiempo al condicional, en este caso la ecuación es:

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

Luego, efectuamos los siguientes pasos:

1. $T(n,m) = C2 * n + [(n * m)(C3 + C4 + C5 + C6)] + C$ **--->(Factor común y regla de la suma)**
2. $T(n,m) = C2 * n + [(n * m) * C'] + C$ **--->(Regla de la suma)**
3. $T(n,m) = C2 * n + [(n * m) * C']$ **--->(Regla de la suma)**
4. $T(n,m) = n + (n * m)$ **---->(Regla del producto)**
5. $T(n,m) = n * m$ **---->(Tomó el valor más grande, en este caso n*m es más grande que n)**
6. $T(n,m)\ es\ O(n * m)$ **--->(Notación O)**

Donde n es la longitud del arreglo inner y m es la longitud del arreglo outer, así podemos ver que el algoritmo es malo para grandes dimensiones del problema.

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

- **CanBalance:**
  Para calcular la complejidad de este algoritmo, lo que hicimos fue calcular línea por línea la complejidad del mismo, para luego realizar una serie de operaciones aplicando algunas de las reglas de la complejidad para llegar a la misma.
  Línea 2=C1
  Línea 3=C2*n
  Línea 4=C3*n
  Línea 5=C4*n
  Línea 6=C5*n*(n-i-1)
  Línea 7=C6*n*(n-i-1
  Línea 8=C7*n
  Línea 9=C8*n
  Línea 10=C9

  El peor caso para este algoritmo es que deba de ejecutarse desde la línea 2 hasta la línea 10, es decir que ingrese a los 2 ciclos y al mismo tiempo al condicional, en este caso la ecuación es:

  $$T(n) = C1 + C2 * n + C3 * n + C4 * n + C5 * n * (n - i - 1) + C6 * n * (n - i - 1) + C7 * n + C8 * n + C9$$

  Luego, efectuamos los siguientes pasos:

  1. $T(n) = n * (C2 + C3 + C4 + C7 + C8) + [(n * (n - i - 1)) * (C5 + C6)] + C$ **-->(Factor común y regla de la suma)**
  2. $T(n) = n * C'' + (C' * (n * (n - i - 1))) + C$ **--->(Regla de la suma)**
  3. $T(n) = (C'' * n) + (C' * n * (n - i - 1))$ **-->(Regla de la suma)**
  4. $T(n) = n + (n * (n - i - 1))$ **-->(Regla del producto)**
  5. $T(n) = n + n^2 - ni - n$ **--->(Multiplicación)**
  6. $T(n) = n^2 - ni$ **--->(Resta, cancelo n)**
  7. $T(n) = n^2 - n$ **--->(Regla del producto)**
  8. $T(n) = n^2$ **--->(El más grande es n^2, regla de la suma)**
  9. $T(n)\ es\ O(n^2)$ **-->(Notación O)**

  Donde n es la longitud del arreglo, así, podemos ver que el algoritmo es malo para grandes dimensiones del problema.

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

3.6

ARRAY 2.

- **CountEvents**: ‘n’ es la longitud del arreglo
- **FizzArray**: ‘n’ es la longitud del arreglo
- **Lucky13**: ‘n’ es la longitud del arreglo
- **Sum28**:’n’ es la longitud del arreglo
- **IsEveryWhere**: ‘n’ es la longitud del arreglo
- **TripleUp**: ‘n’ es la longitud del arreglo

ARRAY 3.

- **Fix34**: ‘n’ es la longitud del arreglo.
- **Fix45**: ‘n’ es la longitud del arreglo.
- **MaxSpan**: ‘n’ es la longitud del arreglo
- **LinearIn**: ‘n’ es la longitud del arreglo inner y ‘m’ es la longitud del arreglo outer.
- **CanBalance**: ‘n’ es la longitud del arreglo.

4) Simulacro de Parcial

4.1 *10 segundos*
4.2 *D*
4.3 *A*
4.4
1.O(mxn)
2.O(mxn)
4.5
1.D
2.A
4.6 *A*
4.7 *Todas las proposiciones son verdaderas*
4.8 *A*
4.9 *C*
4.10 *C*
4.11 *C*
4.12 *A*

6) Trabajo en Equipo y Progreso Gradual (Opcional)

6.1 Actas de reunión

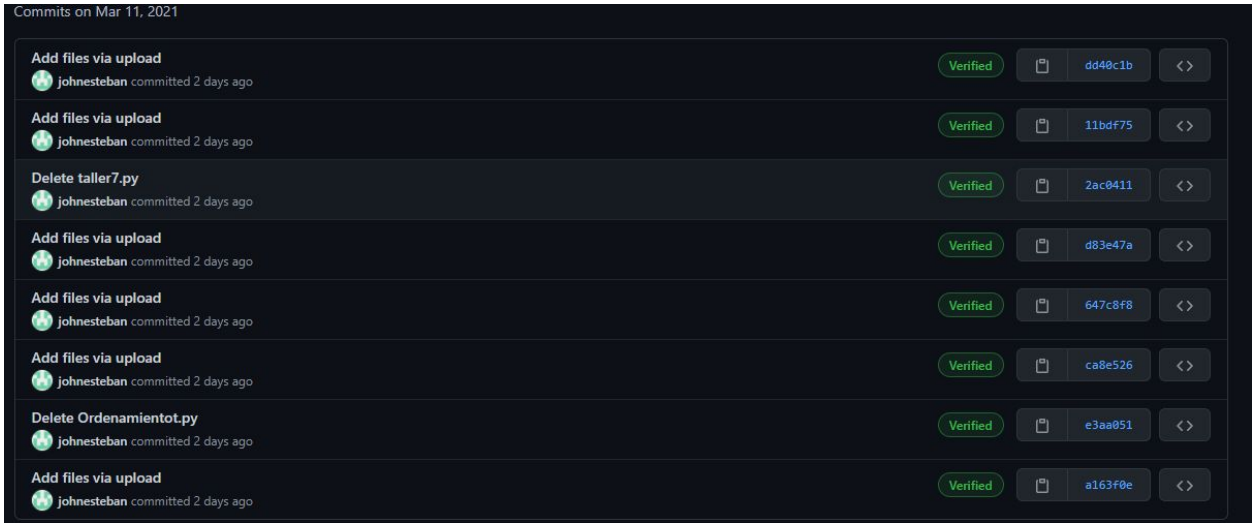
Miembros	Fecha	Por hacer	Haciendo	Hecho

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

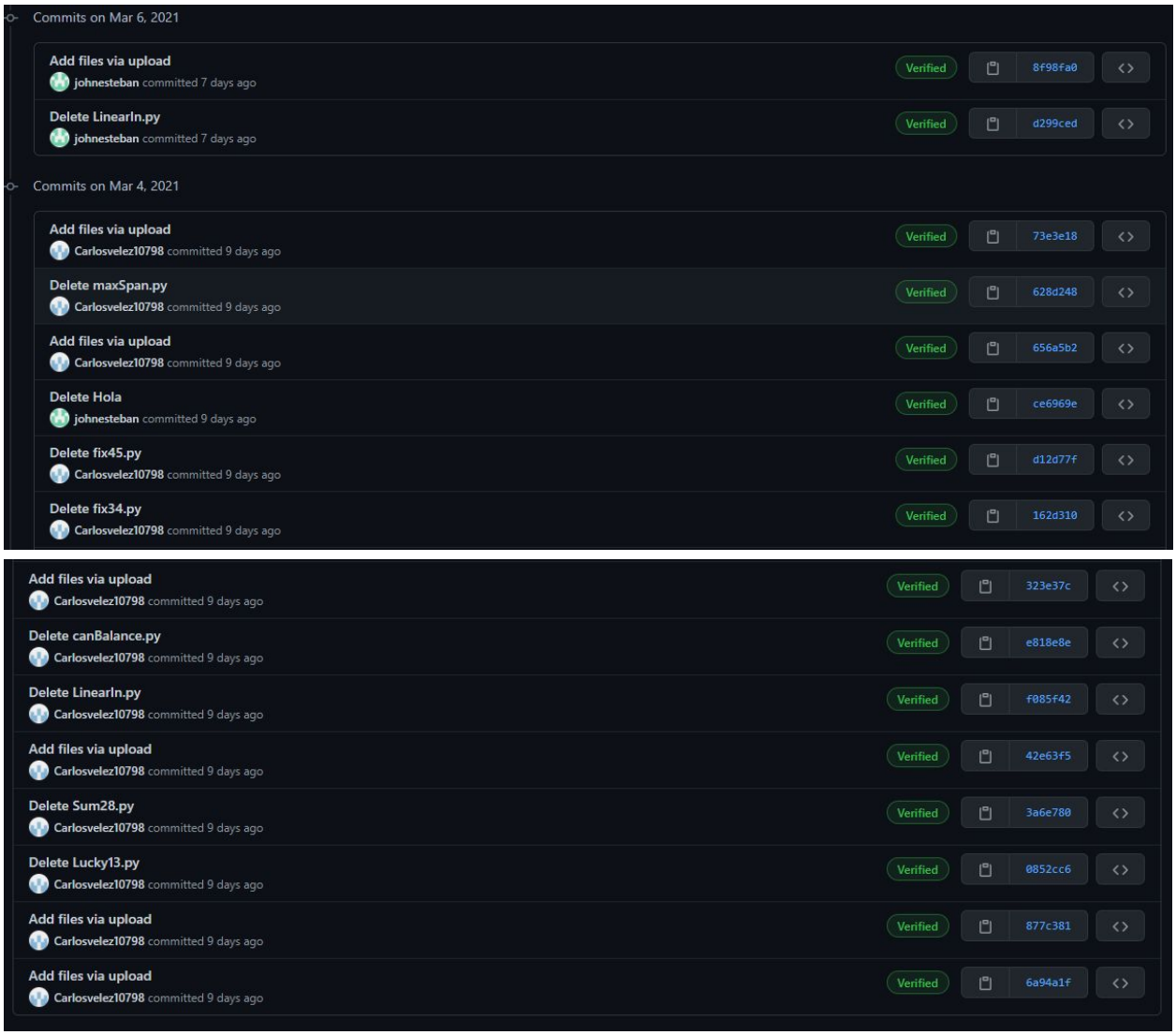
John Esteban y Carlos Gustavo	03/03/2021	Cálculo de complejidades, algoritmo Merge Sort	Realización ejercicios Coding Bat e Insertion Sort	Lectura de ejercicios
John Esteban y Carlos Gustavo	04/03/2021	Merge Sort, toma de tiempos Ejercicio 1.1, informe	Cálculo de complejidades de ejercicios de CodingBat	Ejercicios CodingBat con sus respectivas complejidades.
John Esteban y Carlos Gustavo	*08/03/2021*	*Cálculo de complejidad de MergeSort, punto 3.3 y 3.4*	*Simulacro Parcial*	*Simulacro Parcial, ejercicios CodingBat y complejidades*
John Esteban y Carlos Gustavo	*13/03/2021*		*Detalles finales del laboratorio*	

***6.2** El reporte de cambios en el código*

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

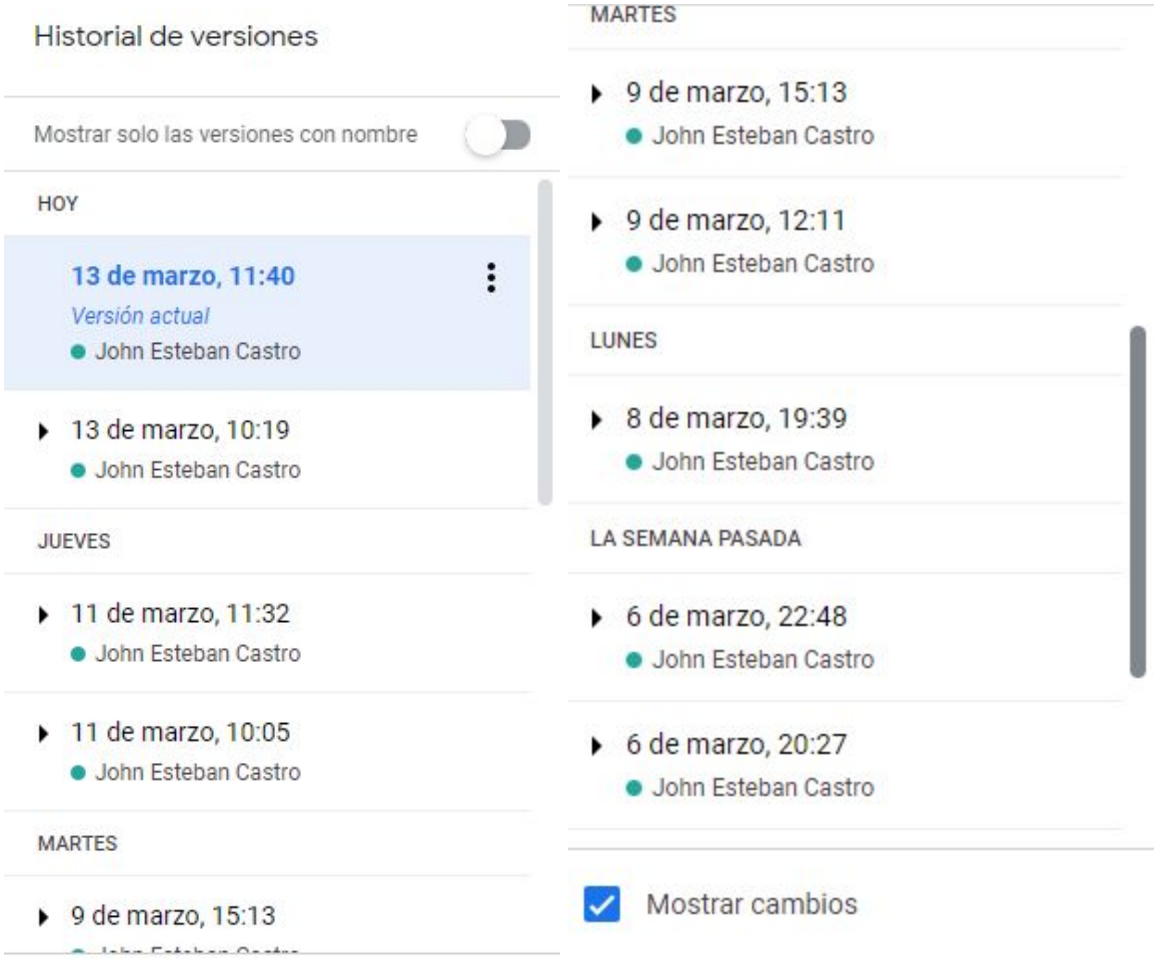


PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473



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



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