



UNIVERSIDAD CENTROAMERICANA
JOSÉ SIMEÓN CAÑAS
FACULTAD DE INGENIERÍA Y ARQUITECTURA

Análisis de algoritmos

Taller 1

Análisis de orden de magnitud

Estudiantes:

Ordoñez Cruz, Carlos Armando, 00072122
Calderon Arguera, Oscar Armando, 00090822

Fecha de entrega: 06/09/2024

Análisis formal:

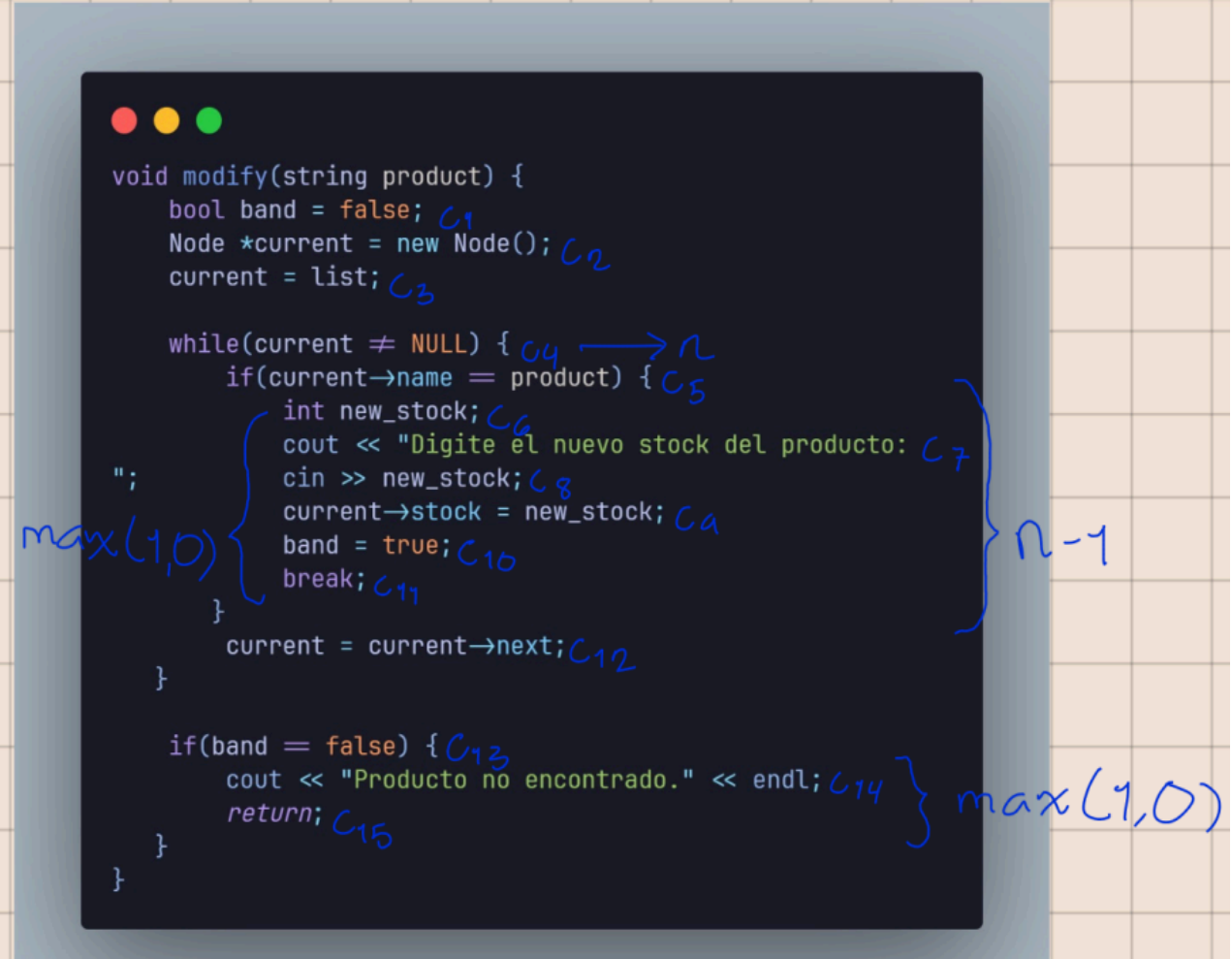
Función bubbleSort:

```
void bubbleSort() {  
    if(!list || !list->next) {  
        return;  
    }  
  
    bool swapped;  
    Node* current;  
    Node* lastSorted = nullptr;  
  
    do {  
        swapped = false;  
        current = list;  
  
        while(current->next != lastSorted) {  
            if(current->stock > current->next->stock) {  
                string tempName = current->name;  
                int tempStock = current->stock;  
  
                current->name = current->next->name;  
                current->stock = current->next->stock;  
  
                current->next->name = tempName;  
                current->next->stock = tempStock;  
  
                swapped = true;  
            }  
            current = current->next;  
        }  
        lastSorted = current;  
    } while(swapped);  
}
```

$$\begin{aligned} T(n) &= (c_1 + c_2 + c_3 + c_4 + c_5) + (c_6 + c_{20} + c_7 + c_{18} + c_{19})n + n(n-1) \\ &\quad (c_9 + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} + c_{15} + c_{16} + c_{17}) + c_8 * n * n \\ T(n) &= A + B + Cn(n-1) + c_8 n^2 = O(n^2) \end{aligned}$$

En el peor caso se tiene una lista totalmente desordenada, por lo que el número de comparaciones que realiza el bucle while tiene una magnitud cuadrática al realizar comparaciones por cada iteración.

Función Modify:



```
void modify(string product) {  
    bool band = false;  $C_1$   
    Node *current = new Node();  $C_2$   
    current = list;  $C_3$   
  
    while(current != NULL) {  $C_4 \rightarrow n$   
        if(current->name == product) {  $C_5$   
            int new_stock;  $C_6$   
            cout << "Digite el nuevo stock del producto: ";  $C_7$   
            cin >> new_stock;  $C_8$   
            current->stock = new_stock;  $C_9$   
            band = true;  $C_{10}$   
            break;  $C_{11}$   
        }  
        current = current->next;  $C_{12}$   
    }  
  
    if(band == false) {  $C_{13}$   
        cout << "Producto no encontrado." << endl;  $C_{14}$   
        return;  $C_{15}$   
    }  
}
```

Handwritten annotations in the image include:

- $\max(1,0)$ next to the first if block.
- $n-1$ next to the while loop.
- $\max(1,0)$ next to the final if block.

$$T(n) = (C_1 + C_2 + C_3 + C_{13} + C_{14} + C_{15}) + C_4 * n + (n-1) * (C_4 + C_5 + C_6 + C_7 + C_8 + C_9 + C_{10} + C_{11} + C_{12})$$

A

$$T(n) = A + C_4 * n + (n-1)B = O(n)$$

B

En el peor caso el bucle recorrerá toda la lista sin encontrar el producto buscado, lo que implica n iteraciones donde n es igual a la cantidad de nodos de la lista, obteniendo una magnitud de orden $O(n)$.

Función ShowNNodes:

```
void showNNodes(int n) {  
    Node *current = new Node();  $c_1$   
    current = list;  $c_2$   
    int j = 0;  $c_3$   
  
    while(current != NULL && j < n) {  $c_4 \rightarrow n+1$   
        cout << "Del producto " << current->name << " existen " << current->stock << " unidades." << end  $c_5$   
        current = current->next;  $c_6$   
        j++;  $c_7$   
    }  
}
```

$$T(n) = (c_1 + c_2 + c_3) + c_4(n+1) + (c_5 + c_6 + c_7)(n+1)n$$

A B

$$T(n) = A + c_4(n+1) + B(n+1)n = O(n)$$

Se recorre la lista hasta encontrar los primeros n productos con menor stock, por lo que en el peor caso la magnitud es de $O(n)$.

Función ShowLessThanTen:

```
1 void showLessThanTen() {  
2     Node *current = new Node();  $c_1$   
3     current = list;  $c_2$   
4  
5     while(current != NULL) {  $c_3 \rightarrow n+1 \rightarrow n$   
6         if(current->stock < 10) {  $c_4 \rightarrow n$   
7             cout << "Del producto " << current->name << " existen " << current->stock << " unidades." << end  $c_5 \rightarrow \max(1,0)$   
8             current = current->next;  $c_6$   
9         }  
10    }  
11 }
```

$$T(n) = (c_1 + c_2) + (n+1)c_3 + (c_4 + c_5 + c_6)n$$
$$T(n) = A + (n+1)c_3 + Bn = O(n)$$

Al recorrer la lista para encontrar todos los productos con menos de 10 unidades, en el peor caso se va a recorrer toda la lista, la magnitud es $O(n)$.

Función insert:

```
1 void insert(int stock, string name) {  
2     Node *new_node = new Node();  $c_1$   
3     new_node->stock = stock;  $c_2$   
4     new_node->name = name;  $c_3$   
5  
6     Node *aux1 = list;  $c_4$   
7     Node *aux2;  $c_5$   
8  
9     while((aux1 != NULL) && (aux1->stock < stock)) {  $c_6 \rightarrow n+1$   
10         aux2 = aux1;  $c_7$   
11         aux1 = aux1->next;  $c_8$  }  $n$   
12 }  
13  
14 if(list == aux1) {  $c_9$   
15     list = new_node;  $c_{10}$   
16 } else {  $c_{11}$   
17     aux2->next = new_node;  $c_{12}$   
18 }  
19  
20 new_node->next = aux1;  $c_{13}$   
21  
22 cout << "Elemento " << name << " insertado a lista correctamente." << endl;  $c_{14}$   
23 };
```

$$T(n) = \overset{A}{(c_1 + c_2 + c_3 + c_4 + c_5)} + c_6(n+1) + \overset{B}{(c_7 + c_8)}n + \overset{C}{(c_9 + c_{10} + c_{13} + c_{14})}$$
$$T(n) = A + c_6(n+1) + Bn + C = O(n)$$

En el peor de los casos se recorre toda la lista enlazada para insertar un producto, por lo que el while se ejecuta n veces, dando $O(n)$ como magnitud.

Función deleteProducto:

```
void deleteProducto(string producto) {  
    if(list != NULL) {  
        Node *aux_delete;  
        Node *previous = NULL;  
  
        aux_delete = list;  
  
        while((aux_delete != NULL) && (aux_delete->name != producto))  
        {  
            previous = aux_delete;  
            aux_delete = aux_delete->next;  
        }  
  
        if(aux_delete == NULL) {  
            cout << "El elemento no ha sido encontrado." << endl;  
        } else if(previous == NULL) {  
            list = list->next;  
            delete aux_delete;  
        } else {  
            previous->next = aux_delete->next;  
            delete aux_delete;  
        }  
    }  
}
```

$\rightarrow (n+1)$
 $\max(1,0)$

$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_8 + c_9) + c_5(n+1) + (c_6 + c_7)n$$
$$T(n) = A + c_5(n+1) + Bn = O(n)$$

En el peor de los casos se recorre toda la lista para encontrar el producto a eliminar, dando una magnitud de $O(n)$.

Función Menu:

```
void menu() {  
    int c = 1;  
    string producto;  
    while (c != 0){  
        cout << "===== " << endl; C1  
        cout << "Seleccione una opción: " << endl; C2  
        cout << "===== " << endl; C3  
        cout << "1. Agregar un producto" << endl; C4  
        cout << "2. Eliminar un producto" << endl; C5  
        cout << "3. Modificar un producto" << endl; C6  
        cout << "4. Ver productos con menos de 10 unidades" << endl; C7  
        cout << "5. Ver n cantidad de productos con menor stock" << endl; C8  
        cout << "6. Salir" << endl; C9  
        cout << "Digite: "; C10  
        cin >> c; C11  
  
        switch(c){ C12  
            case 1: { C13  
                string name; C14  
                int stock; C15  
                cout << "Digite un producto: "; C16  
                cin >> name; C17  
                cout << "Digite el stock: "; C18  
                cin >> stock; C19  
                insert(stock, name); C41 n  
                break; C20  
            }  
            case 2: { C21  
                cout << "Digite el producto que desea eliminar: "; C22  
                cin >> producto; C23  
                deleteProducto(producto); C42 n  
                break; C24  
            }  
            case 3: { C25  
                cout << "Digite el producto que desea modificar: "; C26  
                cin >> producto; C27  
                modify(producto); C43 n  
                bubbleSort(); C44 n2  
                break; C28  
            }  
            case 4: { C29  
                showLessThanTeen(); C45 n  
                break; C30  
            }  
        }  
    }  
}
```

```

case 5: { C31
    int n; C32
    cout << "Digite la cantidad de producto que desea ver: "; C33
    cin >> n; C34
    showNNodes(n); C46 n
    break; C35
}
case 6: { C36
    c = 0; C37
    break; C39
}
default: { C39
    cout << "Opción incorrecta" << endl; C40
}
}
}
}

```

$$T_1(n) = (c_{14} + c_{15} + c_{16} + c_{17} + c_{18} + c_{19} + c_{20}) + c_{41}n = O(n)$$

$$T_2(n) = (c_{22} + c_{23} + c_{24}) + c_{42}n = O(n)$$

$$T_3(n) = (c_{26} + c_{27} + c_{28}) + c_{43}n + c_{44}n^2 = O(n^2)$$

$$T_4(n) = c_{45}n + c_{30} = O(n)$$

$$T_5(n) = (c_{32} + c_{33} + c_{34} + c_{35}) + c_{46}n = O(n)$$

$$T_6(n) = c_{37} + c_{38} = O(1)$$

$$T_7(n) = c_{39} + c_{40} = O(1)$$

$$\max(T_1, T_2, T_3, T_4, T_5, T_6, T_7) = T_3(n) = O(n^2)$$

$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8 + c_9 + c_{10} + c_{11} + c_{12} + c_{13} + c_{21} + c_{25}) + n^2$$

$$T(n) = A + n^2 = O(n^2)$$

Al analizar la función del menú vemos que hay un switch y analizamos cada caso y tomamos el peor de los casos que nos da $O(n^2)$ por lo concluimos que nuestro código tiene un orden de convergencia de $O(n^2)$.

Conclusión

Al analizar todo el código nos dimos cuenta que este tiene un orden de convergencia de $O(n^2)$ por el algoritmo de ordenamiento que se ejecuta después de modificar el stock de un producto.