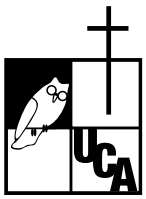
UNIVERSIDAD CENTROAMERICANA

“JOSÉ SIMEÓN CAÑAS”



Taller 1

MATERIA

ANÁLISIS DE ALGORITMOS

CATEDRÁTICO

ING. MARIO ISAAC LÓPEZ

POR:

DOMINGUEZ CORTEZ, DIEGO JOSUÉ	**00081022**
MEJÍA HERNÁNDEZ, SALVADOR MARCELO	**00072020**
RODAS ESCOBAR, KAREN ELIZABETH	**00226013**

6 DE SEPTIEMBRE DE 2024

ANTIGUO CUSCATLÁN, EL SALVADOR, C.A.

```cpp
#include <iostream>
#include <string>
#include <algorithm>
#include <limits>
using namespace std;

struct Producto {
    string nameProduct;
    int quantity;
};

// Validacion repeticion de productos
string normalizeProductName(string nameProduct) {
    //Minisculas/Mayusculas
    transform(nameProduct.begin(), nameProduct.end(), nameProduct.begin(), ::tolower);
    //Plural/Singular
    if (!nameProduct.empty() && nameProduct.back() == 's') {
        nameProduct.pop_back();
    }
    return nameProduct;
}

// Algoritmo burbuja basado en la cantidad (stock)
void bubbleSort(Producto productos[], int totalProductos) {
    for (int i = 0; i < totalProductos - 1; i++) {
        for (int j = 0; j < totalProductos - i - 1; j++) {
            if (productos[j].quantity > productos[j + 1].quantity) {
                Producto temp = productos[j];
                productos[j] = productos[j + 1];
                productos[j + 1] = temp;
            }
        }
    }
}

// Búsqueda de productos normalizando el nombre
int searchProduct(const Producto productos[], int totalProductos, const string& nameProduct) {
    string normalizedSearch = normalizeProductName(nameProduct);
    for (int i = 0; i < totalProductos; i++) {
        if (normalizeProductName(productos[i].nameProduct) == normalizedSearch) {
            return i;
```

$\theta(n)$

```
    }
  }
  return -1;
}
```

```
// Validar cantidad de productos
bool validateQuantity(int quantity) {
  return quantity >= 0 && quantity < 10000;
}
```

```
// Validar entrada numérica
int getValidatedIntInput(const string& prompt) {
  int number;
  while (true) {
    cout << prompt;
    cin >> number;
    if (cin.fail() || !validateQuantity(number)) { // Si la entrada no es válida o fuera de rango
      cout << "Entrada no válida. Debe ser un número entre 0 y 9999.\n";
      cin.clear(); // Limpia el error
      cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Descarta la entrada incorrecta
    } else {
      break; // Salir del bucle si la entrada es válida
    }
  }
  return number;
}

// Agregar un producto con las validaciones
void addProduct(Producto productos[], int &totalProductos) {
  string nameProductProducto;
  bool productoExistente = false;

  // Bucle para seguir pidiendo el nombre hasta que se ingrese uno válido
  do {
    cout << "Ingrese el nombre del producto: ";
    cin.ignore();
    getline(cin, nameProductProducto);

    // Verificar si el producto ya existe usando la función de normalización
    if (searchProduct(productos, totalProductos, nameProductProducto) != -1) {
      cout << "El producto '" << nameProductProducto << "' ya existe en el inventario.\n";
      productoExistente = true;
    } else {
      productoExistente = false; // Nombre válido
```

```
    }
  } while (productoExistente);

  // Validar la cantidad
  int cantidad = getValidatedIntInput("Ingrese la cantidad para el producto (0-9999): ");

  productos[totalProductos].nameProduct = nameProductProducto;
  productos[totalProductos].quantity = cantidad;
  totalProductos++;
  cout << "Producto agregado al inventario.\n";
}
```

```
// Mostrar lista de productos
void updateProduct(Producto productos[], int totalProductos) {
  // Mostrar la lista de productos disponibles
  cout << "\n--- Productos Disponibles ---\n";
  for (int i = 0; i < totalProductos; i++) {
    cout << i + 1 << ". Producto: " << productos[i].nameProduct << ", cantidad: " << productos[i].quantity << endl;
  }

  string nameProductProducto;
  cout << "Ingrese el nombre del producto que desea actualizar: ";
  cin.ignore();
  getline(cin, nameProductProducto);

  int indice = searchProduct(productos, totalProductos, nameProductProducto);

  if (indice == -1) {
    cout << "El producto '" << nameProductProducto << "' no existe en el inventario.\n";
    return;
  }

  // Editar nombre del producto
  string nuevoNombre;
  cout << "Ingrese el nuevo nombre para el producto '" << productos[indice].nameProduct << "' (o presione ENTER
para no cambiarlo): ";
  getline(cin, nuevoNombre);
  if (!nuevoNombre.empty()) {
    if (searchProduct(productos, totalProductos, nuevoNombre) == -1) {
      productos[indice].nameProduct = nuevoNombre;
    } else {
      cout << "El nombre del producto ya existe en el inventario.\n";
    }
  }
```

```
    // Editar cantidad del producto
    int nuevaCantidad = getValidatedIntInput("Ingrese la nueva cantidad para el producto (0-9999): ");
    productos[indice].quantity = nuevaCantidad;
    cout << "Producto actualizado exitosamente.\n";
}

// Mostrar lista de productos antes de eliminar
void deleteProduct(Producto productos[], int &totalProductos) {
    // Mostrar la lista de productos antes de eliminar
    cout << "\n--- Productos Disponibles para Eliminar ---\n";
    for (int i = 0; i < totalProductos; i++) {
        cout << i + 1 << ". Producto: " << productos[i].nameProduct << ", cantidad: " << productos[i].quantity << endl;
    }

    string nameProductProducto;
    cout << "Ingrese el nombre del producto a eliminar: ";
    cin.ignore();
    getline(cin, nameProductProducto);

    int indice = searchProduct(productos, totalProductos, nameProductProducto);

    if (indice == -1) {
        cout << "El producto '" << nameProductProducto << "' no existe en el inventario.\n";
        return;
    }

    for (int i = indice; i < totalProductos - 1; i++) { [ j , n-1] → n-1+j+1 = n+j
        productos[i] = productos[i + 1];
    }

    totalProductos--;
    cout << "Producto '" << nameProductProducto << "' se ha eliminado correctamente.\n";
}

// Mostrar inventario en orden ascendente
void showInventory(Producto productos[], int totalProductos) {
    bubbleSort(productos, totalProductos);

    for (int i = 0; i < totalProductos; i++) { [0, n] → n-0+1 = n+1
        cout << "Producto: " << productos[i].nameProduct << ", cantidad: " << productos[i].quantity << endl;
    }
}
```

O(1)
O(n²)
O(1)
O(1)
O(1)
O(1) → O(n²)
O(1)
O(n+1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(n)
O(n²)
O(1)
O(1)
O(1)
O(1)
O(n+j) ≈ O(n)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1) → O(n²)
O(n²) > O(n²)
O(n+1)
O(1)
O(1)
O(1)

```
// Ver productos con stock bajo
void productsInStock(Producto productos[], int totalProductos) {
    int n;
    cout << "Ingrese el número de productos con bajo stock que desea ver: ";
    cin >> n;

    bubbleSort(productos, totalProductos);

    cout << "\n--- Productos que necesitan reabastecimiento ---\n";
    int contador = 0;

    for (int i = 0; i < totalProductos && contador < n; i++) {
        if (productos[i].quantity < 10) {
            cout << "Producto: " << productos[i].nameProduct << ", cantidad: " << productos[i].quantity << endl;
            contador++;
        }
    }

    if (contador == 0) {
        cout << "No hay productos con stock menor a 10.\n";
    }
}
```

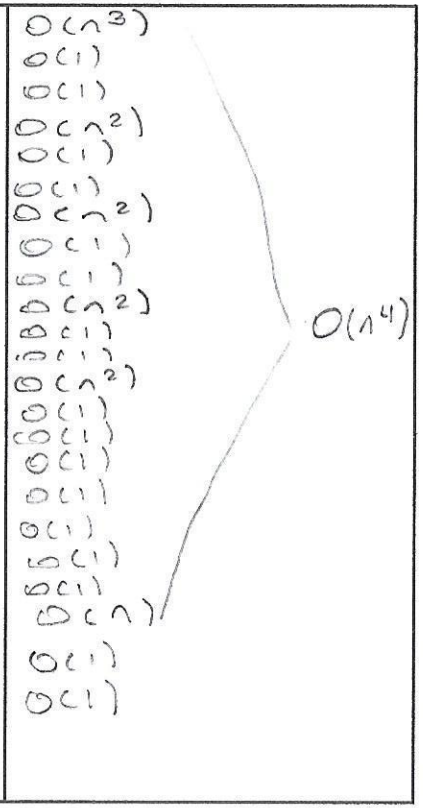
```
int main() {
    const int MAX_PRODUCTOS = 100;
    Producto productos[MAX_PRODUCTOS];
    int totalProductos = 0;
    int opcion;

    do {
        cout << "\n----------------------------------------------------------------------------\n";
        cout << "BIENVENIDOS A NUESTRO GESTION DE INVENTARIO SALEM, seleccione una opción \n";
        cout << "----------------------------------------------------------------------------\n";
        cout << "1. Agregar Producto al inventario\n";
        cout << "2. Mostrar Inventario\n";
        cout << "3. Verificar Productos con menor a 10 cantidades en stock\n";
        cout << "4. Actualizar Producto\n";
        cout << "5. Eliminar Producto\n";
        cout << "6. Salir\n";
        cout << "Seleccione una opción: ";
        cin >> opcion;

        switch(opcion) {
            case 1:
```

```
            addProduct(productos, totalProductos);
            break;
        case 2:
            showInventory(productos, totalProductos);
            break;
        case 3:
            productsInStock(productos, totalProductos);
            break;
        case 4:
            updateProduct(productos, totalProductos);
            break;
        case 5:
            deleteProduct(productos, totalProductos);
            break;
        case 6:
            cout << "Saliendo del programa...\n";
            break;
        default:
            cout << "Opción no válida, elija correctamente por favor.\n";
    }
  } while (opcion != 6);

  return 0;
}
```



Se puede concluir que el programa realizado tiene orden de magnitud: O(n^4).

Validaciones que se realizaron:

- Hemos validado que los productos no se puedan repetir, tomando en cuenta mayúsculas, minúsculas, plural y singular
- Hemos validado que la cantidad de productos no sea negativa ni mayor a 10,000 para que el programa tenga una mayor eficiencia. (en la función validateQuantity se puede modificar el valor para incrementar el limite del stock en los productos)
- Se hace uso de validaciones generales en campos numéricos y campos string