ESTRUCTURA DE DATOS 1
Código ST0245

Laboratorio Nro. 3
Listas enlazadas y vectores dinámicos.

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

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

3) Simulación de preguntas de sustentación de proyectos.

3.1. CÁLCULO DE COMPLEJIDADES DE LOS EJERCICIOS 1.X

- Ejercicio 1.1. Mapa de Medellín:
  Para este ejercicio hicimos varias opciones de algoritmos que se encuentran en el repositorio de GitHub, calculamos su complejidad y encontramos la siguiente ecuación de recurrencia para el peor de los casos:

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

  Y luego ejecutamos los siguientes pasos:

  1. **Notación O:**
     $$T(n,m)\ es\ O(C2 + C2 + C3 * n + C4 * n + C5 * m + C6 * m + C7 * m + C8 + C9)$$
  2. **Regla de la suma y factor común:**
     $$O(C2 + C2 + C3 * n + C4 * n + C5 * m + C6 * m + C7 * m + C8 + C9) = O(\big(n * (C3 + C4)\big) + \big(m * (C5 + C6 + C7)\big) + C')$$
  3. **Regla de la suma:**
     $$O\Big(\big(n * (C3 + C4)\big) + \big(m * (C5 + C6 + C7)\big) + C'\Big) = O((n * C) + (m * C))$$
  4. **Regla del producto:**
     $$O\big((n * C) + (m * C)\big) = O(n + m)$$
  5. **Complejidad:**
     $$T(n,m) es\ O(n + m)$$

  En donde n es la cantidad de vértices y m es la cantidad de arcos.

- Ejercicio 1.3. Pivote.
  Para este ejercicio empleamos un vector dinámico para así almacenar los diferentes números para luego proceder con el cálculo de la posición efectiva del pivote. En cuanto a su complejidad para el peor de los casos encontramos la siguiente ecuación de recurrencia:

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

$$T(n) = C1 + C2 + C3 + C4 + C5 * n + C6 * n + C7 * n + C8 + C9 * (n-1) + C10 * (n-1) + C11 * (n-1) + C12 * (n-1) + C13 * (n-1) + C14 * (n-1) + C15 * (n-1) + C16$$

Y luego ejecutamos los siguientes pasos:

Notación O:

1. $$T(n) es\ O(C1 + C2 + C3 + C4 + C5 * n + C6 * n + C7 * n + C8 + C9 * (n-1) + C10 * (n-1) + C11 * (n-1) + C12 * (n-1) + C13 * (n-1) + C14 * (n-1) + C15 * (n-1) + C16)$$

Factor común y regla de la suma:

2. $$O(C1 + C2 + C3 + C4 + C5 * n + C6 * n + C7 * n + C8 + C9 * (n-1) + C10 * (n-1) + C11 * (n-1) + C12 * (n-1) + C13 * (n-1) + C14 * (n-1) + C15 * (n-1) + C16) = O\big((n * (C5 + C6 + C7)) + ((n-1) * (C9 + C10 + C11 + C12 + C13 + C14 + C15)) + C'\big)$$

Regla de la suma:

3. $$O\big((n * (C5 + C6 + C7)) + ((n-1) * (C9 + C10 + C11 + C12 + C13 + C14 + C15)) + C'\big) = O((n * C) + ((n-1) * C))$$

Regla del producto:

4. $$O\big((n * C) + ((n-1) * C)\big) = O(n + (n-1))$$

Regla de la suma:

5. $$O(n + (n-1)) = O(n)$$

Complejidad:

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

Así, la complejidad de este algoritmo para el peor de los casos es O(n) en donde n es la longitud del vector dinámico o arreglo que se le ingresa al algoritmo.

- Ejercicio 1.4. Neveras:

Para este ejercicio utilizamos una pila para representar las neveras en el almacén y una cola para representar las solicitudes de las diferentes tiendas, calculando la complejidad del método asignar solicitudes para el peor de los casos encontramos la siguiente ecuación de recurrencia:

$$T(n,m) = C1 + C2 + C3 + C4 + C5 + C6 * n + C7 * n + C8 * n + C9 * (n * m) + C10 * (n * m) + C11 * (n * m) + C12 * (n * m) + C13 * n + C14 * n + C15$$

Y luego ejecutamos los siguientes pasos:

Notación O:

1. $$T(n,m) es\ O(C1 + C2 + C3 + C4 + C5 + C6 * n + C7 * n + C8 * n + C9 * (n * m) + C10 * (n * m) + C11 * (n * m) + C12 * (n * m) + C13 * n + C14 * n + C15)$$

Regla de la suma y factor común:

2. $$O(C1 + C2 + C3 + C4 + C5 + C6 * n + C7 * n + C8 * n + C9 * (n * m) + C10 * (n * m) + C11 * (n * m) + C12 * (n * m) + C13 * n + C14 * n + C15) = O\big((n * (C6 + C7 + C8 + C13)) + ((n * m)(C9 + C10 + C11 + C12)) + C'\big)$$

Regla de la suma:

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. $$O\Big(\big(n*(C6+C7+C8+C13)\big)+\big((n*m)(C9+C10+C11+C12)\big)+C'\Big) = O((n*C')+((n*m)*C))$$

Regla del producto:

4. $$O\Big((n*C')+\big((n*m)*C\big)\Big) = O(n+(n*m))$$

Regla de la suma(tomó el valor máximo)

5. $$O\big(n+(n*m)\big) = O(n*m)$$

Complejidad:

6. $T(n,m)\ es\ O(n*m)$

Así, la complejidad de este algoritmo para el peor de los casos es O(n*m) en donde n es el total de tiendas que están solicitando neveras y m es el total de neveras que están disponibles en el almacén.

.

- Ejercicio 1.5. Listas doblemente enlazadas.
  La complejidad del método insertar un elemento en la lista doblemente enlazada en el peor de los casos, es decir, cuando no se inserta en la cabeza o cola de la lista es O(n), donde n es la longitud de la lista.
  La complejidad del método borrar un elemento en la lista doblemente enlazada en el peor de los casos, es decir, cuando no se borra la cabeza o la cola de la lista es de O(n), donde n es la longitud de la lista.
  La complejidad del método para verificar si la lista doblemente enlazada contiene algún dato para el peor de los casos es O(n), donde n es la longitud de la lista.

- Ejercicio 1.6. Simulación cajeros.
  Para este ejercicio implementamos una cola para representar cada una de las 4 filas en los cajeros y al final implementamos un tipo de cola final que reunía las 4 filas en el orden en que iban a ser atendidos según las condiciones que se planteaban en el problema. Encontramos la siguiente ecuación de recurrencia de nuestro algoritmo para el peor de los casos:

$$T(n,m) = C1+C2*n+C3*n+C4*n+C5*n+C6*n+C7*n+C8*n+C9*n+C10*n+C11 + C12+C13+C14*(m-1)+C15*(m-1)+C16*(m-1)+C17+C18+C19$$

Y luego ejecutamos los siguientes pasos:

Notación O:

1. $$T(n,m)\ es\ O(C1+C2*n+C3*n+C4*n+C5*n+C6*n+C7*n+C8*n+C9*n+C10*n+C11+C12+C13 + C14*(m-1)+C15*(m-1)+C16*(m-1)+C17+C18+C19)$$

Factor común y regla de la suma:

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. $$O(C1 + C2 * n + C3 * n + C4 * n + C5 * n + C6 * n + C7 * n + C8 * n + C9 * n + C10 * n + C11 + C12 + C13 + C14 * (m-1) + C15 * (m-1) + C16 * (m-1) + C17 + C18 + C19) = O((n * (C2 + C3 + C4 + C5 + C6 + C7 + C8 + C9 + C10)) + ((m-1)(C14 + C15 + C16)) + C')$$

   Regla de la suma:

3. $$O\left((n * (C2 + C3 + C4 + C5 + C6 + C7 + C8 + C9 + C10)) + ((m-1)(C14 + C15 + C16)) + C'\right) = O((n * C) + ((m-1) * C)$$

   Regla del producto:

4. $$O((n * C) + ((m-1) * C) = O(n + (m-1))$$

   Regla de la suma:

5. $$O(n + (m-1)) = O(n + m)$$

   Regla de la suma, se toma el valor máximo y en este caso sé que m es mayor o igual a n:

6. $$O(n + m) = O(m)$$

   Complejidad:

7. $T(n, m)$ es $O(m)$

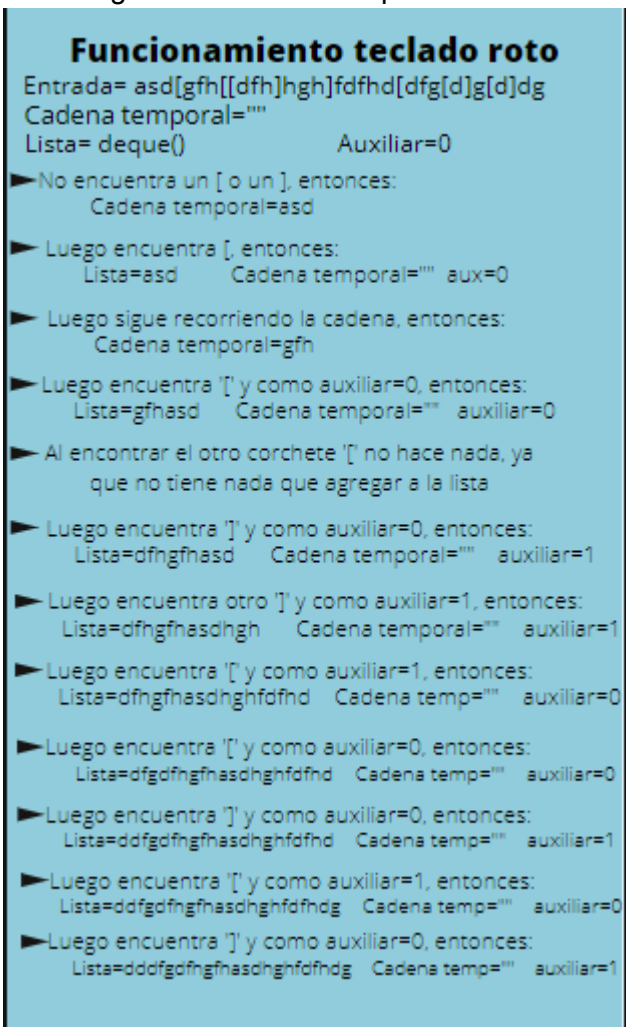
Así, la complejidad de este algoritmo para el peor de los casos es O(m) en donde m es el total de personas que están en el cajero, es decir la suma de personas entre las 4 filas; n era la fila con mayor número de personas entre las 4 existentes y por eso podía asegurar que m siempre era mayor o igual a n.

3.2 [Ejercicio opcional] Explicación de cómo funciona la implementación de los algoritmos de los ejercicios 2.x.

- **Ejercicio 2.1. Teclado roto.**
  El funcionamiento de nuestro algoritmo para darle solución al problema del teclado roto es el siguiente, lo primero que hacemos es un ciclo while que se va a ejecutar por cada carácter de la cadena de texto ingresada, si encuentra un carácter que no es un '[' o un ']' qué son las teclas de inicio y fin, lo que hace es ir almacenando esos caracteres en una variable temporal; luego si encuentra un '[' va a agregar lo que tiene almacenado en la variable temporal ya sea al inicio o al final de la lista, todo depende de si la variable auxiliar es igual a 0 o a 1, si es igual a 0 lo agrega al inicio y si es igual a 1 al final y declara a la variable auxiliar igual a 0 y la variable temporal igual a una cadena vacía, luego si encuentra un ']' pasa lo mismo, si auxiliar es igual a 0 agrega la variable temporal al inicio de la lista y si es igual a 1 al final y declara la variable auxiliar igual a 1. Ya luego de terminar toda la ejecución del ciclo, nos faltaría agregar la última parte del texto, es decir, la parte que está luego del último corchete en la cadena; entonces, si la variable auxiliar es igual a 0 va a agregar esta cadena restante al inicio de la lista y en caso contrario lo agrega al final.

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

Ya tenemos la lista completa, ahora para imprimirla por consola correctamente lo que hacemos es un ciclo que recorra la lista y vaya removiendo el elemento al inicio de la lista y lo almacene en un string acumulativo, por último retornamos ese string con el resultado esperado.



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

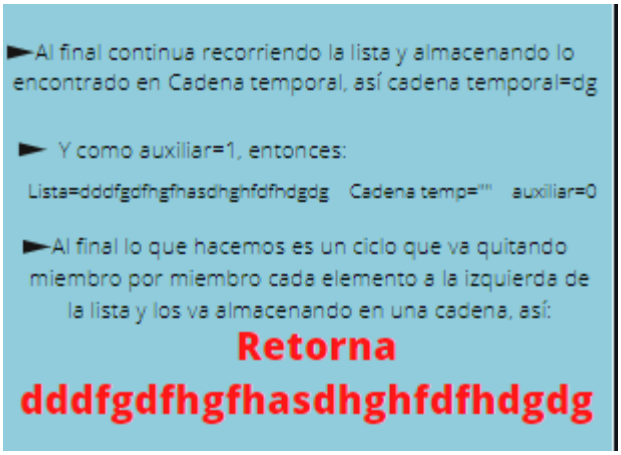


Figura 1. Funcionamiento de teclado roto con una entrada específica

- **Ejercicio 2.3. Ilya and Queries.**
  El problema nos pedía que ingresada una cadena de texto que solo constaba de '#' y '.' y un número de consultas, en cada consulta se ingresaban dos números que nos determinan un intervalo en la cadena de caracteres; entonces debíamos de retornar la cantidad de caracteres iguales consecutivos que habían desde la posición indicada por el primer entero de la consulta hasta una posición antes del segundo; es decir, si se ingresaba por ejemplo '#..###' y en una consulta se ingresaba el 1 y el 5, la posición 1 es '.' y lo siguiente hasta la posición 4 es '.##' entonces contando la posición 1 que es '.' sabemos que no es igual al caracter en la posición 0 que es un '#' luego el de la posición 2 es un '.' y es igual al anterior entonces llevamos por así decirlo en un contador imaginario un 1, luego el carácter en la posición 3 no es igual al caracter en la posición 2 entonces el contador sigue en 1 y por último la posición 4 si es igual a la anterior entonces el contador es igual a 2 y esto es lo que retorna el algoritmo.

  Para implementarlo lo que hicimos fue crear una cola que nos indicará con enteros si un carácter era consecutivo a otro o no, entonces en la primera posición de esta cola agregamos un 0, ya que la primera posición de la cadena nunca será consecutiva a otra porque no tiene un antecesor; y luego implementamos un ciclo que recorriera desde la posición 1 de la cadena hasta el final de la misma y si la posición que estábamos evaluando era igual a la anterior, es decir, un carácter igual consecutivo entonces sumará 1 a una variable contador y eso lo iba agregando a la cola y en caso de que esto no se cumpliera sumaba un 0 al contador e igualmente lo agregaba a la cola; ya teníamos la cola lista, e hicimos otro ciclo que se ejecutará el número de veces que nos indicaba las consultas ingresadas por consola y dentro del mismo se hiciera una copia de la cola para no perderla en la primera consulta, luego se piden los 2 enteros para evaluar lo mencionado anteriormente y mientras se llegará a la posición del entero 1, que nos indica desde donde empezamos íbamos removiendo los elementos al inicio de la cola y así al final de este while obtenemos el primer número deseado; luego para encontrar la segunda posición

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

que nos indicaba el entero 2 haciamos lo mismo obteniendo el segundo número deseado y al final simplemente restamos esos enteros obteniendo el resultado esperado. La cola nos funcionaba como algo auxiliar para delimitar esa 'consecutividad' y hacer más sencilla la operación.

ACLARAMOS que encontramos otra manera de solucionar el algoritmo visto de una manera más sencilla pero sin usar ninguna estructura de datos, en la cual lo que hacíamos era un ciclo que se recorriera desde la posición indicada por el entero 1 hasta la posición 2 y si dicha posición era igual a la anterior iba sumando un 1 al contador y en caso contrario no se sumaba nada y al final solo se retornaba el contador.

- **Ejercicio 2.4. Balanced brackets.**
  Este ejercicio lo que nos pedía era verificar si una serie de paréntesis, corchetes y llaves estaba bien balanceada, es decir que todos se cerrarán y lo hicieran en el orden correcto según la apertura de los mismos.

  Para realizarlo hicimos uso de una pila y un ciclo que recorriera cada carácter de la cadena de texto ingresada, pasamos a verificar si dicho carácter era un símbolo de apertura, es decir '{', '[' o '(' y si lo era, lo agregamos a la pila y si el símbolo era uno de cierre lo que hacíamos era remover un elemento de la pila, entonces si el carácter que estábamos evaluando era igual a ')' y el que removió de la pila era un '(' o las demás combinaciones de símbolos iguales no había ningún problema y el algoritmo podía seguir su recorrido, en caso contrario nos iba a retornar un "NO" indicando que estaba mal balanceada; luego al terminar el ciclo y pasar todas estas pruebas si quedaba algún carácter en la pila, es decir, algún símbolo no se había cerrado nunca entonces iba a retornar igualmente "NO" y en caso contrario de que no se quedará en ninguna de estas condiciones anteriores retornaría "YES" indicando que la cadena ingresada estaba balanceada.

3.3. Complejidad de los ejercicios 2.x

- Ejercicio 2.1. Teclado roto.
  Para este ejercicio encontramos la siguiente ecuación de recurrencia para el peor de los casos:

$$T(n,m) = C1 + C2 + C3 + C4 + C5 + C6 * n + C7 * n + C8 * n + C9 * n + C10 * n + C11 * n + C12 * n + C13 + C14 + C15 + C16 * m + C17 * m + C18$$

  Luego, aplicamos los siguientes pasos:

  Notación O:

  1. $T(n,m)\ es\ O(C1 + C2 + C3 + C4 + C5 + C6 * n + C7 * n + C8 * n + C9 * n + C10 * n + C11 * n + C12 * n + C13 + C14 + C15 + C16 * m + C17 * m + C18)$

  Regla de la suma y factor común:

  2. $Lo\ anterior\ es\ igual\ a\ O((n * (C6 + C7 + C8 + C9 + C10 + C11 + C12)) + (m * (C16 + C17)) + C')$

  Regla de la suma:

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. $Lo\ anterior\ es\ igual\ a\ O((n*C)+(m*C))$

Regla del producto:

4. $O\big((n*C)+(m*C)\big)=O(n+m)$

Regla de la suma:

5. $O(n+m)=O(n)$

Complejidad:

6. $T(n,m) es\ O(n)$

Así, la complejidad de este algoritmo para el peor de los casos es O(n) donde n es la longitud de la cadena de texto ingresada por consola, en el paso #5 se sabía que n era mayor o igual a m ya que m es la longitud de la cadena de texto ingresada sin las teclas de inicio y fin, es decir, sin '[' y ']'

- Ejercicio 2.3. Ilya and queries.

Para este ejercicio encontramos la siguiente ecuación de recurrencia para el peor de los casos:

$$T(n,m)=C1+C2+C3+C4+C5+C6+C7*(n-1)+C8*(n-1)+C9*(n-1)+C10*(n-1)+C11+C12*m+C13*m+C14*m+C15*m+C16*m+C17*m+C18*(m*n)+C19*(m*n)+C20*(m*n)+C21*(m*n)+C22*(m*n)+C23*(m*n)+C24*m+C25*m+C26*m+C27$$

Luego, aplicamos los siguientes pasos:

Notación O:

1. $T(n,m) es\ O(C1+C2+C3+C4+C5+C6+C7*(n-1)+C8*(n-1)+C9*(n-1)+C10*(n-1)+C11+C12*m+C13*m+C14*m+C15*m+C16*m+C17*m+C18*(m*n)+C19*(m*n)+C20*(m*n)+C21*(m*n)+C22*(m*n)+C23*(m*n)+C24*m+C25*m+C26*m+C27)$

Factor común y regla de la suma:

2. $Lo\ anterior\ es\ igual\ a\ O(\big((n-1)*(C7+C8+C9+C10)\big)+\big(m*(C12+C13+C14+C15+C16+C17+C24+C25+C26)\big)+\big((m*n)*(C18+C19+C20+C21+C22+C23)\big)+C')$

Regla de la suma:

3. $Lo\ anterior\ es\ igual\ a\ O(\big((n-1)*C\big)+(m*C)+\big((m*n)*C\big))$

Regla del producto:

4. $O\left(\big((n-1)*C\big)+(m*C)+\big((m*n)*C\big)\right)=O\big((n-1)+m+(m*n)\big)$

Regla de la suma:

5. $O\big((n-1)+m+(m*n)\big)=O(m*n)$

Complejidad:

6. $T(n,m) es\ O(m*n)$

Así, la complejidad de este algoritmo para el peor de los casos es O(m*n) en donde n es la longitud de la cadena de caracteres ingresada y m es el número de consultas a realizar; en el paso número 5 se sabía que m*n era el valor

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

máximo entre los 3 sumandos ya que el problema especificaba que m siempre sería mayor o igual a 1 y n siempre sería mayor o igual a 2.

- Ejercicio 2.4. Balanced brackets.
  En este ejercicio encontramos la siguiente ecuación de recurrencia para el peor de los casos:

$$T(n) = C1 + C2 * n + C3 * n + C4 * n + C5 * n + C6 * n + C7 * n + C8 * n + C9 + C10$$

Luego, aplicamos los siguientes pasos:

Notación O:

1. $T(n) es\ O(C1 + C2 * n + C3 * n + C4 * n + C5 * n + C6 * n + C7 * n + C8 * n + C9 + C10)$

Factor común y regla de la suma:

2. $$O(C1 + C2 * n + C3 * n + C4 * n + C5 * n + C6 * n + C7 * n + C8 * n + C9 + C10) = O\left(\left(n * (C3 + C4 + C5 + C6 + C7 + C8)\right) + C'\right)$$

Regla de la suma:

3. $O\left(\left(n * (C3 + C4 + C5 + C6 + C7 + C8)\right) + C'\right) = O(n * C)$

Regla del producto:

4. $O(n * C) = O(n)$

Complejidad:

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

Así, la complejidad para el peor de los casos es O(n), en donde n es la longitud del texto o cantidad de caracteres ingresados.

3.4. Variables del cálculo de complejidad del punto 3.3.

- Ejercicio 2.1. Teclado roto: 'n' es la longitud de la cadena de texto ingresada por consola y 'm' es la longitud de dicha cadena suprimiendo las teclas de inicio y fin, es decir, sin '[' y ']'.
- Ejercicio 2.3. Ilya and queries: 'n' es la longitud de la cadena de texto ingresada por consola y 'm' es el número de consultas a realizar.
- Ejercicio 2.4. Balanced brackets: 'n' es la longitud de la cadena de texto ingresada.

4) Simulacro de Parcial

4.1.

1.Línea 4: res=res+(int(vector[i])*(2**((len(vector)-1)-i)))
2. Complejidad: O(n)

4.2. C

4.3.

1.IV
2.I

4.4.

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

1.Línea 21: output.append(token).append(‘ ‘);
2.C

4.5.A
4.6.A
4.7.III
4.8.D
4.9.

1.A
2.C
3.C

4.10.

1.D
2.A
3.B

4.11.

1.C
2.B

4.12.

1.Línea 13: !(s1.isEmpty())
2.Línea 14: s1.pop()
3.Línea 17: s2.pop()

4.13.

1.I
2.III

6) Trabajo en Equipo y Progreso Gradual (Opcional)

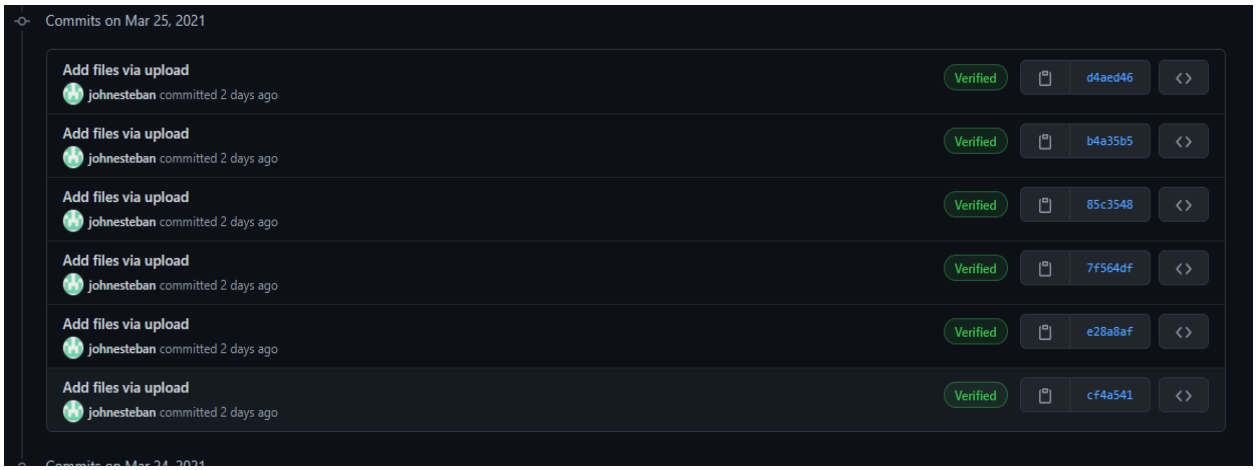
6.1. Actas de reunión

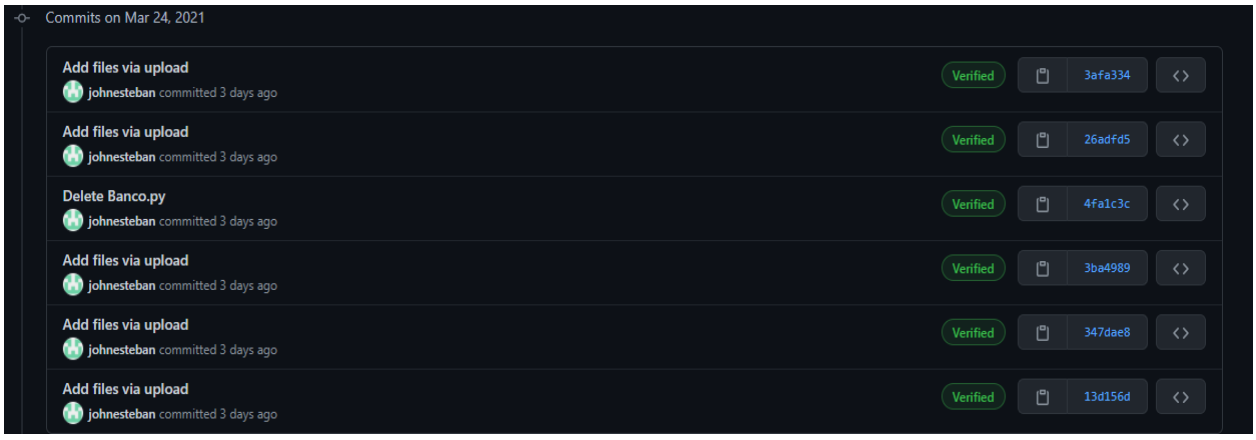
Miembros	Fecha	Por hacer	Haciendo	Hecho
John Esteban y Carlos Gustavo	18/03/2021	Códigos	Simulacro parcial	Lectura de ejercicios
John Esteban y Carlos Gustavo	21/03/2021	Ejercicio 1.1 a 1.3 y 2.1 a 2.4	Ejercicio 1.4, 1.5 y 1.6	Simulacro parcial
John Esteban y Carlos Gustavo	24/03/2021	Toma de complejidades e informe	Ejercicio 1.3 y 2.1	Ejercicios 1.4 a 1.6 y simulacro parcial
John Esteban y Carlos Gustavo	25/03/2021	Informe y terminar de calcular	Ejercicio 1.1, 2.3 y 2.4 y cálculo de algunas	Ejercicios 1.3 a 1.6 y 2.1 y toma de algunas

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

		complejidades	complejidades	complejidades

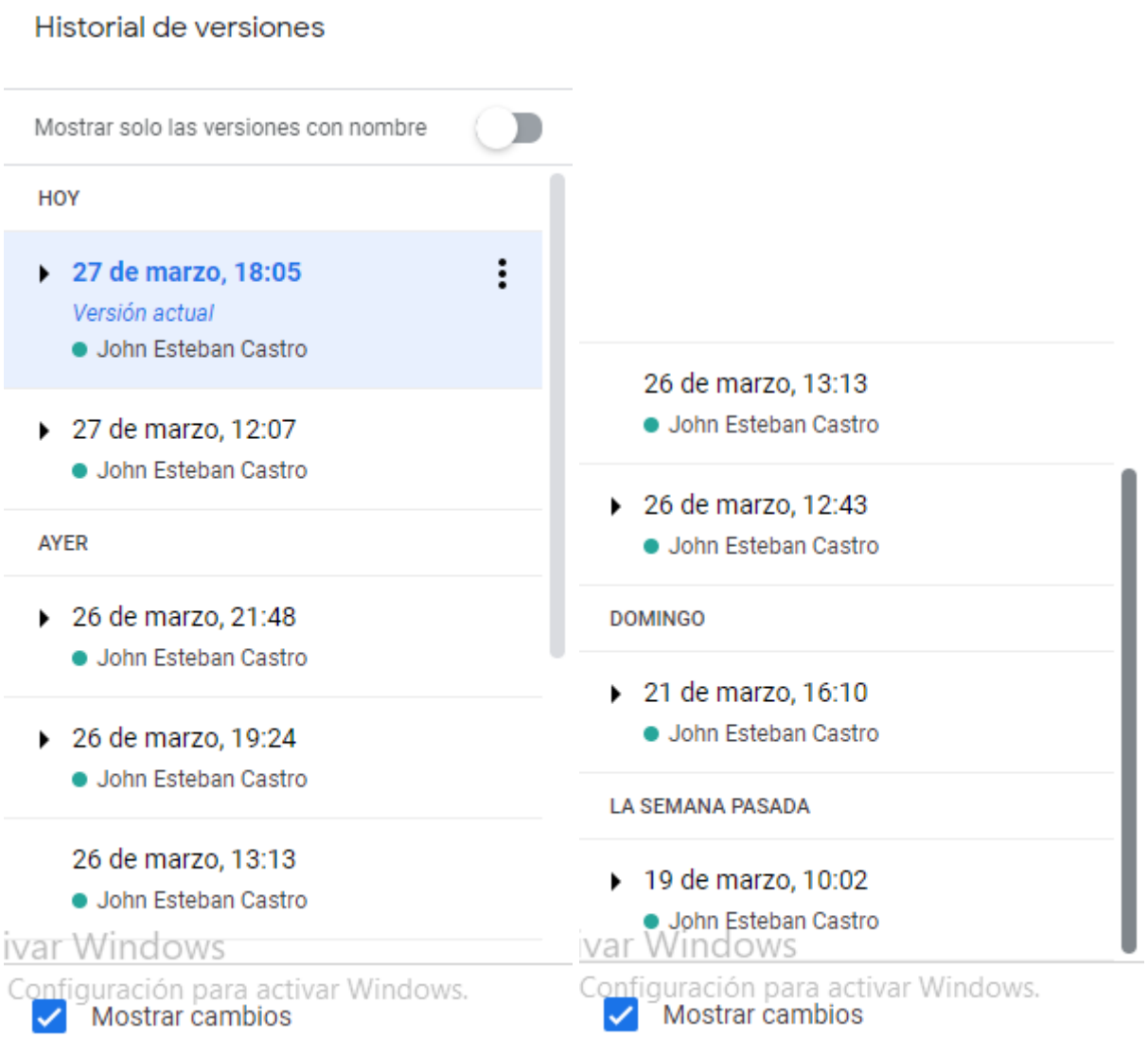
6.2 El reporte de cambios en el código





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

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