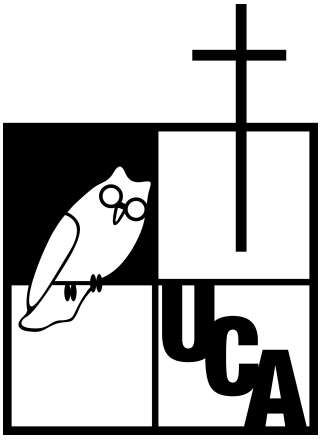
UNIVERSIDAD CENTROAMERICANA

JOSÉ SIMEÓN CAÑAS



TALLER 1.

ANÁLISIS DE ORDEN DE MAGNITUD.

INTEGRANTES:

MAURICIO ALEJANDRO CONTRERAS MONTOYA 00000422

JOSÉ RAÚL MORÁN HENRÍQUEZ 00135321

DIEGO FERNANDO ALVARADO SIBRIÁN 00127522

6 de Septiembre de 2024, Antiguo Cuscatlán, El Salvador.

SISTEMA DE ORGANIZACIÓN DE PRODUCTOS.

```
#Add Product function
def addProduct(data):
     products[data[0]]=data[1];

#Delete product function
def removeProduct(key):
  products.pop(key);


#Update stock function
def updateStock(data):
   products[data[0]]=int(products.get(data[0]))+data[1];

def showProducts():
  for key,value in products.items():
    print(f'Nombre:{key},Stock:{value}')

#Verify stock
def verifyStock():
  low_stock={}
  for i in products:
    temp=int(products.get(i))
    if temp<10:
          low_stock[i]=temp
    print(low_stock)
  orderLowStcok(low_stock) #order low stock  call function

# Order low stock products function
def orderLowStcok(data):
  stock_ordered = sorted(data.items(), key=lambda item: item[1])
  n=int(input('Cuantos productos con bajo stock desea ver?'))
  for i in range(n):
   print(f'{stock_ordered[i][0]}:{stock_ordered[i][1]}')

#Register function
def register():
   registro=input('Formatos\n Para agregar o actualizar producto nombre:stock\nPara eliminar nombre\n').split(':');
   return registro;

"""
product{
  name:stock
}"""
products={}

while True:
  new=input('1)Agregar productos \n2)Actualizar Stock \n3)Eliminar producto \n4)Verificar Stock \n5)Ver
productos\n6)Salir\n\n ')
  if new=='1':
      while True :
            info = register();        #Register call function add format
            if info[0]!='salir':
               addProduct(info);      #Add call function
            else: break;
  elif new=='2':
      up_data=register();            #Register call function update format
      updateStock(up_data);          #Update call function
  elif new=='3':
      product_name=register();       #Register call function delete format
      removeProduct(product_name),   #Delete call function
  elif new=='4':
      verifyStock();                 #verify stock call function
  elif new=='5':
      showProducts();
  elif new=='6':
      break ;
  else:
    print('Opcion no valida \n\n');
```

ANÁLISIS DEL CÓDIGO

```
def addProduct(data):
```

Description: Se define la función, espera una lista para “data”.
Action: La función se encargará de agregar o actualizar un producto al diccionario.
Order: O(1)

```
products[data[0]]=data[1];
```

Description: data[0] es el nombre del producto y data[1] es el stock del producto.
Action: Inserta o actualiza el producto en el diccionario “products”, la clave es el nombre del producto (data[0]) y el valor es el stock (data[1]).
Order: O(1).

```
def removeProduct(key):
```

Description: Se define la función que recibe el parámetro “key”, en este caso es el nombre del producto.
Action: La función eliminará los productos del diccionario según la clave que se proporcione.
Order: O(1).

```
products.pop(key);
```

Description: “key” es el nombre del producto que se desea eliminar.
Action: Elimina el producto utilizando ***pop()***.
Order: O(1).

```
def updateStock(data):
```

Description: Se define la función que recibe una lista como parámetro.
Action: Esta función actualizará el stock de un producto.
Order: O(1).

```
products[data[0]]=int(products.get(data[0]))+data[1];
```

Description: data[0] es el nombre del producto y data[1] es la cantidad a sumar al stock.
Action: Utiliza el método get() para obtener el valor del stock actual del

producto y lo suma a data[1]. Si el producto no existe devuelve **none**.
Order: O(1).

```
def showProducts():
```

Description: Se define la función sin entrada.
Action: Esta función mostrará todos los productos y el stock.
Order: O(1).

```
for key,value in products.items():
```

Description: Itera sobre el diccionario de productos.
Action: Recorre cada producto en el diccionario utilizando items() obteniendo el nombre y el stock.
Order: O(n).

```
print(f'Nombre:{key},Stock:{value}')
```

Description: key es el nombre del producto y value es la cantidad en stock.
Action: Muestra el nombre y la cantidad de stock como se muestra en el formato.
Order: O(1).

```
def verifyStock():
```

Description: Se define la función sin parámetros.
Action: Este verifica que producto tiene un stock menor a 10.
Order: O(1)

```
low_stock={}
```

Action: Se crea un diccionario vacío que almacenará los productos con bajo stock.
Order: O(1).

```
for i in products:
```

Description: Itera con las llaves del diccionario de products.
Action: Entra al diccionario de productos y los examina uno por uno.
Order: O(n)

```
temp = int(products.get(i))
```
Description: i será el nombre del producto.
Action: Toma el stock del producto que tiene y lo convierte en un int (número entero).
Order: O(1)

```
if temp < 10:
```
Description: temp es el stock del producto.
Action: En la condición verifica si el stock es menor a 10.
Order: O(1)

```
low_stock[i] = temp
```
Description: i es el nombre del producto que habíamos declarado y temp es el stock del producto.
Action: La condición es que si el stock es menor a 10 entonces el producto se añade al diccionario de low_stock.
Order: O(1)

```
print(low_stock)
```
Description: Este es un diccionario llamado low_stock, que contiene los productos que tienen bajo stock
Action: Imprime el contenido que tiene el diccionario low_stock
Order: O(n)

```
orderLowStcok(low_stock)
```
Description: El diccionario low_stock es la entrada
Action: Llama a la función orderLowStcok y le pasa el diccionario de low_stock como argumento.
Order: O(1)

```
def orderLowStcok(data):
```
Description: Toma como parámetro data, al diccionario low_stock.
Action: Se define la función ordeLowStcok
Order: O(1)

```
stock_ordered = sorted(data.items(), key=lambda item: item[1])
```
Description: Data que es el diccionario creado que se le asignó low_stock devuelve una vista de pares clave y valor del diccionario.
Action: En data se hace un ordenamiento basado en el valor del stock.
Order: O(k*log(k))

```
n=int(input('Cuantos productos con bajo stock desea ver?'))
```
Description: El usuario ingresa un número de tipo entero a través del input.
Action: El programa solicita al usuario cuando productos con bajo stock desea ver.
Order: O(1)

```
for i in range(n):
```
Description: El valor de n es donde se guardó el número de productos de bajo stock quiere ver el usuario.
Action: Crea un bucle que itera n veces para mostrar los productos.
Order: O(n)

```
print(f'{stock_ordered[i][0]}:{stock_ordered[i][1]}')
```
Description: El producto con bajo stock se mostrará en la posición i en la lista ordenada.
Action: Imprime la cantidad y el nombre del producto que está en la posición i.
Order: O(1)

```
def register():
```
Description: Se define una función llamada register, sin ningún tipo de parámetros.
Action: Esta función solicita la entrada del usuario.
Order: O(1).

```
registro = input('Formatos\n Para agregar o actualizar producto nombre:stock\nPara eliminar nombre\n').split(':');
```
Description: El usuario da una cadena de texto que tiene el nombre del producto y su stock.
Action: Muestra un mensaje al usuario donde debe ingresar unos datos.
Order: O(1)

```
return registro;
```
Description: registro es una lista con el nombre del producto y su stock.
Action: Devuelve la lista llamada registro.
Order: O(1).

```
products = {}
```
Description: No hay entrada en esta línea.
Action: Se crea un diccionario products que en este caso está vacío.
Order: O(1).

```
while True:
```
Description: No hay ningún tipo de entrada en esta línea.
Action: Se crea un bucle que se repetirá hasta que el usuario quiera salir.
Order: No se puede dar un orden de magnitud debido a que no se sabe la cantidad de veces que el usuario va a usar ese menú, por lo tanto no se toma en cuenta para el orden de magnitud.

```
 new = input('1)Agregar productos \n2)Actualizar Stock
\n3)Eliminar producto \n4)Verificar Stock \n5)Ver
productos\n6)Salir\n\n ')
```
Description: El usuario ingresa una opción del menú.
Action: Le muestra el menú al usuario y recibe lo que seleccionó.
Order: O(1).

```
if new == '1':
```
Description: Se da entrada a la variable new donde se realiza la elección del usuario.
Action: En la condición verifica si el usuario seleccionó la opción para agregar productos.
Order: O(1)

```
while True:
```
Description: No hay ningún tipo de entrada.
Action: Hace un bucle para que el usuario agregue la cantidad de productos que el quiera, hasta que el decida salir.
Order: O(n).

```
info = register();
```
Description: Llama a la función a register para obtener los datos del usuario.
Action: Solicita al usuario que ingrese nombre y stock del producto.
Order: O(1)

```
if info[0] != 'salir':
```
Description: Verifica el primer elemento de la lista info.
Action: Si el usuario no le da salida, continua adición del producto.
Order: O(1).

```
addProduct(info);
```
Description: Pasa una lista llamada info a una función llamada addProduct, que es la agrega el producto al diccionario.
Action: Llama a la función addProduct para agregarla al diccionario llamado products.
Order: O(1)

```
else: break;
```
Description: No hay entrada de datos.
Action: Si el valor de info es salir, se sale del bucle que habíamos abierto.
Order: O(1)

```
elif new == '2':
```
Description: La variable new tiene otra elección del usuario.
Action: Verifica si el usuario seleccionó la opción 2 para poder actualizar el producto.
Order: O(1)

```
up_data = register();
```
Description: Llama a la función register
Action: Solicita que el usuario ingrese el nombre del producto y la cantidad en stock para así actualizarlo.
Order: O(1).

```
updateStock(up_data);
```
Description: Pasa la lista up_data a la función donde se actualiza el stock updateStock.

Action: Llama la función updateStock para actualizarlo en el diccionario de products.
Order: O(1)

```
elif new == '3':
```

Description: Entrada de la variable new para la elección del usuario.
Action: Verifica si el usuario seleccionó la opción 3 para eliminar un producto.
Order: O(1)

```
product_name = register();
```

Description: Llama a la función register para así obtener el nombre del producto que se va a eliminar.
Action: Se le solicita al usuario que ingrese el nombre del producto que desea eliminar.
Order: O(1)

```
removeProduct(product_name);
```

Description: Pasa la lista product_name a la función para eliminar removeProduct.
Action: Llama a la función removeProduct para eliminar el producto desde el diccionario products.
Order: O(1)

```
elif new=='4':
```

Description: new contiene la “elección del usuario”, en este caso un 4.
Action: Verifica si el usuario seleccionó la opción 4 correspondiente a verificar el stock de productos.
Order: O(1).

```
verifyStock();
```

Action: Llama a la función para verificar los productos.
Order: O(n).

```
elif new=='5':
```

Description: new contiene la “elección del usuario”, en este caso un 5.
Action: Verifica si el usuario seleccionó la opción 5 que corresponde a mostrar los productos.

Order: O(1).

```
showProducts();
```

Action: Llama a la función para mostrar los productos y el stock.
Order: O(n).

```
elif new=='6':
```

Description: new contiene la “elección del usuario”, en este caso un 6.
Action: Verifica si el usuario seleccionó la opción 6 que corresponde a salir del programa.
Order: O(1).

```
break ;
```

Action: Finaliza el bucle.
Order: O(1).

```
 else:
```

Action: Si el usuario no proporciona una opción contenida en el menú que sea válida se ejecuta esta condición.
Order: O(n).

```
print('Opcion no valida \n\n');
```

Action: Imprime el mensaje indicando su contenido.
Order: O(1).

ANÁLISIS DE MAGNITUD GENERAL

```
def addProduct(data):
    products[data[0]] = data[1]
```

Order: O(1)
Description: Es O(1) debido a que las dos líneas de código son funciones constantes.

```
def removeProduct(key):
  products.pop(key);
```

Order: O(1)
Description: Es O(1) debido a que las dos líneas de código son funciones constantes.

```
def updateStock(data):
   products[data[0]]=int(products.get(data[0]))+data[1];
```

Order: O(1)
Description: Es O(1) debido a que las dos líneas de código son funciones constantes.

```
def showProducts():
  for key,value in products.items():
    print(f'Nombre:{key},Stock:{value}')
```

Order: O(n)
Description: Es O(n) debido a que la función itera sobre todos los producto y realiza una impresión para cada uno y el for que por sí solo es O(n).

```
def verifyStock():
  low_stock={}
  for i in products:
    temp=int(products.get(i))
    if temp<10:
          low_stock[i]=temp
    print(low_stock)
  orderLowStcok(low_stock) #order low stock  call function
```

Order: O(n)
Description: Esto se debe al bucle que pasa por todos los productos para

verificar su stock.

```
def orderLowStcok(data):
  stock_ordered = sorted(data.items(), key=lambda item: item[1])
  n=int(input('Cuantos productos con bajo stock desea ver?'))
  for i in range(n):
   print(f'{stock_ordered[i][0]}:{stock_ordered[i][1]}')
```

Order: O(k*log(k))
Description: n es el número de productos con bajo stock del diccionario data. El bucle for que tenía se ejecuta después de que se ordena, entonces no afecta el orden de la magnitud total.

```
def register():
   registro=input('Formatos\n Para agregar o actualizar producto
nombre:stock\nPara eliminar nombre\n').split(':');
   return registro;

"""
product{
  name:stock
}"""
products={}
```

Order: O(m).
Description: Esto se debe a que la entrada del usuario y la operación de la cadena donde se divide, dependen de la longitud de la cadena de entrada.

```
    new = input('1)Agregar productos \n2)Actualizar Stock
\n3)Eliminar producto \n4)Verificar Stock \n5)Ver
productos\n6)Salir\n\n ')
    if new == '1':
        while True:
            info = register()  # Register call function
            if info[0] != 'salir':
                addProduct(info)  # Add call function
            else:
                break
    elif new == '2':
        up_data = register()  # Register call function
        updateStock(up_data)  # Update call function
```

```
    elif new == '3':
        product_name = register()  # Register call function
        removeProduct(product_name)  # Delete call function
    elif new == '4':
        verifyStock()  # Verify stock call function
    elif new == '5':
        showProducts()  # Show products call function
    elif new == '6':
        break
    else:
        print('Opcion no valida \n\n')
```

Order: O(n).
Description: Los productos que se agregan serían n productos, por lo tanto la relación de estos serían O(n).

ANÁLISIS COMPLETO DEL SISTEMA.

$$C_{iteraciones} = C_5 \cdot n + C_6 \cdot k \cdot log(k) + C_4 \cdot n$$
$$C_{total} = m \cdot (C_5 \cdot n + C_6 \cdot k \cdot log(k) + C_4 \cdot n)$$
$$C_{total} = m \cdot ((C_5 + C_4) \cdot n + C_6 \cdot k \cdot log(k))$$

$$C_{total} = m \cdot (n +\cdot k \cdot log(k))$$

Despues de realizar un analisis de linea por linea y función por función se llegó a esta respuesta, se tomaron en cuenta diferentes factores para poder tomar una decisión de porque esa respuesta, en cuando a la “m” representa el numero de interacciones que el usuario puede tener, la “es el número total de productos que se encuentran en el diccionario” y “k” es el la cantidad de productos con bajo stock. Las funciones como addProduct(), removeProduct(), y updateStock() tienen una complejidad de “O(1)”, ya que realizan operaciones constantes sobre el diccionario. Por lo tanto funciones como showProducts() y verifyStock() tienen una complejidad de “O(n)”, dado que recorren todos los productos que tiene el diccionario, ahora pasamos a un caso más complejo que es la función orderLowStock() el cual hace que en su total sea “O(k \ log(k))”, debido a que se realizó una búsqueda en el cual sorted hace que tenga esa magnitud esa parte del código. El bucle principal del pro ejecuta en función de las interacciones que tiene el usuario, por lo que esta respuesta depende mucho del usuario en este caso, el numero de productos, como las interacciones realizadas en el menú, resultando en la complejidad mencionada.