

UNIVERSIDAD CENTROAMERICANA “JOSE SIMEON CAÑAS”



Integrantes:

Luis Andree Hernández Maltéz

Edson Enrique Palacios Quinteros

Oscar Ernesto Menjivar Ayala

Asignatura:

Análisis de algoritmos

Actividad:

Taller 01

Entrega:

Viernes, 06 de septiembre

Problemática:

En un almacén se busca un sistema capaz de ordenar su productos, dichos productos tienen como datos , su nombre y su cantidad de stock, el sistema tiene como objetivo facilitar la obtención de los n-productos con menor stock.

Solución:

```
struct Product C1 · 1 = O(1)
{
    string name; C2 · 1 = O(1)
    int stock; C3 · 1 = O(1)

    Product(const string &nom = "", int stockP = 0) C4 · 1 = O(1)
    {
        name = nom; C5 · 1 = O(1)
        stock = stockP; C6 · 1 = O(1)
    }
};
```

$$T(n) = O(1)$$

$$T(n) = C_1 + C_2 + C_3 + C_4 + C_5 + C_6 = O(1)$$

```
void updateStock(Product products[], int numProducts, const string &name, int newStock) C1 · 1 = O(1)
{
    for (int i = 0; i < numProducts; ++i) C2 · (n+2) = O(n)
    {
        if (products[i].name == name) C3 · (n+1) · C7 = O(n)
        {
            products[i].stock = newStock; C4 · (n+1) · max(1,0) · C8 = O(n)
            return; C5 · (n+1) · max(1,0) · C9 = O(n)
        }
    }
    cout << "Producto no encontrado: " << name << endl; C6 · 1
```

Por simplicidad

numProducts = n

For [0, n]

$$(n-0)+1 = (n+1)$$

$$T_n = C_1 + C_2(n+2) + C_3(n+1) + C_4C_7(n+1) + C_5C_8(n+1) + C_6$$

$$T_n = O(n)$$

```
void sortProducts(Product products[], int numProducts) C1 · 1 = O(1)
{
    for (int i = 0; i < numProducts - 1; ++i) C2 · (n+1) = O(n)
    {
        int minIndex = i; C3 · n
        for (int j = i + 1; j < numProducts; ++j) C4 · n · (n-i+1)
        {
            if (products[j].stock < products[minIndex].stock) C5 · n · (n-i) · C11
            {
                minIndex = j; C6 · n · (n-i) · C12 · max(1,0)
            }
        }
        if (minIndex != i) C7 · n · (n-i) · C14
        {
            Product temp = products[i]; C8 · n · C15 · max(1,0)
            products[i] = products[minIndex]; C9 · n · C16 · max(1,0)
            products[minIndex] = temp; C10 · n · C17 · max(1,0)
        }
    }
}
```

• For [0, n-1]

$$[(n-1)-0]+1 = n$$

• For [n-(i+1)+1

$$(n-i)$$

$$T(n) = C_1 + C_2n + C_3n + C_4n(n-i+1) + C_5C_{11} + C_6C_{12} + C_7C_{14}n(n-i) + C_8C_{15} + C_9C_{16} + C_{10}C_{17}(n)$$

$$T(n) = C_1 + C_2n + C_3n + C_4n(n-i+1) + An(n-i) + Bn$$

$$T(n) = C_1 + D \cdot n + C_4 \cdot n \cdot (n-i+1) + An(n-i)$$

$$T(n) = C_1 + D \cdot n + C_4 \sum_{i=0}^{n-1} n-i+1 + A \sum_{i=0}^{n-1} n-i$$

$$T(n) = C_1 + D \cdot n + C_4 \sum_{i=0}^{n-1} n - \sum_{i=0}^{n-1} i + \sum_{i=0}^{n-1} 1 + A \sum_{i=0}^{n-1} n - \sum_{i=0}^{n-1} i$$

$$T(n) = C_1 + D \cdot n + C_4 \left(n + \sum_{i=1}^{n-1} n - 0 + \sum_{i=1}^{n-1} i + 1 + \sum_{i=1}^{n-1} 1 \right) + A \left(n + \sum_{i=1}^{n-1} n - 0 + \sum_{i=1}^{n-1} i \right)$$

$$T(n) = C_1 + D \cdot n + C_4 \left(n + \underbrace{n(n-1)}_{n^2} + \underbrace{\frac{(n-1)(n)}{2}}_{n^2} + 1 + (n-1) \right) + A \left(n + \underbrace{n(n-1)}_{n^2} + \underbrace{\frac{(n+1)(n)}{2}}_{n^2} \right)$$

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

```

void getLowStock(Product products[], int numProducts, int n) C1 · 1 = O(1)
{
    sortProducts(products, numProducts); C2 · O(n^2) = O(n^2)

    int lowStockProducts = 0; C3 · 1 = O(1)
    for (int i = 0; i < numProducts; ++i) C4 · (n+2) = O(n)
    {
        if (products[i].stock ≤ 10) C5 · C17 · (n+1) = O(n)
        {
            lowStockProducts++; C6 · C18 · (n+1) · max(1,0) = O(n)
        }
        else C7 · C19 · (n+1) = O(n)
        {
            break; C8 · C20 · (n+1) = O(n)
        }
    }

    if (lowStockProducts == 0) C9 · C21 = O(1)
    {
        cout << "No hay productos con stock igual o menor a 10." << endl; C10 · C21 · max(1,0) = O(1)
        return; C11 · C22 · max(1,0) = O(1)
    }

    if (n > lowStockProducts) C12 · C23 = O(1)
    {
        n = lowStockProducts; C13 · C24 · max(1,0) = O(1)
    }

    cout << "Los " << n << " productos con menor stock (menor o igual a 10) son:\n"; C14 = O(1)
    for (int i = 0; i < n; ++i) C15 · (n+2) = O(n)
    {
        cout << "Nombre: " << products[i].name << ", Stock: " << products[i].stock << endl; C16 · (n+1) = O(n)
    }
}

```

For
[0, n]
(n-0)+1
= n+1

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

```

        cout << "Producto no encontrado: " << nameProduct << endl; C54 · 1
    }
    break; C55 · 1
}
case 4:
    cout << "Saliendo del programa ..." << endl; C56 · 1
    return; C57 · 1
default:
    cout << "Opción no válida. Inténtelo de nuevo." << endl; C58 · 1
    break; C59 · 1
}
}

```

= O(1)

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

```

int main() C1
{
    int numProducts = 0; C2 · 1
    Product* products = nullptr; C3 · 1
    showMenu(products, numProducts); C4 · O(n^2)
    delete[] products; C5 · 1
    return 0; C6 · 1
}

```

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

Conclusión:

Para abordar la problemática del almacén es conveniente el uso de algoritmos de ordenamiento, debido a su capacidad de ordenar datos, el algoritmo “Selection Sort” fue crucial para la resolución del problema, su importancia se da debido a su funcionalidad de encontrar el elemento más pequeño de un subarreglo desordenado, Selection Sort itera el arreglo y encuentra el valor mínimo, intercambiando dicho valor con el primer elemento no arreglado y repite el procedimiento para el resto del arreglo, avanzando un elemento en cada iteración.

Debido a la serie de pasos que realiza Selection Sort, la obtención de los n-productos con menor stock tuvo una solución, ordenando los productos de menor a mayor cantidad. La comprensión e implementación del algoritmo es fácil de desarrollar, sin embargo, la eficiencia del algoritmo es de baja calidad, siendo más notorio cuando hay una gran cantidad de datos de por medio.