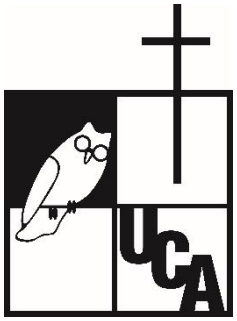

Universidad Centroamericana "José Simeón Cañas"

Facultad de Ingeniería y Arquitectura

Implementación y análisis del algoritmo para la gestión de productos en stock

Integrantes del equipo:

Michelle Stefania Hernandez Flamenco

Jonathan Rodrigo Zelaya Alvarado

Eduardo Alessandro Rivera Diaz

Catedráticos:

Enmanuel Amaya Araujo, MSc.

Ing. Mario Isaac López

Antiguo Cuscatlán, 6 de septiembre de 2024

INDICE

Introducción

El presente informe presenta la implementación y el análisis para la gestión de productos en stock de un almacén. El código, escrito en el lenguaje de programación C++, hace uso de listas enlazadas, funciones y punteros. Posteriormente, se analiza para determinar el orden de magnitud de cada línea de código, tomando en cuenta las operaciones y bucles involucrados.

La problemática a resolver es la siguiente:

El almacén Salem necesita un sistema que le permita organizar todos los productos con los que cuenta en sus instalaciones, y para cumplir este propósito ha contratado un equipo de desarrolladores (ustedes, estudiantes) que puedan generar una solución.

El almacén maneja, para cada producto, dos datos: su nombre y su cantidad en stock. Sin embargo, como hasta la fecha se han manejado los datos en cuadernos, existe un desorden generalizado, apuntando las cosas conforme van llegando. Por lo tanto, el gerente ha expresado que desea que el sistema a desarrollar le provea los datos de una manera que le sea fácil ubicar cualquier producto de su interés.

El sistema debe identificar todos los productos que necesiten reabastecimiento, para que el gerente tenga en mente llamar a los respectivos proveedores. Se considera que un producto está en esta situación cuando quedan en bodega menos de 10 unidades.

Se busca que ustedes desarrollen un algoritmo que facilite la obtención de los n productos que tengan el menor stock; es decir, deben desarrollar una función (sin utilizar ningún tipo de librería estándar o externa) que devuelva un listado de los n productos con menor stock.

Por último, se requiere una manera de actualizar el stock de los productos.

El proyecto presentado consiste en un pequeño sistema que ha sido diseñado para manejar el stock de un inventario que incluye solamente el nombre y cantidad disponible de cada producto.

Para realizar la solución se ha hecho uso de una lista que permite dar un seguimiento organizado a los productos que se vayan añadiendo al sistema. La aplicación permite revisar cuales son los productos que necesitan reabastecimiento, para esto, se hace una validación de la cantidad que hay de ese producto, en este caso, el producto necesita reabastecimiento si hay menos de 10, adicionalmente, el sistema permite elegir la cantidad de productos que necesitan reabastecimiento que el usuario desea ver, mostrándolos en orden de menor a mayor stock.

Descripción del código

El programa es un sistema completo para la gestión de inventario en un almacén, desarrollado utilizando una lista enlazada simple como estructura de datos. El inventario se maneja mediante nodos, donde cada nodo representa un producto con atributos como nombre, cantidad en stock y un puntero al siguiente producto en la lista.

Las principales funcionalidades del programa incluyen:

- **Agregar productos:** Permite al usuario ingresar un nuevo producto al inventario, especificando su nombre y cantidad en stock. El producto se añade al final de la lista enlazada.
- **Actualizar productos:** El usuario puede seleccionar un producto de un listado para modificar su nombre o cantidad en stock, sin necesidad de escribir el nombre del producto manualmente.
- **Eliminar productos:** Se puede eliminar un producto del inventario seleccionándolo de la lista, lo que implica eliminar su nodo correspondiente y ajustar los punteros de la lista enlazada para mantener la estructura.
- **Mostrar todos los productos:** Muestra un listado de todos los productos ingresados en el inventario, indicando su nombre y cantidad en stock.
- **Mostrar productos que necesitan reabastecimiento:** Identifica y muestra aquellos productos cuyo stock está por debajo de un umbral definido, lo que ayuda a gestionar la reposición de inventario.
- **Liberar memoria:** Al finalizar la ejecución, el programa recorre la lista enlazada liberando la memoria asignada para cada nodo, evitando fugas de memoria.

El programa incluye un menú interactivo que permite al usuario navegar entre estas opciones de manera sencilla. Además, no se utilizan librerías externas como 'vector' o 'algorithms', implementando manualmente las operaciones sobre la lista enlazada.

Código en C++

```
#include <iostream>
#include <string>

using namespace std;

// Declaracion de la estructura Nodo para almacenar cada producto en
el inventario
struct Nodo
{
  string nombre;   // Nombre del producto
  int cantidad;    // Cantidad de productos en stock
  Nodo *siguiente; // Puntero al siguiente nodo de la lista enlazada
};

// Declaracin de constantes globales

const int kStockMinimo = 10; // Cantidad mínima de stock para
reabastecimiento

// Prototipos de las funciones para manejar el inventario
void agregar_producto(Nodo *&head, const string &nombre, int
cantidad);
void actualizar_producto(Nodo *head);
void eliminar_producto(Nodo *&head);
void mostrar_productos(Nodo *head);
void reabastecer_productos(Nodo *head);
void menos_stock(Nodo *head, int n);
void liberar_memoria(Nodo *&head);

int main(int argc, char *argv[])
{
  // Variables para almacenar la opcion del menu y los datos del
producto
  int opcion, cantidad, numProductos;
  string nombre;

  // Puntero inicial del inventario (lista enlazada vacia al
principio)
  Nodo *inventario = nullptr;

  // Bucle principal del menu
  while (true)
  {
    // Mostramos las opciones disponibles para gestionar el inventario
    cout << "\n========== Menu ==========\n\n";
    cout << "1. Agregar producto\n";
    cout << "2. Modificar producto\n";
    cout << "3. Productos a rebastecer\n";
    cout << "4. Mostrar inventario\n";
    cout << "5. Eliminar producto\n";
    cout << "6. Productos con menor stock\n";
    cout << "7. Salir";
    cout << "\n\nSeleccione una opcion: ";
    cin >> opcion;
```

```
    cin.ignore(); // Ignoramos el salto de linea restante en el buffer

    switch (opcion)
    {
    case 1:
      // Pedimos al usuario que ingrese los detalles del nuevo
producto
      cout << "\nIngrese el nombre del producto: ";
      getline(cin, nombre);
      cout << "\nIngrese la cantidad en stock: ";
      cin >> cantidad;
      agregar_producto(inventario, nombre, cantidad); // Llamamos a la
funcion para agregar el producto
      break;
    case 2:
      actualizar_producto(inventario); // Llamamos a la funcion para
modificar un producto existente
      break;
    case 3:
      reabastecer_productos(inventario); // Mostramos los productos
que necesitan reabastecimiento
      break;
    case 4:
      mostrar_productos(inventario); // Mostramos el inventario
completo
      break;
    case 5:
      eliminar_producto(inventario); // Permitimos al usuario eliminar
un producto
      break;
    case 6:;
      cout << "\nIngrese el numero de productos a mostrar: ";
      cin >> numProductos;
      menos_stock(inventario, numProductos); // Llamamos a la funcion
para mostrar n productos con menor stock
      break;

    case 7:

      liberar_memoria(inventario); // Liberamos la memoria de todos
los productos antes de salir
      cout << "\nSaliendo del sistema...\n\n";
      return 0;
    default:
      // Si el usuario ingresa una opcion invalida, mostramos un
mensaje
      cout << "\nOpcion no valida, intente de nuevo.\n\n";
    }
  }
  return 0;
}

// Funcion para agregar un producto al inventario
void agregar_producto(Nodo *&head, const string &nombre, int cantidad)
{
  // Creamos un nuevo nodo con los datos del producto
  Nodo *nuevo_nodo = new Nodo{nombre, cantidad, nullptr};
```

```
  // Si la lista esta vacia, el nuevo nodo sera el primero
  if (!head)
  {
    head = nuevo_nodo;
  }
  else
  {
    // Si ya hay productos, recorremos la lista hasta el final
    Nodo *actual = head;
    while (actual->siguiente)
    {
      actual = actual->siguiente;
    }
    // Insertamos el nuevo nodo al final de la lista
    actual->siguiente = nuevo_nodo;
  }
  // Informamos que el producto fue agregado
  cout << "\nProducto '" << nombre << "' agregado con " << cantidad <<
" unidades.\n\n";
}

// Funcion para modificar el nombre o la cantidad de un producto en el
inventario
void actualizar_producto(Nodo *head)
{
  // Verificamos si la lista esta vacia
  if (!head)
  {
    cout << "\nEl inventario esta vacio.\n\n";
    return;
  }

  // Mostramos los productos con su numero para que el usuario elija
cual modificar
  cout << "\nSeleccione el producto que desea modificar:\n";
  Nodo *actual = head;
  int index = 1;

  // Recorremos la lista y mostramos los productos con su cantidad
  while (actual)
  {
    cout << index << ". " << actual->nombre << " (Cantidad: " <<
actual->cantidad << ")\n";
    actual = actual->siguiente;
    index++;
  }

  // Le pedimos al usuario que ingrese el numero del producto a
modificar
  int opcion;
  cout << "\nIngrese el numero del producto: ";
  cin >> opcion;

  // Verificamos si la opcion es valida
  if (opcion < 1 || opcion >= index)
  {
```

```
    cout << "\nOpcion no valida.\n\n";
    return;
  }

  // Recorremos la lista hasta encontrar el producto seleccionado
  actual = head;
  for (int i = 1; i < opcion; ++i)
  {
    actual = actual->siguiente;
  }

  // Preguntamos si desea modificar el nombre o la cantidad
  cout << "\nQue desea modificar?\n";
  cout << "1. Nombre del producto\n";
  cout << "2. Cantidad en stock\n";
  cout << "\nSeleccione una opcion: ";
  cin >> opcion;
  cin.ignore();

  // Segun la eleccion del usuario, actualizamos el nombre o la
cantidad
  switch (opcion)
  {
  case 1:
    cout << "\nIngrese el nuevo nombre: ";
    getline(cin, actual->nombre);
    cout << "\nNombre actualizado.\n\n";
    break;
  case 2:
    cout << "\nIngrese la nueva cantidad: ";
    cin >> actual->cantidad;
    cout << "\nCantidad actualizada.\n\n";
    break;
  default:
    cout << "\nOpcion no valida.\n\n";
  }
}

// Funcion para eliminar un producto del inventario
void eliminar_producto(Nodo *&head)
{
  // Verificamos si la lista esta vacia
  if (!head)
  {

    cout << "\n\nEl inventario esta vacio.\n\n";
    return;
  }

  // Mostramos los productos con su numero para que el usuario elija
cual eliminar
  cout << "\n\nSeleccione el producto que desea eliminar:\n\n";
  Nodo *actual = head;
  int index = 1;

  // Recorremos la lista y mostramos los productos
  while (actual)
```

```
  {

    cout << index << ". " << actual->nombre << " (Cantidad: " <<
actual->cantidad << ")\n";
    actual = actual->siguiente;
    index++;
  }

  // Le pedimos al usuario que ingrese el numero del producto a
eliminar
  int opcion;
  cout << "\n\nIngrese el numero del producto: ";
  cin >> opcion;

  // Verificamos si la opcion es valida
  if (opcion < 1 || opcion >= index)
  {

    cout << "\n\nOpcion no valida.\n\n";
    return;
  }

  // Si es el primer producto, actualizamos el puntero de la cabeza
  if (opcion == 1)
  {

    Nodo *temp = head;
    head = head->siguiente;
    delete temp;
    cout << "\n\nProducto eliminado.\n\n";
    return;
  }

  // Recorremos la lista hasta el producto anterior al seleccionado
  actual = head;
  for (int i = 1; i < opcion - 1; ++i)
  {

    actual = actual->siguiente;
  }

  // Eliminamos el nodo seleccionado
  Nodo *temp = actual->siguiente;
  actual->siguiente = temp->siguiente;
  delete temp;

  cout << "\n\nProducto eliminado.\n\n";
}

// Funcion para mostrar los productos que necesitan reabastecimiento
(menos de kStockMinimo unidades)
void reabastecer_productos(Nodo *head)
{

  cout << "\n\nProductos que necesitan reabastecimiento:\n";
  Nodo *actual = head;
```

```
  // Recorremos la lista y mostramos los productos con stock menor a
kStockMinimo
  while (actual)
  {
    if (actual->cantidad < kStockMinimo)
    {
      cout << actual->nombre << ": " << actual->cantidad << "
unidades\n";
    }
    actual = actual->siguiente;
  }
  cout << "\n";
}

// Funcion para mostrar todo el inventario
void mostrar_productos(Nodo *head)
{

  cout << "\nInventario actual:\n\n";
  Nodo *actual = head;

  // Recorremos la lista y mostramos todos los productos
  while (actual)
  {
    cout << "Producto: " << actual->nombre << ", Cantidad: " <<
actual->cantidad << "\n";
    actual = actual->siguiente;
  }
  cout << "\n";
}

// Funcion para mostrar los n productos con menor stock
void menos_stock(Nodo *head, int n)
{
  if (!head)
  {
    cout << "\n\nEl inventario está vacío.\n\n";
    return;
  }

  // Contar el numero total de productos en la lista enlazada
  int total_productos = 0;
  Nodo *temp = head;
  while (temp)
  {
    total_productos++;
    temp = temp->siguiente;
  }

  // Si n es mayor que el numero total de productos, ajustar n al
total
  if (n > total_productos)
  {
    n = total_productos;
  }
```

```
  // Ordena los productos por cantidad de stock usando una el
algoritmo de burbuja
  for (int i = 0; i < total_productos - 1; i++)
  {
    Nodo *actual = head;
    Nodo *siguiente = actual->siguiente;
    for (int j = 0; j < total_productos - i - 1; j++)
    {
      if (actual->cantidad > siguiente->cantidad)
      {
        // Intercambiaa los valores de nombre y cantidad entre actual
y siguiente
        swap(actual->nombre, siguiente->nombre);
        swap(actual->cantidad, siguiente->cantidad);
      }
      actual = siguiente;
      siguiente = siguiente->siguiente;
    }
  }

  // Mostrar los primeros n productos con menor stock
  cout << "\n\nLos " << n << " productos con menor stock son:\n";
  temp = head;
  for (int i = 0; i < n && temp; i++)
  {
    cout << temp->nombre << " (Cantidad: " << temp->cantidad << ")\n";
    temp = temp->siguiente;
  }
  cout << "\n";
}

// Funcion para liberar la memoria de toda la lista enlazada
void liberar_memoria(Nodo *&head)
{
  while (head)
  {
    Nodo *temp = head;
    head = head->siguiente;
    delete temp;
  }
}
```

Análisis de orden de magnitud

Análisis de las librerías

```
#include <iostream> ---> c1 O(1)--> directiva de preprocesador
#include <string>  ----> c2 O(1)----> directiva de preprocesador para biblioteca

using namespace std;--------> c3  O(1)Uso de biblioteca estandar
```

Análisis de la estructura y constantes globales

```
// Declaracion de la estructura Nodo para almacenar cada producto en el inventario
struct Nodo--> c4 ---> O(1) Declaracion de estructura
{
  string nombre;    // Nombre del producto
  int cantidad;     // Cantidad de productos en stock
  Nodo *siguiente; // Puntero al siguiente nodo de la lista enlazada
};

// Declaracion de constantes globales

const int kStockMinimo = 10; c5 -----> 0(1)Declaración de constante
```

Análisis de los prototipos de las funciones

```
// Prototipos de las funciones para manejar el inventario
void agregar_producto(Nodo *&head, const string &nombre, int cantidad);c6 --> 0(1)
void actualizar_producto(Nodo *head);c7 --> 0(1) Declaracion de funcion
void eliminar_producto(Nodo *&head);c8 --> 0(1) Declaracion de funcion
void mostrar_productos(Nodo *head);c9 --> 0(1) Declaracion de funcion
void reabastecer_productos(Nodo *head); c10 --> 0(1) Declaracion de funcion
void menos_stock(Nodo *head, int n); c11 --> 0(1) Declaracion de funcion
void liberar_memoria(Nodo *&head); c12 --> 0(1) Declaracion de funcion
```

Análisis de la función agregar productos

```
// Funcion para agregar un producto al inventario
void agregar_producto(Nodo *&head, const string &nombre, int cantidad)c60 --> 0(1)
declaracion de la funcion
{
  // Creamos un nuevo nodo con los datos del producto
  Nodo *nuevo_nodo = new Nodo{nombre, cantidad, nullptr};c61 --> 0(1) inicializacion de
objeto


  // Si la lista esta vacia, el nuevo nodo sera el primero
  if (!head)c62 --> 0(1) condicional
  {
    head = nuevo_nodo;c63 --> asignar puntero que apunta al primer nodo
  }
  else c64 --> 0(1) bloque condicional alternativo
  {
    // Si ya hay productos, recorremos la lista hasta el final
    Nodo *actual = head; c65 --> 0(1) Declaracion de puntero actual
    while (actual->siguiente)c66 --> 0(n) bucle
    {
      actual = actual->siguiente;c67 --> 0(1) actualización de puntero
    }
    // Insertamos el nuevo nodo al final de la lista
    actual->siguiente = nuevo_nodo;c68 --> 0(1) puntero apuntando al ultimo nodo
  }
  // Informamos que el producto fue agregado
  cout << "\nProducto '" << nombre << "' agregado con " << cantidad << " unidades.\n\n";c69
--> 0(1) flujo de salida
}
```

Análisis de la función actualizar producto

```
  // Funcion para modificar el nombre o la cantidad de un producto en el inventario
  void actualizar_producto(Nodo *head)c70 --> 0(1) declaracion de la funcion
  {
    // Verificamos si la lista esta vacia
    if (!head)c71 --> 0(1) condicional
    {
      cout << "\nEl inventario esta vacio.\n\n";c72 --> 0(1) flujo de salida
      return;c73 --> 0(1) instrucción de retorno
    }


    // Mostramos los productos con su numero para que el usuario elija cual modificar
    cout << "\nSeleccione el producto que desea modificar:\n";c74 --> 0(1) instruccion de
  salida
    Nodo *actual = head;c75 --> 0(1) Declaracion de puntero actual
    int index = 1;c76 --> 0(1) Declaracion de variable int
```

```
// Recorremos la lista y mostramos los productos con su cantidad
  while (actual) c77 --> 0(n) bucle while
  {
    cout << index << ". " << actual->nombre << " (Cantidad: " << actual->cantidad <<
")\n";c78 --> 0(1) flujo de salida
    actual = actual->siguiente; c79 --> 0(1) puntero apuntando al siguiente nodo
    index++;c80 --> 0(1) incremento de variable
  }


  // Le pedimos al usuario que ingrese el numero del producto a modificar
  int opcion;c81 --> 0(1) Declaracion de variable int
  cout << "\nIngrese el numero del producto: ";c82 --> 0(1) flujo de salida
  cin >> opcion; c83 --> 0(1) flujo de entrada


  // Verificamos si la opcion es valida
  if (opcion < 1 || opcion >= index) c84 --> 0(1) condicional
  {
    cout << "\nOpcion no valida.\n\n"; c85 --> 0(1) flujo de salida
    return; c86 --> 0(1) instrucción de retorno
  }


  // Recorremos la lista hasta encontrar el producto seleccionado
  actual = head; c87 --> 0(1) asignacion de variable
  for (int i = 1; i < opcion; ++i) c88 --> 0(n) bucle
  {
    actual = actual->siguiente;c89 --> 0(1) puntero apuntando al siguiente nodo
  }


  // Preguntamos si desea modificar el nombre o la cantidad
  cout << "\nQue desea modificar?\n";c90 --> flujo de salida
  cout << "1. Nombre del producto\n";c91 --> flujo de salida
  cout << "2. Cantidad en stock\n";c92 --> flujo de salida
  cout << "\nSeleccione una opcion: ";c93--> flujo de salida
  cin >> opcion;c94 --> flujo de salida
  cin.ignore();c95 --> implementacion de funcion ignore

  // Segun la eleccion del usuario, actualizamos el nombre o la cantidad
  switch (opcion)c96 --> 0(1) expresion de control
  {
  case 1: c97 --> 0(1) etiqueta de caso
    cout << "\nIngrese el nuevo nombre: ";c98 --> 0(1) flujo de salida
    getline(cin, actual->nombre);c99 --> 0(1) extraccion de linea
    cout << "\nNombre actualizado.\n\n";c100 --> 0(1) flujo de salida
    break;c101 --> instruccion de interrupcion
  case 2: c102 --> 0(1) etiqueta de caso
    cout << "\nIngrese la nueva cantidad: ";c103 --> 0(1) flujo de salida
    cin >> actual->cantidad; c104 --> 0(1) flujo de entrada
    cout << "\nCantidad actualizada.\n\n"; c105 --> 0(1) flujo de salida
    break;c106 --> 0(1) instruccion de interrupcion
  default: c107 --> 0(1) expresion  de caso por defecto
    cout << "\nOpcion no valida.\n\n";c108 --> 0(1) flujo de salida
  }
}
```

Análisis de la función eliminar producto

```
// Funcion para eliminar un producto del inventario
void eliminar_producto(Nodo *&head) c109---> O(1) declaracion de la funcion
{
  // Verificamos si la lista esta vacia
  if (!head) c110---> O(1) cabecera del if, solo verifica si la lista esta vacia
  {


    cout << "\n\nEl inventario esta vacio.\n\n"; c111---> O(1) muestra un mensaje
    return; c112---> O(1) return del if
  }


  // Mostramos los productos con su numero para que el usuario elija cual eliminar
  cout << "\n\nSeleccione el producto que desea eliminar:\n\n"; c113---> O(1) muestra
un mensaje
  Nodo *actual = head; c114---> O(1) declara el nodo actual
  int index = 1; c115---> O(1) crea una variable entera


  // Recorremos la lista y mostramos los productos
  while (actual) c116---> O(1) el bucle recorre todos los nodos de la lista, por lo
que depende de n
  {


    cout << index << ". " << actual->nombre << " (Cantidad: " << actual->cantidad <<
")\n"; c117---> O(1) muestra un mensaje
    actual = actual->siguiente; c118---> O(1) actualiza una variable
    index++; c119---> O(1) actualiza una variable
  }


  // Le pedimos al usuario que ingrese el numero del producto a eliminar
  int opcion; c120---> O(1) declara una variable
  cout << "\n\nIngrese el numero del producto: "; c121---> O(1) muestra un mensaje
  cin >> opcion; c122---> O(1) lee un input del usuario


  // Verificamos si la opcion es valida
  if (opcion < 1 || opcion >= index) c123---> O(1) cabecera del if, compara numeros
  {


    cout << "\n\nOpcion no valida.\n\n"; c124---> O(1) muestra un mensaje
    return; c125---> O(1) return del if
  }
```

```
// Si es el primer producto, actualizamos el puntero de la cabeza
  if (opcion == 1) c126---> O(1) cabecera del if, compara numeros
  {


    Nodo *temp = head; c127---> O(1) asigna un nodo a una variable temporal
    head = head->siguiente; c128---> O(1) actualiza el puntero
    delete temp; c129---> O(1) elimina la variable temporal
    cout << "\n\nProducto eliminado.\n\n"; c130---> O(1) muestra un mensaje
    return; c131---> O(1) return del if
  }


  // Recorremos la lista hasta el producto anterior al seleccionado
  actual = head; c132---> O(1) incia la variable de la posicion actual del pointer
  for (int i = 1; i < opcion - 1; ++i) c133---> O(n) recorre los elementos hasta
encontrar el nodo a eliminar, en el peor de los casos seria n
  {


    actual = actual->siguiente; c134---> O(1) actualiza la posicion del nodo actual
  }


  // Eliminamos el nodo seleccionado
  Nodo *temp = actual->siguiente; c135---> O(1) crea un nodo temporal
  actual->siguiente = temp->siguiente; c136---> O(1) mueve el nodo a la siguiente
posicion
  delete temp; c137---> O(1) elimina el nodo temporal


  cout << "\n\nProducto eliminado.\n\n"; c136---> O(1) muestra un mensaje
}
```

Análisis de la función reabastecer productos

```
// Funcion para mostrar los productos que necesitan reabastecimiento (menos de
kStockMinimo unidades)
void reabastecer_productos(Nodo *head) c137---> O(1) declaraion de la funcion
{


  cout << "\n\nProductos que necesitan reabastecimiento:\n"; c138---> O(1) muestra un
mensaje
  Nodo *actual = head; c139---> O(1) declara el nodo a usar


  // Recorremos la lista y mostramos los productos con stock menor a kStockMinimo
  while (actual) c140---> O(n) recorre toda la lista, depende de n
  {
    if (actual->cantidad < kStockMinimo) c141---> O(1) cabecera del if, compara
numeros
    {
      cout << actual->nombre << ": " << actual->cantidad << " unidades\n"; c142--->
O(1) muestra un mensaje
    }
    actual = actual->siguiente; c143---> O(1) actualiza el nodo
  }
  cout << "\n"; c144---> O(1) muestra un mensaje (un espacio?)
}
```

Análisis de la función mostrar productos

```
  // Funcion para mostrar todo el inventario
  void mostrar_productos(Nodo *head) c145---> O(1) declaracion de la funcion
  {


    cout << "\nInventario actual:\n\n"; c146---> O(1) muestra un mensaje
    Nodo *actual = head; c147---> O(1) declara el nodo actual


    // Recorremos la lista y mostramos todos los productos
    while (actual) c148---> O(n) recorre toda la lista para mostrarla
    {
      cout << "Producto: " << actual->nombre << ", Cantidad: " << actual->cantidad
  << "\n"; c149---> O(1) muestra un mensaje
      actual = actual->siguiente; c150---> O(1) actualiza el nodo
    }
    cout << "\n"; c151---> O(1) muestra un mensaje (un espacio?)
  }
```

Análisis de la función menos stock

```
// Funcion para mostrar los n productos con menor stock
void menos_stock(Nodo *head, int n) c152---> O(1) declaracion de la funcion
{
  if (!head) c153---> O(1) cabecera del if, verifica si se cumple la condicion
  {
    cout << "\n\nEl inventario está vacío.\n\n"; c154---> O(1) muestra un mensaje
    return; c155---> O(1) return del if
  }


  // Contar el numero total de productos en la lista enlazada
  int total_productos = 0;     c156---> O(1) declaracion de una variable
  Nodo *temp = head; c157---> O(1) declaracion de nodo temporal
  while (temp) c158---> O(n) recorre todo el nodo para ver la cantidad en stock
  {
    total_productos++; c159---> O(1) es una operacion de suma
    temp = temp->siguiente; c160---> O(1) actualiza el nodo
  }


  // Si n es mayor que el numero total de productos, ajustar n al total
  if (n > total_productos) c161---> O(1) cabecera del if, es una comparacion
  {
    n = total_productos; c162---> O(1) asignacion de valor a una variable
  }


  // Ordena los productos por cantidad de stock usando una el algoritmo de burbuja
  for (int i = 0; i < total_productos - 1; i++) c163---> O(n²) un for con otro for
anidado
  {
    Nodo *actual = head; c164---> O(1) declara el nodo actual
    Nodo *siguiente = actual->siguiente; c165---> O(1) actualiza el nodo
    for (int j = 0; j < total_productos - i - 1; j++) c166---> O(n) for anidado, depende
de n (cantidad de productos)
    {
      if (actual->cantidad > siguiente->cantidad) c167---> O(1) cabecera del if,
comparacion de valores
      {
        // Intercambiaa los valores de nombre y cantidad entre actual y siguiente
        swap(actual->nombre, siguiente->nombre); c168---> O(1) cambia de posicion el
nombre del producto
        swap(actual->cantidad, siguiente->cantidad); c169---> O(1) cambia de posicion la
cantidad del producto
      }
      actual = siguiente; c170---> O(1) actualiza el nodo
      siguiente = siguiente->siguiente; c180---> O(1) actualiza el nodo
    }
  }
```

```
  // Mostrar los primeros n productos con menor stock
  cout << "\n\nLos " << n << " productos con menor stock son:\n"; c181---> O(1)
muestra un mensaje
  temp = head; c182---> O(1) crea un nodo temporal
  for (int i = 0; i < n && temp; i++) c183---> O(n) recorre la lista de productos
con menor stock, en el peor de los casos es n
  {
    cout << temp->nombre << " (Cantidad: " << temp->cantidad << ")\n"; c184--->
O(1) muestra un mensaje
    temp = temp->siguiente; c185---> O(1) actualiza el nodo temporal
  }
  cout << "\n"; c186---> O(1) muestra un mensaje (un espacio?)
}
```

Análisis de la función liberar memoria

```
   // Funcion para liberar la memoria de toda la lista enlazada
 void liberar_memoria(Nodo *&head) c187---> O(1) Declaracion de la funcion
 {
   while (head) c188---> O(n) recorre todos los nodos de la lista para eliminarlos
   {
     Nodo *temp = head; c189---> O(1) declara de nodo temporal
     head = head->siguiente; c190---> O(1) actualiza el nodo
     delete temp; c200---> O(1) elimina el nodo temporal
   }
 }
```

Análisis de la función main

```
int main(int argc, char *argv[])c13 --> 0(1) Declaracion de funcion main
{
  // Variables para almacenar la opcion del menu y los datos del producto
  int opcion, cantidad, numProductos;c14 --> 0(1) Declaracion de variables int

  string nombre; c13 --> 0(1) Declaracion de variable string


  // Puntero inicial del inventario (lista enlazada vacia al principio)
  Nodo *inventario = nullptr; c15 --> Inicializacion de puntero


  // Bucle principal del menu
  while (true) c16 --> 0(1) bucle
  {
    // Mostramos las opciones disponibles para gestionar el inventario
    cout << "\n========== Menu ==========\n\n";c17 --> 0(1) flujo de salida
    cout << "1. Agregar producto\n"; c18 --> 0(1) flujo de salida estandar
    cout << "2. Modificar producto\n"; c19 --> 0(1) flujo de salida estandar
    cout << "3. Productos a rebastecer\n"; c20 --> 0(1) flujo de salida estandar
    cout << "4. Mostrar inventario\n"; c21 --> 0(1) flujo de salida estandar
    cout << "5. Eliminar producto\n"; c22 --> 0(1) flujo de salida estandar
    cout << "6. Productos con menor stock\n"; c23 --> 0(1) flujo de salida
estándar
    cout << "7. Salir"; c24 --> 0(1) flujo de salida estandar
    cout << "\n\nSeleccione una opcion: "; c25 --> 0(1) flujo de salida estandar
    cin >> opcion; c26 --> 0(1) flujo de entrada esttandar
    cin.ignore();  c27 --> 0(1) funcion para ignorar salto de linea
```

```
switch (opcion) c28 --> 0(1) expresion de control


    {
    case 1: c29 --> 0(1) etiqueta de caso
      // Pedimos al usuario que ingrese los detalles del nuevo producto
      cout << "\nIngrese el nombre del producto: ";c30 --> 0(1) flujo de salida
      getline(cin, nombre); c31 --> 0(1) extraccion de linea
      cout << "\nIngrese la cantidad en stock: ";c32 --> 0(1) flujo de salida
      cin >> cantidad;c33 --> 0(1) flujo de entrada
      agregar_producto(inventario, nombre, cantidad); // Llamamos a la funcion para
agregar el producto c34 ---> 0(1) invocación de funcion
      break; c35 --> 0(1) instruccion de interrupcion
    case 2: c36 --> 0(1) etiqueta de caso
      actualizar_producto(inventario); // Llamamos a la funcion para modificar un
vproducto existente c37 ---> 0(1) invocación de funcion
      break;c38 --> 0(1) instruccion de interrupcion
    case 3:c39 --> 0(1) etiqueta de caso
      reabastecer_productos(inventario); // Mostramos los productos que necesitan
reabastecimientoc40 ---> 0(1) invocación de funcion
      break;c41 --> 0(1) instruccion de interrupcion
    case 4: c42 --> 0(1) etiqueta de caso
      mostrar_productos(inventario); // Mostramos el inventario completoc43 ---> 0(1)
invocación de funcion
      break; c44 --> 0(1) instruccion de interrupcion
    case 5: c45 --> 0(1) etiqueta de caso
      eliminar_producto(inventario); // Permitimos al usuario eliminar un producto c46
---> 0(1) invocación de funcion
      break;c47 --> 0(1) instruccion de interrupcion
    case 6:c48 --> 0(1) etiqueta de caso
      cout << "\nIngrese el numero de productos a mostrar: "; c49 --> 0(1) flujo de
salida
      cin >> numProductos; c50 --> 0(1) flujo de entrada
      menos_stock(inventario, numProductos); // Llamamos a la funcion para mostrar n
productos con menor stock c51 ---> 0(1) invocación de funcion
      break;c52 --> 0(1) instruccion de interrupcion

    case 7:c53 --> 0(1) etiqueta de caso


      liberar_memoria(inventario); // Liberamos la memoria de todos los productos antes
de salir c54 ---> 0(1) invocación de funcion
      cout << "\nSaliendo del sistema...\n\n"; c55 --> 0(1)flujo de salida
      return 0;c56 --> 0(1) instruccion de retorno
    default: c57 --> 0(1) caso default
      // Si el usuario ingresa una opcion invalida, mostramos un mensaje
      cout << "\nOpcion no valida, intente de nuevo.\n\n";c58 --> 0(1) flujo de salida
    }
  }
  return 0;c59 --> 0(1) instruccion de retorno
}
```

Conclusiones

Para el propósito del proyecto, se ha realizado un análisis de cada línea de código para determinar el orden de magnitud para el peor de los casos. en el análisis se puede observar que la gran mayoría de las líneas de código tienen un orden de magnitud **O(1)**, es decir que tienen un costo que se puede considerar irrelevante en términos de tiempo y costo de hardware, sin embargo, el sistema hace uso de ciertos algoritmos que no se pueden considerar **O(1)**.

En el caso de algunos consiste en el uso de un ***while*** o de un ***for***, los cuales dependen de la cantidad de productos en el sistema, lo que se puede interpretar como **n**, por lo que se puede decir que estos tienen un orden de magnitud **O(n)**.

A grandes rasgos, se puede concluir que el sistema tiene un orden de magnitud **O(n^2)**, esto es debido a que la opción 6 del sistema hace uso del algoritmo de ordenamiento por burbuja, el cual consiste en un ***for*** dentro de otro ***for***, y por el tipo de sistema, estos ***for*** siempre van a depender de la cantidad de productos que se manejan en el sistema, es decir, **n**, esto quiere decir que el **for** interno se realiza **n** veces por cada ronda del **for** externo, lo que se traduce como **n*n**, que como se mencionó anteriormente resulta en **n^2**, de ahi el resultado del orden de magnitud **O(n^2)**.