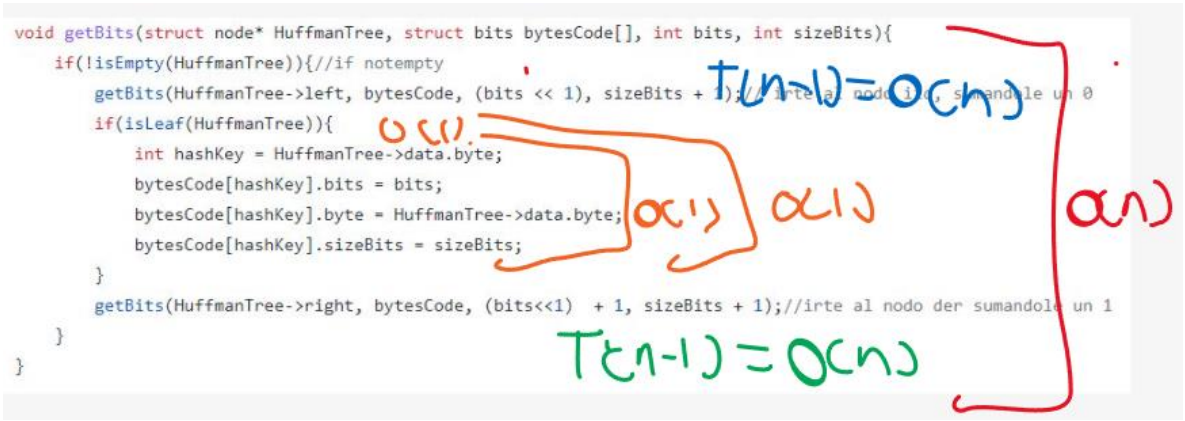
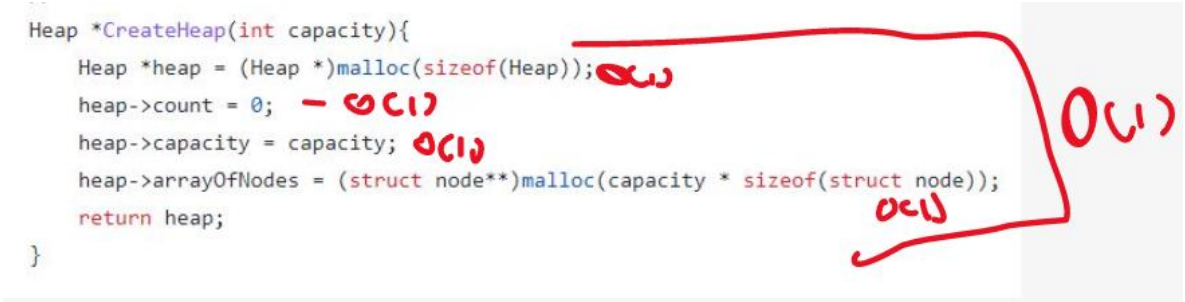
Complejidad de las Funciones HummanStructures.

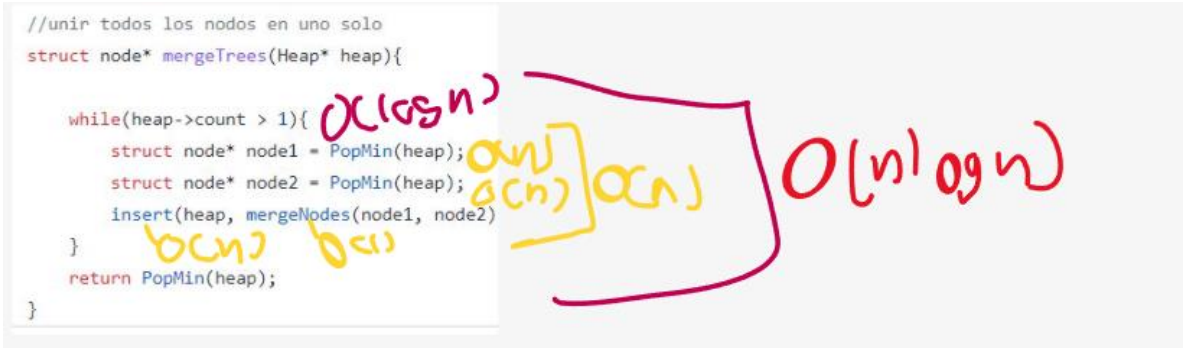
Función getbits O(n)



Nota: hace una recursión T(n-1) con cota de complejidad de O(n) y por jerarquía queda **O(n)Función CreateHeap O(1)**



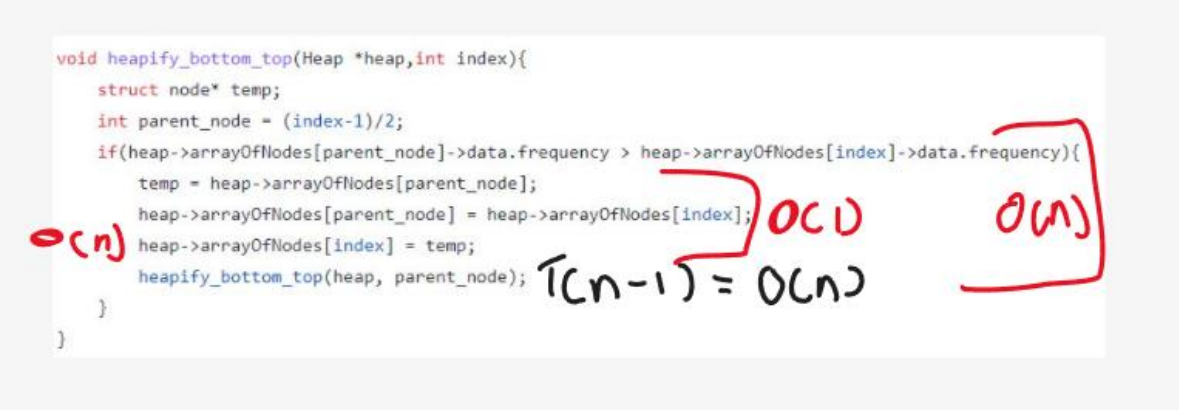
Funcion mergeTrees O(nlog n)



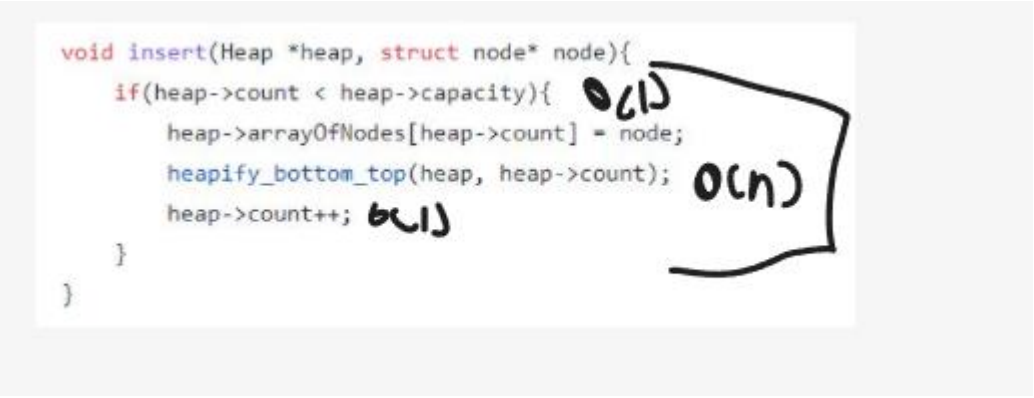
Nota se utiliza el valor de la complejidad insert y dentro de insert se llama a la funcion mergeNodes con pmplejidad de O(1) ya que es llamada durante la funcion mergeTrees al gual que PopMin es llamada dos veces por ciclo y tiene una complejidad O(n) .

Complejidad de los programas Compress y Descompress

Funcion heapy_bottom_Top O(n)

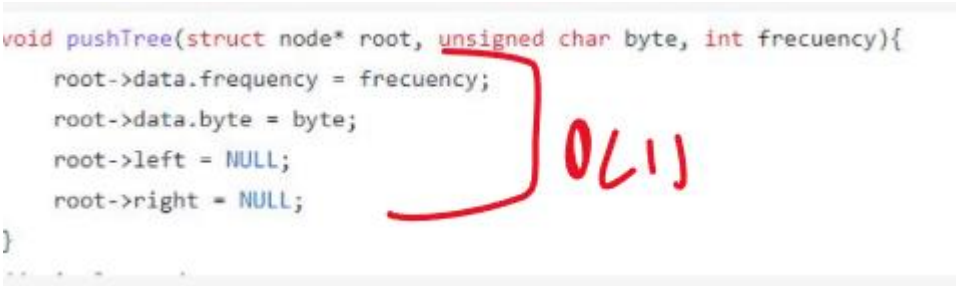


Función insert O(n)



Nota: se utilizo la complejidad de heapy_bottom_top ya que es usada en la función

Función PushTree O(1)

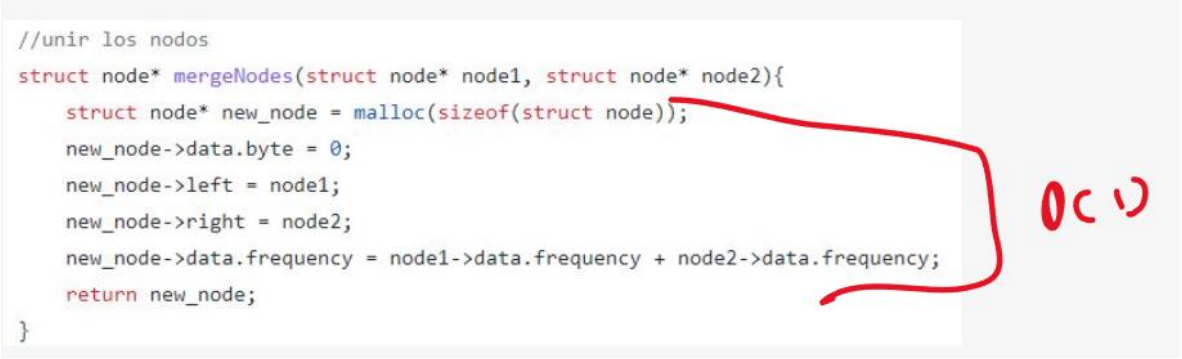


Complejidad de los programas Compress y Descompress

Función InsertTree O(n^2)

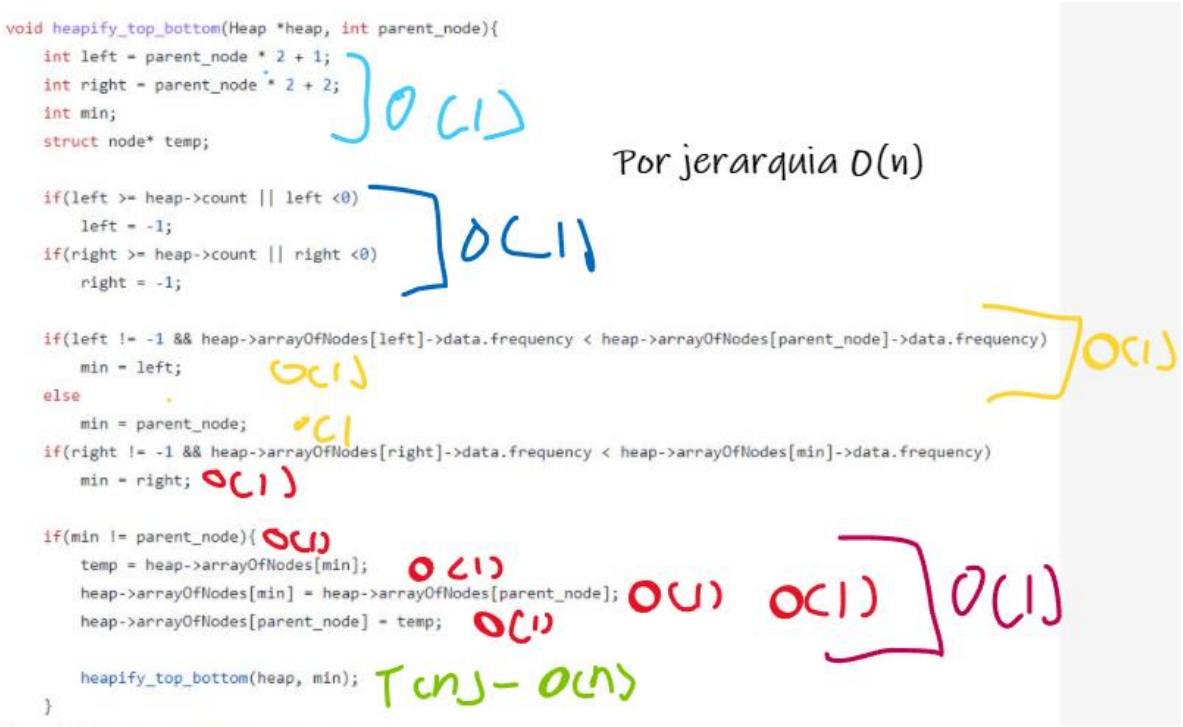
Nota se toma la complejidad de insert ya que es llamada en la función.

Función mergeNodes O(1)



Complejidad de los programas Compress y Descompress

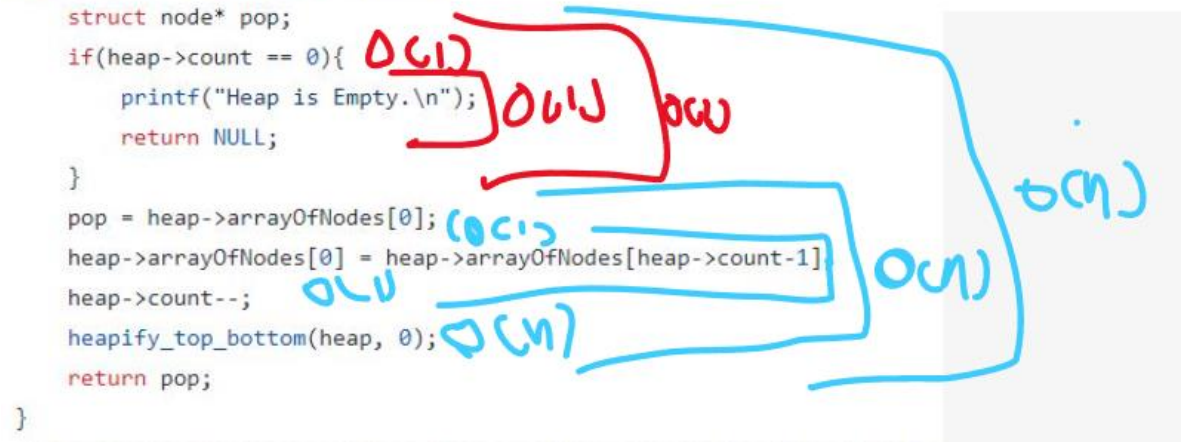
Funcon heapify_Top_bottom O(n)




```
void heapify_top_bottom(Heap *heap, int parent_node){
    int left = parent_node * 2 + 1;
    int right = parent_node * 2 + 2;
    int min;
    struct node* temp;

    if(left >= heap->count || left <0)
        left = -1;
    if(right >= heap->count || right <0)
        right = -1;

    if(left != -1 && heap->arrayOfNodes[left]->data.frequency < heap->arrayOfNodes[parent_node]->data.frequency)
        min = left;
    else
        min = parent_node;
    if(right != -1 && heap->arrayOfNodes[right]->data.frequency < heap->arrayOfNodes[min]->data.frequency)
        min = right;

    if(min != parent_node){
        temp = heap->arrayOfNodes[min];
        heap->arrayOfNodes[min] = heap->arrayOfNodes[parent_node];
        heap->arrayOfNodes[parent_node] = temp;

        heapify_top_bottom(heap, min);
    }
```

Nota hace una recursión N veces por lo que su cota de complejidad es O(n)

Función PopMin O(1)

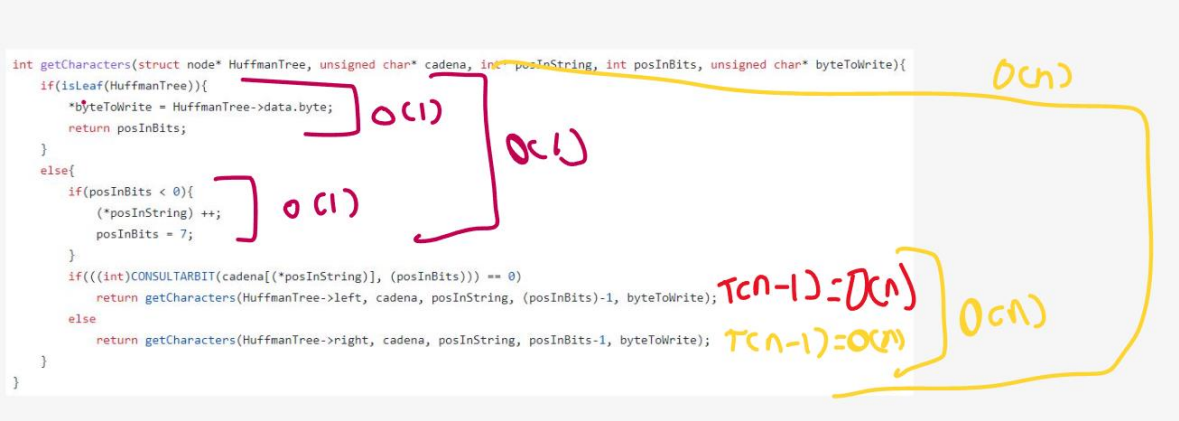



```
struct node* PopMin(Heap *heap){
    struct node* pop;
    if(heap->count == 0){
        printf("Heap is Empty.\n");
        return NULL;
    }
    pop = heap->arrayOfNodes[0];
    heap->arrayOfNodes[0] = heap->arrayOfNodes[heap->count-1];
    heap->count--;
    heapify_top_bottom(heap, 0);
    return pop;
}
```

Nota llama a la funcion heapify_Top_bottom que tiene una complejidad de O(n)

Complejidad de los programas Compress y Descompress

Funcion getCharacters O(n)



Nota: hace una recursión T(n-1) con cota de complejidad de O(n) y por jerarquía queda O(n)

Complejidad de los programas Compress y Descompress

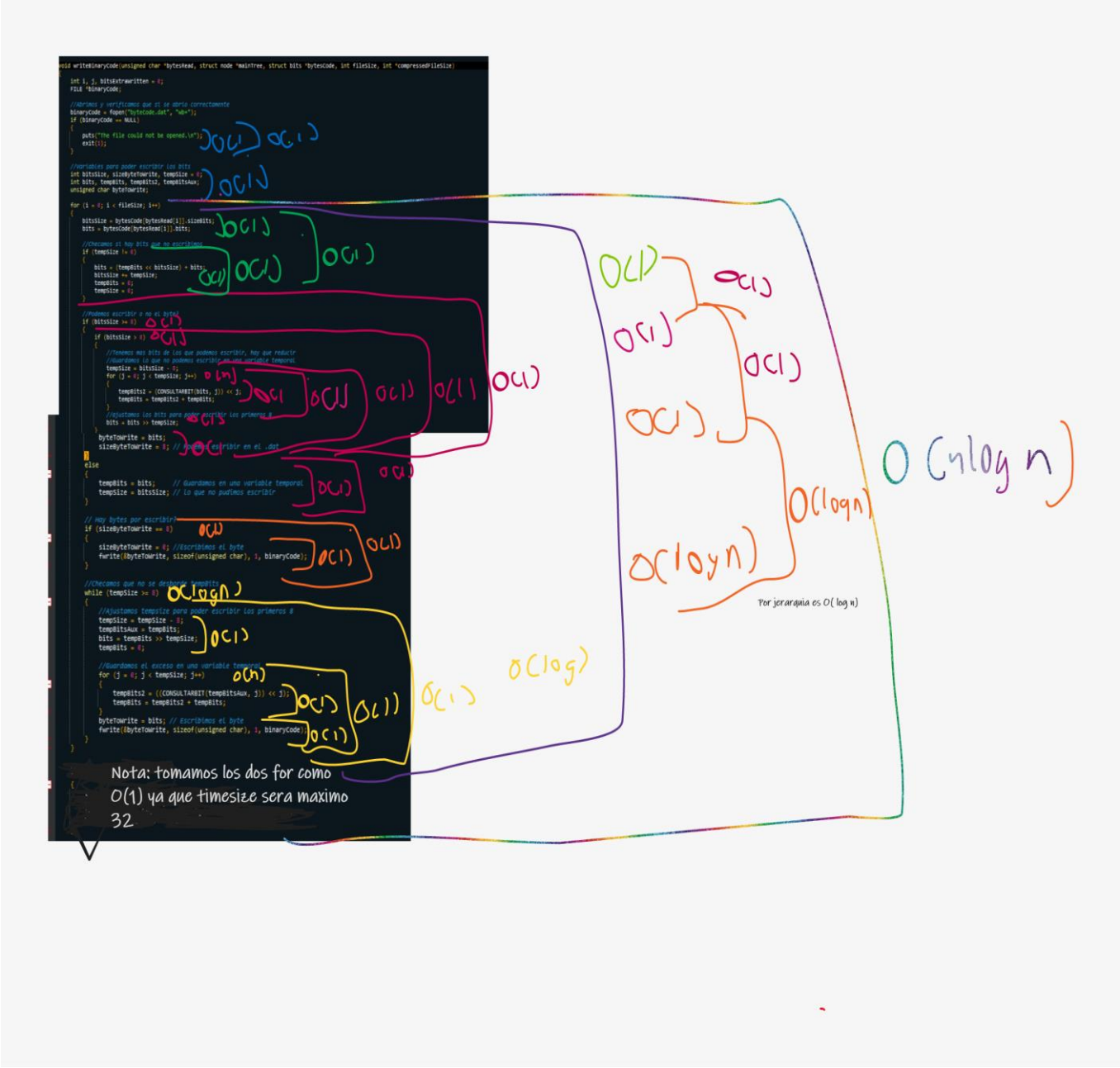
Complejidad de Funciones de CompressHufman

Función wirteFrecuencyTable O(n)

Funcion ReadFile O(n)

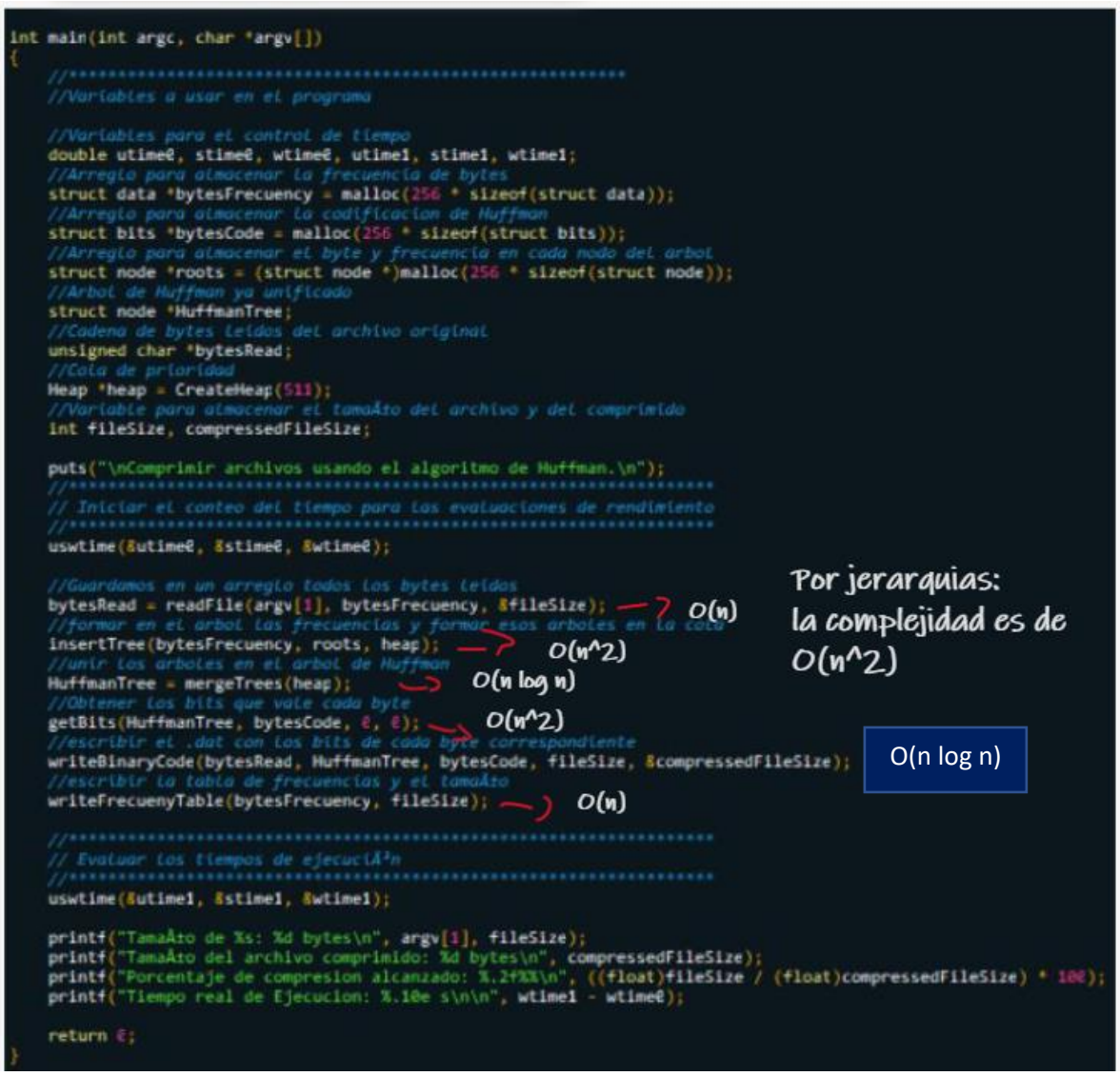
Complejidad de los programas Compress y Descompress

Funcion WriteBinaryCode es O(nlogn



Complejidad del Programa CompressHufman

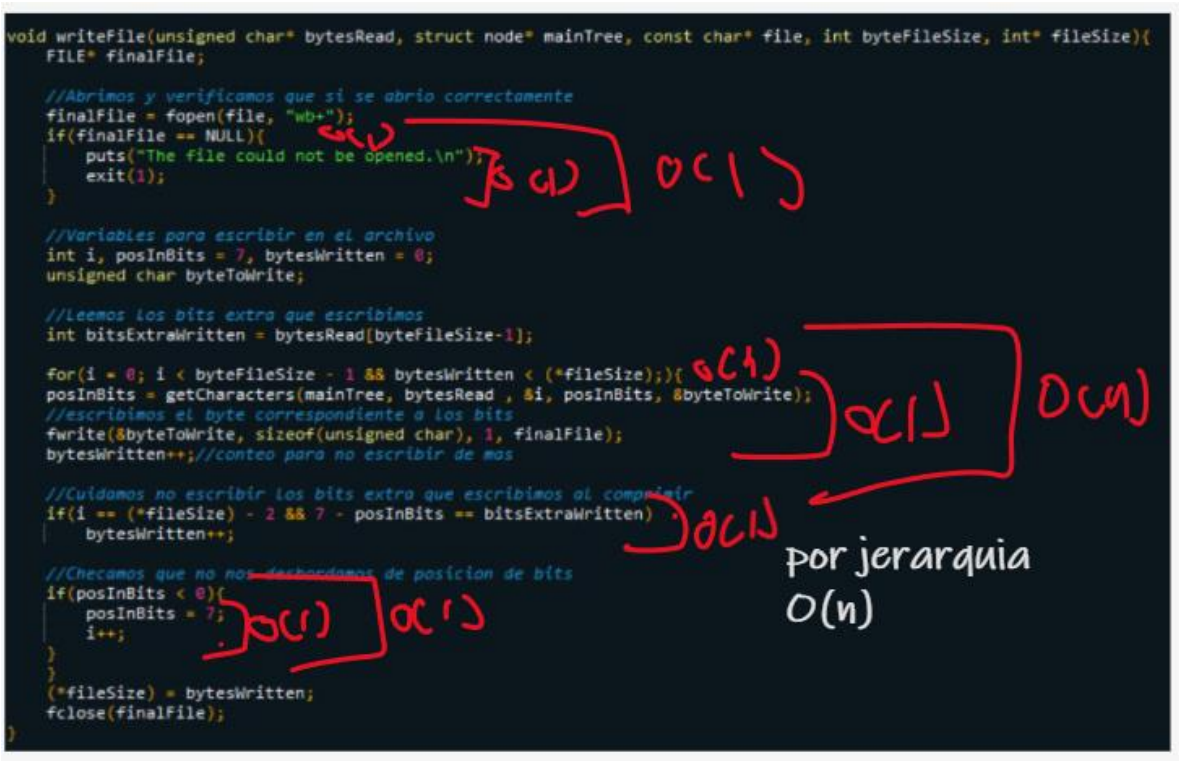
O(n^2)



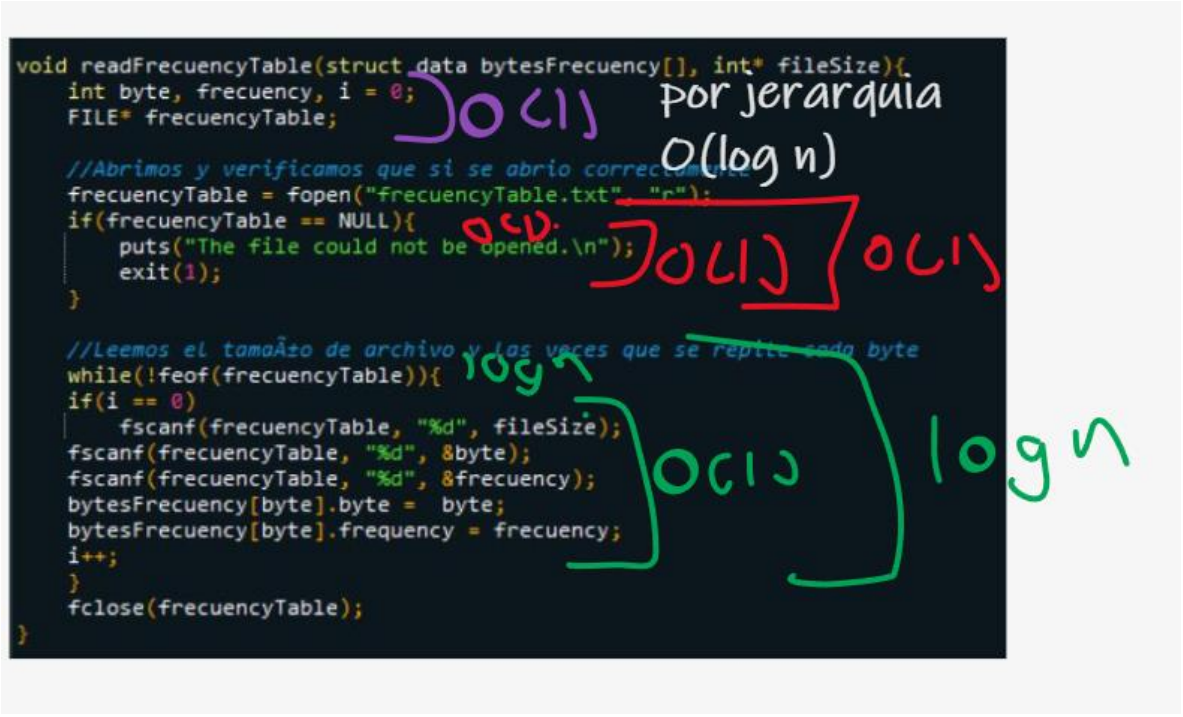
Nota: Son llamada funciones donde ya se saco su complejidad anteriormente y con jerarquía la complejidad queda como O(n^2).

Complejidad Funciones de DescompressedHumman

Funcion writeFileO(n)



Funcion readFrecuencyTable O(logn)



Complejidad de los programas Compress y Descompress

Funcion ReadByteCode O(n)

Complejidad de los programas Compress y Descompress

Complejidad Programa DescompressHumman

O(n^2)