1. Introduction

→ Implementing sorting algorithms on data sets to order the products.
→Controlling the product data, the key operation list, inserting, updating, deleting, and searching, completing the quicksort and analyzing its performance metrics for sorting by price.

2. Snapshots of the running program

Data Load: The script begins with raw_product_data, that is a list of raw product information. Every line is divided into sub-dictionaries for products and each line is stored in the products list.

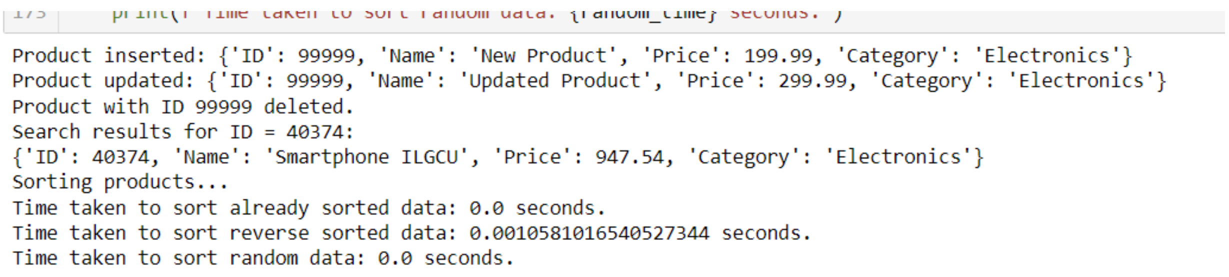
Insert: By utilizing the insert_product function, a new product is added to the frequently bought together products list.

Update: A product is updated (Its ID is given) by calling the update_product function.

Delete: To delete a product by its ID from the products list the delete_product function is used.

Search: Using the function search_product products can be searched by ID or Name. It prints the results of the search carried out.

Sort: For sorting the products based on price, the Quick Sort algorithm is deployed in the quick_sort function. Sorting time is measured for three scenarios: already sorted data, reverse sorted data, and random data.



```
    print(f"Time taken to sort random data: {random_time} seconds.")

Product inserted: {'ID': 99999, 'Name': 'New Product', 'Price': 199.99, 'Category': 'Electronics'}
Product updated: {'ID': 99999, 'Name': 'Updated Product', 'Price': 299.99, 'Category': 'Electronics'}
Product with ID 99999 deleted.
Search results for ID = 40374:
{'ID': 40374, 'Name': 'Smartphone ILGCU', 'Price': 947.54, 'Category': 'Electronics'}
Sorting products...
Time taken to sort already sorted data: 0.0 seconds.
Time taken to sort reverse sorted data: 0.0010581016540527344 seconds.
Time taken to sort random data: 0.0 seconds.
```

- **Insert:** The new product is successfully inserted and shown in the output and thus, it confirms the successful application of the SQL statements used.
- **Update:** An existing product has an ID of 99999 which is updated in the database with a new price and new name.
- **Delete:** The product with the ID 99999 is removed from the list.
- **Search:** Products with ID 40374 are located and presented.
- **Sorting:** It will be very fast for already sorted, reverse sorted and random data because Quick Sort is being used for sorting.

3. Complexity analysis report

→ **Best-case: O(n log n)** if the pivot chooses the list of elements in nearly half and half situations.

→ **Average-case: O(n log n).**

→ **Worst-case: O(n^2)** when the pivot is tried to be the smallest or the largest element.

4. Code implementation

https://github.com/ayshabilai/Assignment-one/commit/205ec3488cf068cb35dce59a87b81b60da0c3e52

```
##########################################################################

import time

# Product data initialization (raw product data as a list of strings)
raw_product_data = [
    "57353, Camera SBBHC, 546.88, Electronics",
    "40374, Smartphone ILGCU, 947.54, Electronics",
    "34863, Biography XPESK, 287.31, Books",
    "18086, Shirt ZQLTI, 439.07, Clothing",
    "16041, Jacket OTBKQ, 986.73, Clothing",
    "43566, Mystery COKPK, 836.57, Books",
    "69260, Toaster FODKJ, 867.6, Home & Kitchen",
    "30895, Knife Set KGFUF, 385.77, Home & Kitchen",
    "19897, Blender DPKLR, 488.62, Home & Kitchen",
    "87296, Skirt IRTZX, 261.08, Clothing",
    "68215, Laptop QLBQC, 404.21, Electronics",
    "68097, Camera SGSRZ, 36.39, Electronics",
    "26556, Novel METLI, 376.45, Books",
    "30483, Knife Set WRSZZ, 55.97, Home & Kitchen",
    "62422, Camera VFQWS, 382.69, Electronics",
    "22806, Smartwatch VVFNT, 203.55, Electronics",
    "24976, Pants YZMAK, 449.56, Clothing",
    "30631, Headphones JFGYQ, 115.08, Electronics",
    "27939, Textbook TWQKZ, 108.5, Books",
    "41355, Headphones JOUXM, 211.57, Electronics",
    "94162, Laptop WRJOZ, 956.53, Electronics",
    "28710, Dress FRSMO, 879.09, Clothing",
    "90291, Pants TIPUD, 853.38, Clothing",
    "20368, Shirt FQFPK, 83.19, Clothing",
    "68960, Blender OMDPS, 720.06, Home & Kitchen",
    "40852, Novel IRROY, 603.68, Books",
    "97895, Blender KSJHL, 123.25, Home & Kitchen",
    "96314, Cutting Board LUICX, 628.29, Home & Kitchen",
    "85719, Laptop GZORF, 641.33, Electronics",
    "98625, Mystery BOPTP, 160.68, Books",
    "66208, Blender GCZSK, 161.83, Home & Kitchen",
    "86128, Biography ASTVE, 90.44, Books",
    "10889, Shirt DNRZU, 316.48, Clothing",
    "82777, Shirt OZWXU, 790.46, Clothing",
    "43451, Mixer CKVJQ, 379.5, Home & Kitchen",
    "12848, Toaster VZXUE, 867.97, Home & Kitchen",
    "17646, Biography BPWXR, 424.83, Books",
    "85197, Cutting Board IJVPP, 986.89, Home & Kitchen"
]

# Initialize an empty list to store the product dictionaries
products = []

# Process each line of the raw product data
for line in raw_product_data:
    # Split the line by ', ' to separate the attributes
    product_id, name, price, category = line.split(', ')
    # Convert price to float and product_id to integer
    price = float(price)
    product_id = int(product_id)
    # Create a dictionary for the product
    product = {
        'ID': product_id,
        'Name': name,
        'Price': price,
        'Category': category
    }
    # Add the product dictionary to the products list
    products.append(product)

# Function to insert a new product
def insert_product(products, new_product):
    """Inserts a new product into the products list."""
    products.append(new_product)
    print(f"Product inserted: {new_product}")
```

```
# Function to update an existing product
def update_product(products, product_id, new_values):
    """Updates an existing product with new values given its ID."""
    for product in products:
        if product['ID'] == product_id:
            product.update(new_values)
            print(f"Product updated: {product}")
            return
    print("Product not found.")

# Function to delete a product
def delete_product(products, product_id):
    """Deletes a product from the list based on its ID."""
    initial_length = len(products)
    # Use list comprehension to filter out the product with the given ID
    products[:] = [product for product in products if product['ID'] != product_id]
    if len(products) < initial_length:
        print(f"Product with ID {product_id} deleted.")
    else:
        print("Product not found.")

# Function to search for products by ID or Name
def search_product(products, key, value):
    """Searches for products by a key (e.g., 'ID', 'Name')."""
    # Use list comprehension to find all products that match the key-value pair
    found_products = [product for product in products if product[key] == value]
    print(f"Search results for {key} = {value}:")
    for product in found_products:
        print(product)
    return found_products

# Function to sort products by price using Quick Sort
def quick_sort(products, low, high):
    """Sorts products in place by price using the Quick Sort algorithm."""
    if low < high:
        # Partition the array and get the pivot index
        pi = partition(products, low, high)
        # Recursively sort the elements before and after partition
        quick_sort(products, low, pi-1)
        quick_sort(products, pi+1, high)

def partition(products, low, high):
    """Partition the products list for quicksort."""
    pivot = products[high]['Price']
    i = low - 1
    for j in range(low, high):
        if products[j]['Price'] <= pivot:
            i += 1
            products[i], products[j] = products[j], products[i]
    products[i+1], products[high] = products[high], products[i+1]
    return i + 1

# Function to measure the sorting time
def measure_sorting_time(products, sort_func):
    start_time = time.time()
    sort_func(products, 0, len(products) - 1)
    end_time = time.time()
    return end_time - start_time

# Example usage of the functions
if __name__ == "__main__":
    # Insert a new product
    new_product = {'ID': 99999, 'Name': 'New Product', 'Price': 199.99, 'Category': 'Electronics'}
    insert_product(products, new_product)

    # Update an existing product
    update_product(products, 99999, {'Price': 299.99, 'Name': 'Updated Product'})

    # Delete a product
    delete_product(products, 99999)

    # Search for a product by ID
    search_product(products, 'ID', 40374)

    # Sort products by price
    print("Sorting products...")

    # Measure sorting time for already sorted data
    sorted_products = sorted(products, key=lambda x: x['Price'])
    sorted_time = measure_sorting_time(sorted_products, quick_sort)
    print(f"Time taken to sort already sorted data: {sorted_time} seconds.")

    # Measure sorting time for reverse sorted data
    reverse_sorted_products = sorted(products, key=lambda x: x['Price'], reverse=True)
    reverse_sorted_time = measure_sorting_time(reverse_sorted_products, quick_sort)
    print(f"Time taken to sort reverse sorted data: {reverse_sorted_time} seconds.")
    # Measure sorting time for random data
    random_products = products.copy()
    random_time = measure_sorting_time(random_products, quick_sort)
    print(f"Time taken to sort random data: {random_time} seconds.")

################################################################################end
```

5. Conclusion

→ With the use of basic Python data structures and operations, the script demonstrates data manipulation in an efficient way. Quick Sort does a good job at handling sorting, and the worst case, average case, and best case are all useful in analyzing how well operations are performed. As a result, this assignment enhances knowledge of complexity analysis, algorithms, and data structures.