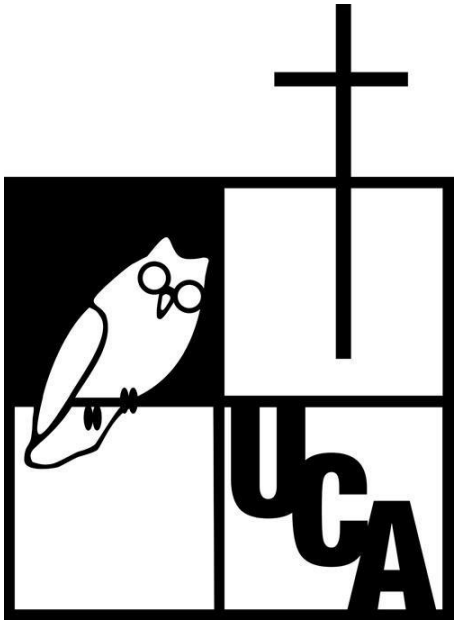
UNIVERSIDAD CENTROAMERICANA
“JOSÉ SIMEON CAÑAS”



ANALISIS DE ALGORITMOS

TALLER 1

INTEGRANTES:

Katherin Gabriela Pérez Baires - 00141621

José Alejandro Chávez Guardado - 00149320

Andrés Felipe Cardona Duarte - 00037820

09 de septiembre del 2024, Antiguo Cuscatlán, El Salvador

```python
class Producto: O(1)
    def __init__(self, nombre, stock): O(1)
        self.nombre = nombre O(1)
        self.stock = stock O(1)

    def __repr__(self): O(1)
        return f"Producto: {self.nombre}, Stock: {self.stock}" O(1)

def menu_principal(inventario): O(m · n²)
    while True: O(m)
        print("\nMenu Principal:") C1 · O(1)
        print("a. Ver todos los productos") C2 · O(1)
        print("b. Ver productos con bajo stock") C3 · O(1)
        print("c. Agregar un producto") C4 · O(1)
        print("d. Eliminar un producto") C5 · O(1)
        print("e. Editar un producto") C6 · O(1)
        print("f. Salir") C7 · O(1)

        eleccion = input("Seleccione una opción: ") C8 · O(1)

        if eleccion == "a": C9 · O(1)
            inventario.mostrar_productos() O(n²)
        elif eleccion == "b": C11 · O(1)
            menu_bajo_stock(inventario) O(n²)
        elif eleccion == "c": C13 · O(1)
            nombre = input("Ingrese el nombre del producto: ") C14 · O(1)
            stock = int(input("Ingrese el stock del producto: ")) C15 · O(1)
            inventario.agregar_producto(nombre, stock) C16 · O(1)
        elif eleccion == "d": C17 · O(1)
            nombre = input("Ingrese el nombre del producto a eliminar: ") C18 · O(1)
            inventario.eliminar_producto(nombre) O(n)
        elif eleccion == "e": C20 · O(1)
            nombre_antiguo = input("Ingrese el nombre del producto a editar: ") C21 · O(1)
            nuevo_nombre = input("Ingrese el nuevo nombre del producto: ") C22 · O(1)
            nuevo_stock = int(input("Ingrese el nuevo stock del producto: ")) C23 · O(1)
            inventario.actualizar_producto(nombre_antiguo, nuevo_nombre, nuevo_stock) O(n)
        elif eleccion == "f": C25 · O(1)
            print("Saliendo del programa...") C26 · O(1)
            break C27 · O(1)
        else: C28 · O(1)
            print("Opción no válida, por favor intente de nuevo.") C29 · O(1)
```

Calculos:

T(n) = C1 * O(1)*m + C2 * O(1)*m + C3 * O(1)*m + C4 * O(1)*m + C5 * O(1)*m + C6 * O(1)*m + C7 * O(1)*m
+ C8 * O(1)*m + C9 * O(1)*m + C10 * O(m) * O(n^2) + C11 * O(1) + C12 * O(m) * O(n^2) + C13 * O(1)*m + C14 * O(1)*m
+ C15 * O(1)*m + C16 * O(1)*m + C17 * O(1)*m + C18 * O(1)*m + C19 * O(n)*m + C20 * O(1)*m + C21 * O(1)*m
+ C22 * O(1)*m + C23 * O(1)*m + C24 * O(n)*m + C25 * O(1) *m+ C26 * O(1)*m + C27 * O(1)*m + C28 * O(1)*m
+ C29 * O(1)*m

T(n) = (C1 + C2 + ... + C29) * m + (C10 + C12) (O(m) * O(n^2)) + (C19 + C24) *(n * m)

el tiempo total sera:

T(n) = O(n^2 * m) + O(n * m) + (C1 + C2 + ... + C29) * m
T(n) = O(n^2 * m), ya que es el término más grande.

```python
class Inventario: C18 · 1
    def __init__(self): C19 · 1
        self.productos = [] C20 · 1
        productos_iniciales = [
            ("Peras", 20), ("Miel", 5), ("Queso panela", 15),
            ("Limones", 7), ("Leche", 8), ("Remolacha", 12),
            ("Jocote de corona", 3), ("Huevos", 9), ("Chocolate", 14),
            ("Guayaba", 11)
        ] C21 · 1
        for nombre, stock in productos_iniciales: C22 · n + 1
            self.agregar_producto(nombre, stock) C23 · n
    def agregar_producto(self, nombre, stock): C24 · 1
        producto = Producto(nombre, stock) C25 · 1
        self.productos.append(producto) C26 · 1
        print(f"Producto agregado: {producto}") C27 · 1

    def bubble_sort(self): C28 · 1
        n = len(self.productos) C29 · 1
        for i in range(n): C30 · n
            for j in range(0, n-i-1): C31 · n − i + 1
                if self.productos[j].stock > self.productos[j+1].stock: C32 · O(1) · n − i
                    self.productos[j], self.productos[j+1] = self.productos[j+1], self.productos[j] C33 · O(1) · max(0,1) + 1 · n − i

    def mostrar_productos(self): C34 · 1
        if not self.productos: C35 · 1
            print("No hay productos en el inventario.") C36 · 1
        else: C37 · 1
            self.bubble_sort() O(n²)
            for producto in self.productos: C38 · n + 1
                print(producto) C39 · n

    def eliminar_producto(self, nombre): C40 · 1
        for producto in self.productos: C41 · n + 1
            if producto.nombre == nombre: C42 · 1
                self.productos.remove(producto) C43 · 1
                print(f"Producto eliminado: {producto}") C44 · 1
                return C45 · 1
        print("Producto no encontrado.") C46 · 1

    def actualizar_producto(self, nombre, nuevo_nombre, nuevo_stock): C47 · 1
        for producto in self.productos: C48 · n + 1
            if producto.nombre == nombre: C49 · n
                producto.nombre = nuevo_nombre C50 · n
                producto.stock = nuevo_stock C51 · n
                print(f"Producto actualizado: {producto}") C52 · n
                return C53 · n
        print("Producto no encontrado.") C54 · n

    def productos_con_bajo_stock(self): C55 · 1
        self.bubble_sort() C56 · O(n²)
        return [producto for producto in self.productos if producto.stock < 10] C57 · n · [max(0,1) + 1]
```

O(1)

O(1)

O(n)

O(n²)

(n²) + max(1,1) + 1 · (n+1) → O(n²)

n · [max(0,1) + 1] → O(n)

O(n)

T(n)=C18∗1+C19∗1+C20∗1+C21∗1+C21∗1+C21∗1+C21∗1+C21∗1+
(C22∗n+1)+C23∗n+C24∗1+C25∗1+C26∗1+C27∗1+C28∗1+C29∗1+C30∗n+C31∗(n−i+1)+C32∗O(1)∗(n−i)+C33∗O(1)∗max(0,1)∗n-1+C35∗1+C36∗1+C37∗1+C38+
(C39∗n+1)+C40∗n+C41∗1+(C42*n+1)+C43∗1+C44∗1+C45∗1+C46∗1+C47∗1+C48∗1+
(C49∗n+1)+C50∗n+C51∗n+C52∗n+C53∗n+C54∗n+C55∗n+C56∗1+C57∗O(n2)+C58∗n+[max(0,1)+1]

Teniendo en cuenta el algoritmo bubble sort desde c28 hasta c33 podemos determinar:

T(n)=C18+C19+C20+C21+(C22∗n+1)+C23∗n+C24+C25+C26+C27+O(n2)+C29+C30∗n+C31∗(n−i+1)+C32∗O(1)∗(n−i)
+C35+C36+C37+C38+C39∗n+C40∗n+C41+C42∗n+C43+C44+C45+C46+C47+C48+C49∗n+C50∗n+C51∗n+C52∗n+C53∗n+C54∗n+C55∗n+C56+O(n2)+C58∗n+1

T(n)=O(n2)+(C22+C23+C30+C39+C40+C42+C49+C50+C51+C52+C53+C54+C55+C58)∗n+
(C18+C19+C20+5∗C21+C24+C25+C26+C27+C29+C35+C36+C37+C38+C41+C43+C44+C45+C46+C47+C48+C56+1)

T(n)=O(n2)+C∗n+K
T(n)=O(n2)+O(n)+O(1)
T(n)=O(n2)

```python
def menu_bajo_stock(inventario): O(n²)
    productos_bajos = inventario.productos_con_bajo_stock() C1
    if not productos_bajos:
        print("No hay productos con bajo stock.") C2 · max(0,1)
        return C3 · max(0,1)

    print("\nProductos con bajo stock:") C4
    cantidad_mostrar = int(input("¿Cuántos productos desea ver?: ")) C5
    for producto in productos_bajos[:cantidad_mostrar]: C6 · n
        print(producto) C7 · n

    if input("¿Desea reabastecer algún producto? (s/n): ").lower() == 's': C8
        nombre_producto = input("Ingrese el nombre del producto a reabastecer: ") C9 · max(0,1)
        cantidad_reabastecer = int(input("Ingrese la cantidad de stock a agregar: ")) C10 · max(0,1)
        for producto in productos_bajos: C11 · n · max(0,1)
            if producto.nombre == nombre_producto: C12 · n · max(0,1)
                producto.stock += cantidad_reabastecer C13 · max(0,1) · max(0,1)
                print(f"Producto {producto.nombre} reabastecido. Nuevo stock: {producto.stock}") C14 · max(0,1) · max(0,1)
                break C15 · max(0,1) · max(0,1)
        else: C16 · max(0,1) · max(0,1)
            print("Producto no encontrado.") C17 · max(0,1) · max(0,1)


inventario = Inventario() O(1)
menu_principal(inventario) O(n²)
```

T(n) = C1 + c2*max(0,1) + C3*max(0,1) + C4 + C5 + C6*(n+1) + C7*n + C8 + C9*max(0,1) + C10*max(0,1) + C11*(n+1)*max(0,1) + C12*n*max(0,1) + C13*max(0,1)*max(0,1) + C14*max(0,1)*max(0,1) + C15*max(0,1)*max(0,1) + C16*max(0,1)*max(0,1) + C17*max(0,1)*max(0,1)

T(n) = C1 + C4 + C5 + *n + C6+ C7*n + C8 + C9 + C10 + C11*n + C11 + C12*n + C13 + C14 + C15
T(n) = (C6+ C7 + C11 + C12)n + (C1 + C4+ C5 + C6 + C8 + C9 + C10 + C11 +C13 + C14 + C15)
T(n) = An + B
T(n) = O(n)

Sumando con la de productos_bajos = inventario.productos_con_bajo_stock()T(n)

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

Conclusiones:

Conclusiones de la clase Inventario: Desde la constante C18 hasta C21, el código es constante, es decir, solo se ejecuta una vez en el programa. Por lo tanto, esta sección del array en el método _init_ es constante. Desde la línea C22 a C23 tenemos un for cuyo funcionamiento es lineal. Las líneas C24 a C27 corresponden a una función lineal. Las líneas C28 a C33 implementan el conocido algoritmo de ordenamiento Bubble Sort, siendo este un algoritmo cuadrático. En las líneas C35 a C40 se encuentra la función mostrar productos, en la cual se llama a la función Bubble Sort. Como mencionamos anteriormente, este es un algoritmo cuadrático, por lo tanto, la función mostrar productos tiene una complejidad cuadrática. Desde la línea C41 a la C47 se encuentra la función eliminar producto, la cual es lineal. En las líneas C48 a C55 se encuentra la función actualizar producto, que tiene un comportamiento lineal. En el caso de la línea C57, se llama nuevamente a la función Bubble Sort, y la línea C58 contiene un if dentro de un for, lo cual nos indica que esta línea es lineal.

Al analizar todas las líneas del código, podemos concluir que su complejidad temporal en el peor de los casos es $O(m \cdot n^2)$ donde:

- m representa el número de iteraciones del bucle while, que es, el número de veces que el usuario interactúa con el menú antes llegar a la condición de paro.
- N^2 representa el número de productos en el inventario.

Esto se debe a que la operación con la mayor complejidad dentro del bucle while ocurre en la opción "a", que implica recorrer todos los productos del inventario, lo que tiene una complejidad $O(n^2)$ Por lo tanto, si esta opción se selecciona en cada iteración del bucle, el tiempo total de ejecución sería $O(m \cdot n^2)$

Este código no es el mejor, debido a que en el peor de los casos el tiempo de ejecución puede crecer a medida que crece el número de productos n y el número de interacciones m.