Block 4: Computational Problem Solving

L39220303
TOSEEF ASHRAF PARVEEN

Content

Table of Figures

Introduction

The aim for this project is to use computational thinking to design algorithms for Jiraya Industries that sells safety products. The 3 algorithms used were create order, delete order, and modify order. These orders would make up a simple order management system with the requirements asked by the company. Later, the Jiraya industries is taken over by Tanjiro Safety Products. Tanjiro now wants to change the requirements which would mean that the three algorithms would also need to be changed to satisfy the new requirements. After a while, Tanjiro would like to compare if the second algorithm made is more resource efficient.

Task 1: Create an algorithm for Jiraya Industries

Jiraya industries is a company that sells motorcycle accessories, and this includes safety products. Some of these safety products would come as first aid kits that include various items. Jiraya wants to sell customizable kits with 5 items per order, and there cannot be duplicates inside. They want a computational problem thinker to understand the problem and find solutions and later translate them into code.

To make a customizable kit, an order would be made that could be created, deleted, and modified. For creating an order, the items selected would be added into a new order. To do this, each item would have a unique 4-digit id that will be used to add into the order. To delete an order, every order would also be assigned a unique 4-digit id, that when added to the delete function it would delete the order id and all the items inside of it. Lastly, to modify the order, an order id would be recalled and there would be an option to either add a new item to the order by its id or an item from the order id can also be deleted by recalling its id.

For this task, the abstract data types used were lists, dictionaries and sets. Dictionaries were used to store the catalogue of items and give them a unique id. Dictionaries are remarkably simple to use and are very flexible and it was used in this algorithm because it includes keys with values associated with it, which was important in this project because items needed to be associated to ids (Seniuk, Y. (2021)). Lists were used to store the order ids and all items within it. Lists are mutable which means their contents can be changed, which makes them very resizable, and it is quite easy to append and delete items (Van Rossum, G. (2023)). This is especially useful because an order can be changed a lot and very easily as well. The last data type used was sets, this data type was used for the specific reason that it does not allow duplicates inside of it, so there could only be one of each item per order id (Van Rossum, G. (2023)).

```
"""
Create_New_Order Function:

1. Generate a unique order ID.
2. Prompt the user to input the item IDs (up to 5) to add to the order.
3. Validate the input to ensure it does not exceed the maximum number of items per order.
4. Check for duplicate items in the new order.
5. Check for duplicates with existing orders.
6. Check if all item IDs are valid.
7. Create a new order list consisting of the order ID and the item IDs.
8. Add the new order to the orders list.
9. Print a success message indicating the creation of the order.
"""
```

Figure 1: Algorithms steps for create new order

```
"""
Remove_Order Function:

1. Iterate through each order in the orders list.
2. Check if the order ID matches the specified order_id.
3. If found, remove the order from the orders list.
4. Print a success message indicating the deletion of the order.
5. If not found, print a message indicating that the order ID does not exist.
"""
```

Figure 2: Algorithms steps for delete order

```
"""
Modify_Order Function:

1. Iterate through each order in the orders list.
2. Check if the order ID matches the specified order_id.
3. If found, print the current items in the order.
4. Prompt the user to choose whether to add or delete an item.
5. If adding an item:
    - Check if the order already has the maximum number of items.
    - Prompt the user to input the item IDs to add.
    - Check for duplicates between the items to add and the items already in the order.
    - Add the items to the order.
6. If deleting an item:
    - Prompt the user to input the item ID to delete.
    - Remove the item from the order.
7. Print a success message indicating the modification of the order.
8. If the order ID does not exist, print a message indicating it.
"""
```

Figure 3:Algorithms steps for modify order

```python
# List to store orders, where each order is represented as a list with the order ID as the first element
orders = []

# Counter to generate unique order IDs
order_id_counter = 1000

# Maximum number of orders
MaxOrders = 20

# Function to generate a unique order ID
def generate_order_id():
    # Accessing the global variable order_id_counter to generate unique order IDs
    global order_id_counter
    # Incrementing the order_id_counter by 1 to generate a new unique order ID
    order_id_counter += 1
    # Returning the newly generated unique order ID
    return order_id_counter
```

Figure 4: Order storage and function for id generation

```python
# Function to add items to a new order
def Create_New_Order(item_ids):
    global orders
    global MaxOrders

    # Check if maximum orders limit has been reached
    if len(orders) >= MaxOrders:
        print("You have reached the maximum limit of 20 orders.")
        return

    # Generate a unique order ID
    order_id = generate_order_id()

    # Check if the number of items to be added exceeds the limit
    if len(item_ids) > 5:
        print("You can only add up to 5 items in one order.")
        return
```

Figure 5: Function to create an order, part 1

```
# Check for duplicate items in the new order
if len(set(item_ids)) != len(item_ids):
    print("Duplicate items found in the new order. Please remove duplicates.")
    return

# Check for duplicates with existing orders
existing_item_ids = {item for order in orders for item in order[1:]}
if set(item_ids) & existing_item_ids:
    print("Some items are already in another order.")
    return

# Check if all item IDs are valid
if not all(item_id in kitdict for item_id in item_ids):
    print("One or more item IDs are invalid.")
    return

# Create a new order list
new_order = [order_id] + item_ids

# Add the new order to the orders list
orders.append(new_order)

# Print success message
print(f"Added {len(item_ids)} items to your order with Order ID: {order_id}.")
```

Figure 6: Function to create an order, part 2

```
# Function to remove an order by its ID
def Remove_Order(order_id):
    # Accessing the global variable orders to remove the specified order by its ID
    global orders
    # Iterating through each order in the orders list
    for order in orders:
        # Checking if the order ID matches the specified order_id
        if order[0] == order_id:
            # Removing the order from the orders list
            orders.remove(order)
            # Printing a success message indicating the deletion of the order
            print(f"Order ID {order_id} deleted successfully.")
            # Exiting the function after deleting the order
            return
    # If the specified order ID is not found in the orders list, print a message indicating it
    print("Order ID not found.")
```

Figure 7: Function to delete an order

```python
# Function to modify an order
def Modify_Order(order_id):
    # Accessing the global variable orders to modify the specified order by its ID
    global orders

    # Iterating through each order in the orders list
    for order in orders:
        # Checking if the order ID matches the specified order_id
        if order[0] == order_id:
            # Printing the current items in the order for the specified order ID
            print(f"Current items in the order for Order ID {order_id}:")
            for index, item_id in enumerate(order[1:], start=1):
                print(f"{index}. Item ID: {item_id}, Item Name: {kitdict.get(item_id)}")

            # Prompting the user to choose whether to add or delete an item
            choice = input("Do you want to add or delete an item? (add/delete): ")

            # If the user chooses to add an item
            if choice.lower() == "add":

                # Checking if the order already has the maximum number of items
                if len(order) >= 6:
                    print("You can only add up to 5 items per order.")
                    return
```

Figure 8: Function to modify an order, part 1

```
else:
    # Asking the user to input the item IDs (comma-separated) to add to the order
    items_to_add = input("Enter the item IDs (comma-separated) to add: ").split(",")
    # Converting the input item IDs to integers and stripping any whitespace
    items_to_add = [int(item_id.strip()) for item_id in items_to_add]

    # Checking for duplicates between the items to add and the items already in the order
    duplicates = set(items_to_add).intersection(order[1:])
    if duplicates:
        print(f"The following items are already in the order: {', '.join(map(str, duplicates))}")
        return

    # Checking if adding more items would exceed the limit
    if len(order) + len(items_to_add) > 5:
        print("Adding these items would exceed the limit of 5 items per order.")
        return

    # Adding the items to the order
    order.extend(items_to_add)
    # Printing a success message indicating that the items were added to the order
    print("Items added to the order.")
    return
```

Figure 9: Function to modify an order, part 2

```
        # If the user chooses to delete an item from the order
        elif choice.lower() == "delete":
            # Asking for the item ID to delete from the order
            item_to_delete = int(input("Enter the item ID to delete: "))

            # Checking if the item ID exists in the order
            if item_to_delete in order:
                # Removing the item from the order
                order.remove(item_to_delete)
                print(f"Item ID {item_to_delete} deleted from Order ID {order_id}.")

            else:
                # If the item ID is not found in the order
                print("Item ID not found in the order. No changes made.")
            return

# If the order ID doesn't exist
print(f"Order ID {order_id} does not exist.")
```

Figure 10: Function to modify an order, part 3

Task 2: Recreate the algorithm based on the change in scenario

Jiraya industries has now been taken over by Tanjiro Safety Products Ltd. Tanjiro has decided that they do not want to sell customizable kits and that they will sell 1 premade kit and they will sell up to 5 kits per order. This changes the way the create and modify order functions.

For the create function, instead of asking what items to include in the kit, now it will ask how many of the premade kits to add to the order. It will also now append and display the count of kits added. The delete function will act the same and it will delete the order id with all the kit count inside of it. The modify order, will display the previous count of kits in the order id and will prompt the user to enter the new number of kits they want. And just like the create function, it will append the new count of kits and will delete the previous count.

With this change, the abstract data type used where also changed to better suit the new requirements. Instead of using the dictionary to store the catalogue of items, now a tuple is used in its stead. The reason for this change is because tuples are immutable, meaning that the contents cannot be changed (Baka, B. (2017b)). For the storing of the orders, a queue was used instead of a list. This is because they are quite easy to use and have a more efficient memory management than lists, this is because a queue has fewer operations than lists thus having a simpler memory management (Lanaro, G. (2017)). A queue also is used because of it first in, first out structure because it resembles what a genuine business would work like, the first in the queue would be processed first.

```
"""
create_order Function:

1. Check if the maximum total kits limit has been reached.
2. Generate a unique order ID.
3. Prompt the user to input the number of kits to add to the order.
4. Validate the input to ensure it does not exceed the maximum number of kits per order.
5. Append the order (order ID, kits count) to the deque.
6. Print a success message indicating the creation of the order.
"""
```

Figure 11:Algorithm steps for create order

```
"""
delete_order Function:

1. Iterate through each order in the orders queue.
2. Check if the order ID matches the specified order_id.
3. If found, remove the order from the orders queue.
4. Print a success message indicating the deletion of the order.
5. If not found, print a message indicating that the order ID does not exist.
"""
```

Figure 12: Algorithm steps for delete order

```
"""
modify_order Function:

1. Iterate through each order in the orders queue.
2. Check if the order ID matches the specified order_id.
3. If found, print the current number of kits in the order.
4. Prompt the user to enter the new number of kits.
5. Validate the input to ensure it does not exceed the maximum number of kits per order.
6. Update the number of kits in the order.
7. Print a success message indicating the modification of the order.
8. If the order ID does not exist, print a message indicating it.
"""
```

Figure 13: Algorithm steps for modify order

```
# Queue to store orders, where each order is a tuple (order ID, count of kits)
orders_queue = deque()

# Counter to generate unique order IDs
order_id_counter = 1000

# Maximum number of kits per order
MaxKitsPerOrder = 5

# Maximum total number of kits across all orders
MaxTotalKits = 100


# Function to generate a unique order ID
def generate_order_id():
    # Access the global order_id_counter variable
    global order_id_counter
    # Increment the order_id_counter to generate a new unique order ID
    order_id_counter += 1
    # Return the newly generated unique order ID
    return order_id_counter
```

Figure 14: Order storage and function for id generation

```
# Function to calculate the total number of kits across all orders
def calculate_total_kits():
    # Initialize a variable to store the total number of kits
    total_kits = 0
    # Iterate through each order in the orders_queue
    for order in orders_queue:
        # Add the number of kits in the current order to the total number of kits
        total_kits += order[1]
    # Return the total number of kits across all orders
    return total_kits
```

Figure 15: Function to calculate total kits

```python
# Function to create an order with a specified number of kits
def create_order():
    global orders_queue
    # Check if maximum total kits limit has been reached
    if calculate_total_kits() >= MaxTotalKits:
        # Print a message if the maximum total kits limit has been reached
        print("You have reached the maximum limit of 100 kits.")
        # Exit the function
        return

    # Generate a unique order ID
    order_id = generate_order_id()
    # Ask the user how many kits they want to add to the order
    kits_count = int(input("Enter the number of kits to add to the order (maximum 5): "))
    # Validate maximum kits per order
    if kits_count > MaxKitsPerOrder:
        # Print a message if the number of kits exceeds the maximum kits per order
        print("Maximum number of kits per order allowed is 5.")
        # Exit the function
        return

    # Append the order to the queue
    orders_queue.append((order_id, kits_count))
    # Print a message indicating the creation of the order
    print(f"Order ID {order_id} for {kits_count} kits created.")
```

Figure 16: Function to create an order

```python
# Function to delete an order
def delete_order(order_id):
    global orders_queue
    # Iterate through each order in the orders_queue
    for order in orders_queue:
        # Check if the order ID matches the specified order_id
        if order[0] == order_id:
            # If a match is found, remove the order from the orders_queue
            orders_queue.remove(order)
            # Print a success message indicating the deletion of the order
            print(f"Order with ID {order_id} deleted successfully.")
            # Exit the function after deleting the order
            return
    # If the specified order ID is not found in the orders_queue, print a message indicating it
    print(f"Order with ID {order_id} does not exist.")
```

Figure 17: Function to delete an order

```python
# Function to modify an order by changing the count of kits
def modify_order(order_id):
    global orders_queue
    # Iterate through each order in the orders_queue using enumerate to track the index
    for index, order in enumerate(orders_queue):
        # Check if the order ID matches the specified order_id
        if order[0] == order_id:
            # Print the current number of kits in the order for the specified order ID
            print(f"Current number of kits in the order for Order ID {order_id}: {order[1]}")
            # Ask the user to enter the new number of kits
            new_count = int(input("Enter the new number of kits (maximum 5): "))
            # Validate maximum kits per order
            if new_count > MaxKitsPerOrder:
                print("Maximum number of kits per order allowed is 5.")
                return

            # Update the number of kits in the order
            orders_queue[index] = (order_id, new_count)
            # Print a success message indicating the modification of the order
            print(f"Order ID {order_id} modified successfully.")
            # Exit the function after modifying the order
            return
    # If the specified order ID is not found in the orders_queue, print a message indicating it
    print(f"Order ID {order_id} does not exist.")
```

Figure 18: Function to modify an order

Task 3: Compare the two algorithms created

To compare the time complexity of both algorithms, the big O notation was used. The big O is a mathematical notation used to describe the performance or time complexity of an algorithm. The big O notation expresses the worse time taken as a function of the input size. It is usually presented as O(1), O(n) or O(n^2) where n is the input size. O(1) shows a constant time complexity meaning that it ignores input size, and it shows a constant time. O(n) shows a linear time complexity meaning that time increments proportionally to the input size. O(n^2) shows a quadratic time complexity meaning that for every input the time grows exponentially (Shaffer, C. (2011)). The big O notation is

not only used to analyse and optimise algorithm, but it also is particularly useful to compare with regards to their performance.

The time complexity for creating an order for task 1 can be described as O(1), this is because it is a small process, and the input size is quite small as only 5 items can be inputted into the order. Checking for duplicates also has a constant time complexity of O(1) as there can only be 5 items per order. Generating an order id is constant as well because it only needs 1 id per order.

The time complexity for creating an order for task 2 can be described as O(n). This is because to check how many kits were added across all orders would require iterating through all orders in the list and this will cause a linear time complexity of O(n) where n is the input size. Apart from this, appending to the queue has a constant time complexity of O(1) as only 5 kits can be added to the order. So, the average time complexity for creating an order for task 2 would be linear, O(n).

The time complexity for deleting an order for both tasks can be described as O(n). This is because it iterates through the order lists with all the order ids, this means that it will take longer depending on how many order ids there are. But deleting an order id itself has constant time complexity of O(1) because ids are unique. Therefore, deleting an order id has a time complexity of O(n) where n is the input size.

The time complexity for modifying an order can be described as O(n). This is because, like previously mentioned, iterating through the order list is dependent on the input size and has a time complexity of O(n). But modifying the order itself has constant time complexity of O(1) because its only 5 items or kits per order, and 1 item will be added or deleted by its unique id. This means that the average time complexity for modifying an order is O(n) where n is the input size.

With all this data, it is safe to say that both algorithms have a similar performance and have no clear distinction in their time complexity.

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    57     49.5 MiB     49.5 MiB           1   @profile
    58                                         # Function to add items to a new order
    59                                         def Create_New_Order(item_ids):
    60                                             global orders
    61                                             global MaxOrders
    62
    63                                             # Check if maximum orders limit has been reached
    64     49.5 MiB      0.0 MiB           1       if len(orders) >= MaxOrders:
    65                                                 print("You have reached the maximum limit of 20 orders.")
    66                                                 return
    67
    68                                             # Generate a unique order ID
    69     49.5 MiB      0.0 MiB           1       order_id = generate_order_id()
    70
    71                                             # Check if the number of items to be added exceeds the limit
    72     49.5 MiB      0.0 MiB           1       if len(item_ids) > 5:
    73                                                 print("You can only add up to 5 items in one order.")
    74                                                 return
    75
    76                                             # Check for duplicate items in the new order
    77     49.5 MiB      0.0 MiB           1       if len(set(item_ids)) != len(item_ids):
    78                                                 print("Duplicate items found in the new order. Please remove duplicates.")
    79                                                 return
    80
    81                                             # Check for duplicates with existing orders
    82     49.5 MiB      0.0 MiB           3       existing_item_ids = {item for order in orders for item in order[1:]}
    83     49.5 MiB      0.0 MiB           1       if set(item_ids) & existing_item_ids:
    84                                                 print("Some items are already in another order.")
    85                                                 return
    86
    87                                             # Check if all item IDs are valid
    88     49.5 MiB      0.0 MiB           5       if not all(item_id in kitdict for item_id in item_ids):
    89                                                 print("One or more item IDs are invalid.")
    90                                                 return
    91
    92                                             # Create a new order list
    93     49.5 MiB      0.0 MiB           1       new_order = [order_id] + item_ids
    94
    95                                             # Add the new order to the orders list
    96     49.5 MiB      0.0 MiB           1       orders.append(new_order)
    97
    98                                             # Print success message
    99     49.5 MiB      0.0 MiB           1       print(f"Added {len(item_ids)} items to your order with Order ID: {order_id}.")
```

Figure 19: Memory usage for create new order in task 1

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
   112     49.6 MiB     49.6 MiB           1   @profile
   113                                         # Function to remove an order by its ID
   114                                         def Remove_Order(order_id):
   115                                             # Accessing the global variable orders to remove the specified order by its ID
   116                                             global orders
   117                                             # Iterating through each order in the orders list
   118     49.6 MiB      0.0 MiB           1       for order in orders:
   119                                                 # Checking if the order ID matches the specified order_id
   120     49.6 MiB      0.0 MiB           1           if order[0] == order_id:
   121                                                     # Removing the order from the orders list
   122     49.6 MiB      0.0 MiB           1               orders.remove(order)
   123                                                     # Printing a success message indicating the deletion of the order
   124     49.6 MiB      0.0 MiB           1               print(f"Order ID {order_id} deleted successfully.")
   125                                                     # Exiting the function after deleting the order
   126     49.6 MiB      0.0 MiB           1               return
   127                                             # If the specified order ID is not found in the orders list, print a message indicating it
   128                                             print("Order ID not found.")
```

Figure 20: Memory usage for delete order in task 1

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
   150     49.5 MiB     49.5 MiB           1   @profile
   151                                         # Function to modify an order
   152                                         def Modify_Order(order_id):
   153                                             # Accessing the global variable orders to modify the specified order by its ID
   154                                             global orders
   155
   156                                             # Iterating through each order in the orders list
   157     49.5 MiB      0.0 MiB           1       for order in orders:
   158                                                 # Checking if the order ID matches the specified order_id
   159     49.5 MiB      0.0 MiB           1           if order[0] == order_id:
   160                                                     # Printing the current items in the order for the specified order ID
   161     49.5 MiB      0.0 MiB           1               print(f"Current items in the order for Order ID {order_id}:")
   162     49.5 MiB      0.0 MiB           2               for index, item_id in enumerate(order[1:], start=1):
   163     49.5 MiB      0.0 MiB           1                   print(f"{index}. Item ID: {item_id}, Item Name: {kitdict.get(item_id)}")
   164
   165                                                     # Prompting the user to choose whether to add or delete an item
   166     49.5 MiB      0.0 MiB           1               choice = input("Do you want to add or delete an item? (add/delete): ")
   167
   168                                                     # If the user chooses to add an item
   169     49.5 MiB      0.0 MiB           1               if choice.lower() == "add":
   170
   171                                                         # Checking if the order already has the maximum number of items
   172     49.5 MiB      0.0 MiB           1                   if len(order) >= 6:
   173                                                             print("You can only add up to 5 items per order.")
   174                                                             return
   175
   176                                                         else:
   177                                                             # Asking the user to input the item IDs (comma-separated) to add to the order
   178     49.5 MiB      0.0 MiB           1                   items_to_add = input("Enter the item IDs (comma-separated) to add: ").split(",")
   179                                                             # Converting the input item IDs to integers and stripping any whitespace
   180     49.5 MiB      0.0 MiB           4                   items_to_add = [int(item_id.strip()) for item_id in items_to_add]
   181
   182                                                         # Checking for duplicates between the items to add and the items already in the order
   183     49.5 MiB      0.0 MiB           1                   duplicates = set(items_to_add).intersection(order[1:])
   184     49.5 MiB      0.0 MiB           1                   if duplicates:
   185                                                             print(f"The following items are already in the order: {', '.join(map(str, duplicates))}")
   186                                                             return
   187
   188                                                         # Checking if adding more items would exceed the limit
   189     49.5 MiB      0.0 MiB           1                   if len(order) + len(items_to_add) > 5:
   190                                                             print("Adding these items would exceed the limit of 5 items per order.")
   191                                                             return
   192
   193                                                         # Adding the items to the order
   194     49.5 MiB      0.0 MiB           1                   order.extend(items_to_add)
   195                                                         # Printing a success message indicating that the items were added to the order
   196     49.5 MiB      0.0 MiB           1                   print("Items added to the order.")
   197     49.5 MiB      0.0 MiB           1                   return
   198
```

Figure 21: Memory usage for modify order in task 1, part 1

```
199                                             # If the user chooses to delete an item from the order
200     49.5 MiB      0.0 MiB           1        elif choice.lower() == "delete":
201                                                 # Asking for the item ID to delete from the order
202     49.6 MiB      0.0 MiB           1            item_to_delete = int(input("Enter the item ID to delete: "))
203
204                                                 # Checking if the item ID exists in the order
205     49.6 MiB      0.0 MiB           1            if item_to_delete in order:
206                                                     # Removing the item from the order
207     49.6 MiB      0.0 MiB           1                order.remove(item_to_delete)
208     49.6 MiB      0.0 MiB           1                print(f"Item ID {item_to_delete} deleted from Order ID {order_id}.")
209
210                                                 else:
211                                                     # If the item ID is not found in the order
212                                                     print("Item ID not found in the order. No changes made.")
213     49.6 MiB      0.0 MiB           1            return
214
215                                         # If the order ID doesn't exist
216                                         print(f"Order ID {order_id} does not exist.")
```

Figure 22: Memory usage for modify order in task 1, part 2

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    66     49.2 MiB     49.2 MiB           1   @profile
    67                                         # Function to create an order with a specified number of kits
    68                                         def create_order():
    69                                             global orders_queue
    70                                             # Check if maximum total kits limit has been reached
    71     49.2 MiB      0.0 MiB           1       if calculate_total_kits() >= MaxTotalKits:
    72                                                 # Print a message if the maximum total kits limit has been reached
    73                                                 print("You have reached the maximum limit of 100 kits.")
    74                                                 # Exit the function
    75                                                 return
    76
    77                                             # Generate a unique order ID
    78     49.2 MiB      0.0 MiB           1       order_id = generate_order_id()
    79                                             # Ask the user how many kits they want to add to the order
    80     49.2 MiB      0.0 MiB           1       kits_count = int(input("Enter the number of kits to add to the order (maximum 5): "))
    81                                             # Validate maximum kits per order
    82     49.2 MiB      0.0 MiB           1       if kits_count > MaxKitsPerOrder:
    83                                                 # Print a message if the number of kits exceeds the maximum kits per order
    84                                                 print("Maximum number of kits per order allowed is 5.")
    85                                                 # Exit the function
    86                                                 return
    87
    88                                             # Append the order to the queue
    89     49.2 MiB      0.0 MiB           1       orders_queue.append((order_id, kits_count))
    90                                             # Print a message indicating the creation of the order
    91     49.2 MiB      0.0 MiB           1       print(f"Order ID {order_id} for {kits_count} kits created.")
```

Figure 23: Memory usage for create order in task 2

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
   104     49.3 MiB     49.3 MiB           1   @profile
   105                                         # Function to delete an order
   106                                         def delete_order(order_id):
   107                                             global orders_queue
   108                                             # Iterate through each order in the orders_queue
   109     49.3 MiB      0.0 MiB           1       for order in orders_queue:
   110                                                 # Check if the order ID matches the specified order_id
   111     49.3 MiB      0.0 MiB           1           if order[0] == order_id:
   112                                                     # If a match is found, remove the order from the orders_queue
   113     49.3 MiB      0.0 MiB           1               orders_queue.remove(order)
   114                                                     # Print a success message indicating the deletion of the order
   115     49.3 MiB      0.0 MiB           1               print(f"Order with ID {order_id} deleted successfully.")
   116                                                     # Exit the function after deleting the order
   117     49.3 MiB      0.0 MiB           1               return
   118                                             # If the specified order ID is not found in the orders_queue, print a message indicating it
   119                                             print(f"Order with ID {order_id} does not exist.")
```

Figure 24: Memory usage for delete order in task 2

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
   136     49.3 MiB     49.3 MiB           1   @profile
   137                                         # Function to modify an order by changing the count of kits
   138                                         def modify_order(order_id):
   139                                             global orders_queue
   140                                             # Iterate through each order in the orders_queue using enumerate to track the index
   141     49.3 MiB      0.0 MiB           1       for index, order in enumerate(orders_queue):
   142                                                 # Check if the order ID matches the specified order_id
   143     49.3 MiB      0.0 MiB           1           if order[0] == order_id:
   144                                                     # Print the current number of kits in the order for the specified order ID
   145     49.3 MiB      0.0 MiB           1               print(f"Current number of kits in the order for Order ID {order_id}: {order[1]}")
   146                                                     # Ask the user to enter the new number of kits
   147     49.3 MiB      0.0 MiB           1               new_count = int(input("Enter the new number of kits (maximum 5): "))
   148                                                     # Validate maximum kits per order
   149     49.3 MiB      0.0 MiB           1               if new_count > MaxKitsPerOrder:
   150                                                         print("Maximum number of kits per order allowed is 5.")
   151                                                         return
   152
   153                                                     # Update the number of kits in the order
   154     49.3 MiB      0.0 MiB           1               orders_queue[index] = (order_id, new_count)
   155                                                     # Print a success message indicating the modification of the order
   156     49.3 MiB      0.0 MiB           1               print(f"Order ID {order_id} modified successfully.")
   157                                                     # Exit the function after modifying the order
   158     49.3 MiB      0.0 MiB           1               return
   159                                             # If the specified order ID is not found in the orders_queue, print a message indicating it
   160                                             print(f"Order ID {order_id} does not exist.")
```

Figure 25: Memory usage for modify order in task 3

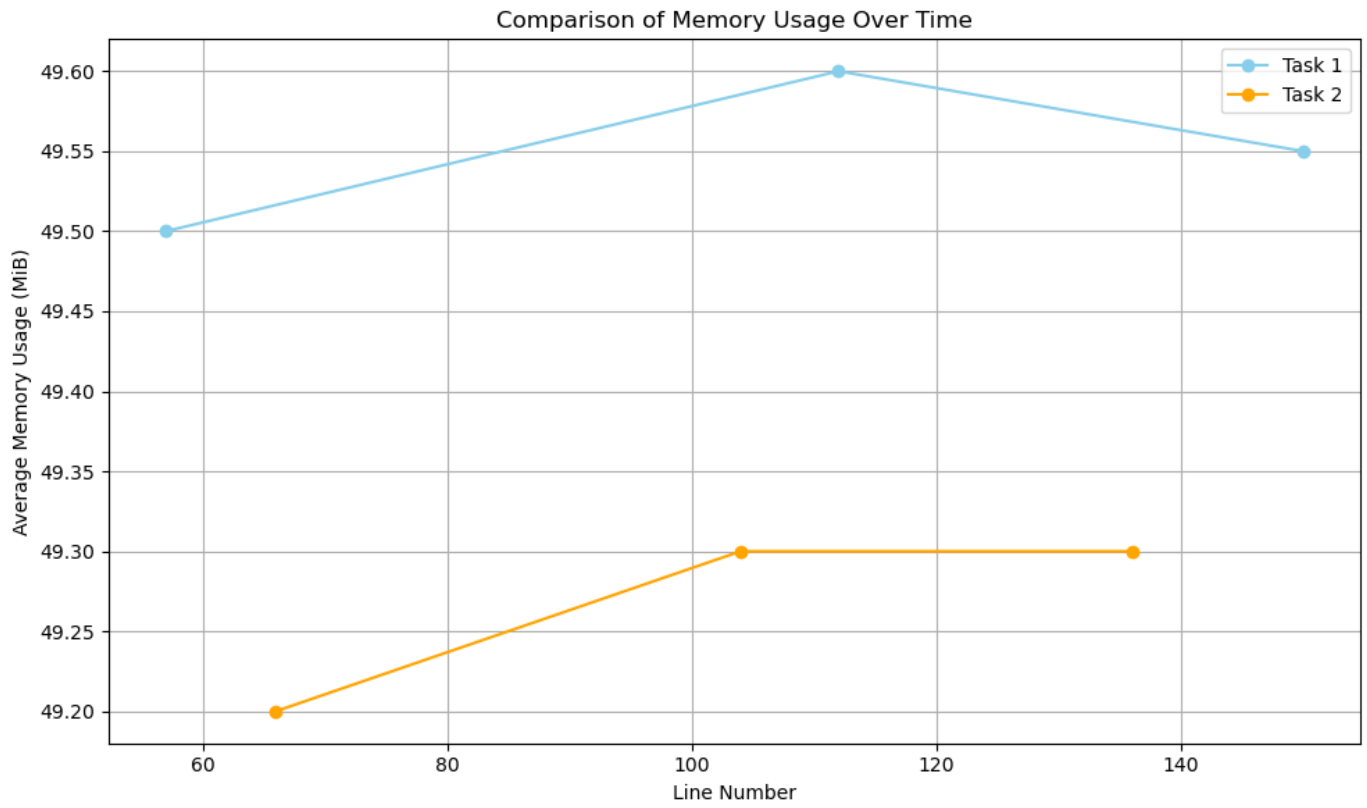

Figure 26: Graph of memory usage per line number

After collecting all the memory usage of every algorithm as shown in figures 19-25. This data was then inputted in a graph to compare their performance. Although, task 2 performed better overall than task 1 and had better memory usage, it is still not enough to say there was a significant improvement of resource utilization. This mean that Tanjiro has used a different algorithm, falsely claiming a 25% performance increase.

Task 4: Calculating the probability of receiving defective items

```
#overall number of kits supplied
OverallKits = 2000
#Number of kits supplied by Uzumaki Safety Suppliers
UzumakiKitsSupplied = 1500
#Number of kits supplied by Khairan Safety Company
KhairanKitsSupplied = 500


#Percentage given of defective kits supplied by Uzumaki Safety Suppliers (20% = 0.2)
UzumakiFaultyPercentage = 0.2
print("The percentage of faulty items supplied by Uzumaki is: ", UzumakiFaultyPercentage * 100, "%")
#Percentage given of defective kits supplied by Khairan Safety Company (10% = 0.1)
KhairanFaultyPercentage = 0.1
print("The percentage of faulty items supplied by Khairan is: ", KhairanFaultyPercentage * 100, "%")


#Calculating the number of defective kits supplied by Uzumaki Safety Suppliers
UzumakiFaultyKits = (UzumakiFaultyPercentage * UzumakiKitsSupplied)
#Calculating the number of defective kits supplied by Khairan Safety Company
KhairanFaultyKits = (KhairanFaultyPercentage * KhairanKitsSupplied)
#Displaying the number of defective kits from Uzumaki Safety Suppliers and Khairan Safety Company
print("The number of defective kits supplied by Uzumaki Safety Suppliers is: ", UzumakiFaultyKits,"\n",
      "The number of defective kits supplied by Khairan Safety Company is: ", KhairanFaultyKits)
```

Figure 27: Calculating the probability of defective kits supplied by each company

```
#Calculating the probability of receiving a defective kit
ProbabilityFaultyKit = (UzumakiFaultyKits + KhairanFaultyKits)/(UzumakiKitsSupplied + KhairanKitsSupplied)
#Displaying the probability of receiving a defective kit
print("The probability of receiving a defective kit is: ", ProbabilityFaultyKit * 100, "%")
```

Figure 28: Calculating the probability of finding a faulty kit supplied

```
#Calculating the probability of a kit coming from Uzumaki Safety Suppliers given that its a defective kit using base theorem
ProbabilityUzumakiGivenDefective = (UzumakiFaultyPercentage * 0.75) / ProbabilityFaultyKit
#Displaying the probability of a kit coming from Uzumaki Safety Suppliers given that its a defective kit
print("The probability of a kit coming from Uzumaki Safety Suppliers given that its a defective kit is: ", ProbabilityUzumakiGivenDefective * 100, "%")
```

Figure 29: Calculating the probability that a kit is supplied by Uzumaki given that it is a defective kit

```
# Calculating the probability of a kit being defective given that it came from Uzumaki Safety Suppliers
ProbabilityDefectiveGivenUzumaki = (ProbabilityUzumakiGivenDefective * ProbabilityFaultyKit) / (UzumakiKitsSupplied / OverallKits)
#Displaying the probability of a kit being defective given that it came from Uzumaki Safety Supplierst
print("The probability of a kit being defective given that it came from Uzumaki Safety Suppliers is: ", ProbabilityDefectiveGivenUzumaki * 100, "%")
```

Figure 30: Calculating the probability that a kit is defective given that it was supplied by Uzumaki

Bayes theorem is a mathematical equation used for conditional probability just like in figure 5 (Joyce, J. (2003)). The equation for bayes theorem is:

$$\mathbf{P}_E(H) = [\mathbf{P}(H)/\mathbf{P}(E)]\ \mathbf{P}_H(E)$$

Where $\mathbf{P}(H)$ would be the Uzumaki faulty percentage and $\mathbf{P}_H(E)$ would be the probability of faulty kits. The condition for this problem is to find the probability of a kit being supplied by Uzumaki given that it is a faulty kit.

```
The percentage of faulty items supplied by Uzumaki is:  20.0 %
The percentage of faulty items supplied by Khairan is:  10.0 %
The number of defective kits supplied by Uzumaki Safety Suppliers is:  300.0
 The number of defective kits supplied by Khairan Safety Company is:  50.0
The probability of receiving a defective kit is:  17.5 %
The probability of a kit coming from Uzumaki Safety Suppliers given that its a defective kit is:  85.71428571428574 %
The probability of a kit being defective given that it came from Uzumaki Safety Suppliers is:  20.000000000000004 %
```

Figure 31: Result for task 4

Task 5: Calculating the probability of receiving defective items with new supplier

```
#overall number of kits supplied
OverallKits = 2000
#Number of kits supplied by Uzumaki Safety Suppliers
UzumakiKitsSupplied = OverallKits * 0.5
#Number of kits supplied by Khairan Safety Company
KhairanKitsSupplied = OverallKits * 0.2
#Number of kits supplied by Leaf Firt-Aid Kits Ltd
LeafKitsSupplied = OverallKits * 0.3


#Percentage given of defective kits supplied by Uzumaki Safety Suppliers (20% = 0.2)
UzumakiFaultyPercentage = 0.2
#Percentage given of defective kits supplied by Khairan Safety Company (10% = 0.1)
KhairanFaultyPercentage = 0.1
#Assuming Percentage given of defective kits supplied by Leaf Firt-Aid Kits Ltd (10% = 0.1)
LeafFaultyPercentage = 0.1


#Calculating the number of defective kits supplied by Uzumaki Safety Suppliers
UzumakiFaultyKits = (UzumakiFaultyPercentage * UzumakiKitsSupplied)
#Calculating the number of defective kits supplied by Khairan Safety Company
KhairanFaultyKits = (KhairanFaultyPercentage * KhairanKitsSupplied)
#Calculating the number of defective kits supplied by Leaf Firt-Aid Kits Ltd
LeafFaultyKits = (LeafFaultyPercentage * LeafKitsSupplied)
#Displaying the number of defective kits from Uzumaki Safety Suppliers, Khairan Safety Company and Leaf Firt-Aid Kits Ltd
print("The number of defective kits from Uzumaki Safety Suppliers is", UzumakiFaultyKits, "\n",
      "The number of defective kits from Khairan Safety Company is", KhairanFaultyKits, "\n",
      "The number of defective kits from Leaf Fisrt-Aid Kits Ltd is", LeafFaultyKits)
```

Figure 32: Calculating the probability of defective kits supplied by each company

```
#Calculating the probability of receiving a defective kit
ProbabilityFaultyKit = (UzumakiFaultyKits + KhairanFaultyKits + LeafFaultyKits)/(UzumakiKitsSupplied + KhairanKitsSupplied + LeafKitsSupplied)
#Displaying the probability of receiving a defective kit
print("The probability of receiving a defective kit is", ProbabilityFaultyKit * 100, "%")
```

Figure 33: Calculating the new probability of finding faulty kit supplied

```
#Calculating the probability of a kit coming from Uzumaki Safety Suppliers given that its a defective kit using base theorem
ProbabilityUzumakiGivenDefective = UzumakiFaultyPercentage * 0.5 / ProbabilityFaultyKit
#Displaying the probability of a kit coming from Uzumaki Safety Suppliers given that its a defective kit
print("The probability of a kit coming from Uzumaki Safety Suppliers given that its a defective kit is", ProbabilityUzumakiGivenDefective * 100, "%")
```

Figure 34: Calculating the new probability that a kit is supplied by Uzumaki given that it is a defective kit

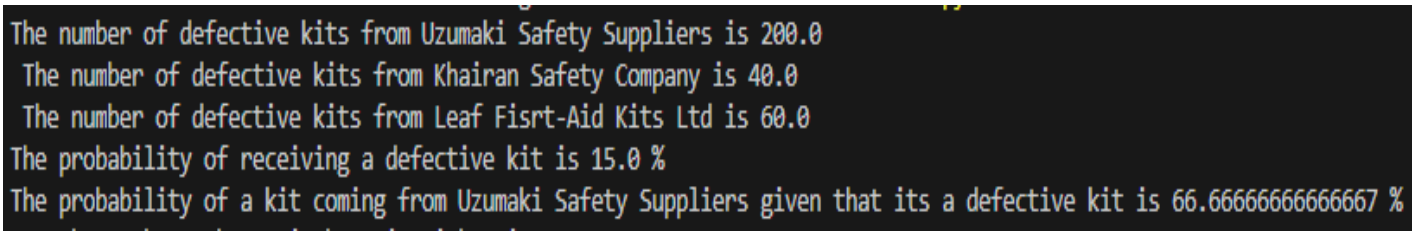

Figure 35: Result for task 5

The introduction of Leaf First Aid Kits Ltd has decreased the percentage of faulty items overall. The probability of finding a faulty item went from 17.5% to 15% which is a 2.5% improvement overall. In conclusion, the introduction of Leaf First Aid Kits Ltd has somewhat helped in decreasing the probability of finding a faulty item.

Conclusion

This project helped to enhance the understanding of algorithms. It also enhanced my knowledge of the different data structures, and how to optimise the algorithms with different abstract data types and analysing its time complexity. This knowledge will be helpful in the future, not only in python but also in the planning stages of algorithms and performance improvements of time complexity.

References

Seniuk, Y. (2021). *Ukrainian and Foreign Science: Yesterday, Today, Tomorrow CONFERENCE PROCEEDINGS*. [online] pp.230–232. Available at: https://kamgs3.kpi.ua/wp-content/uploads/2021/12/Zbirnyk_PART_1_m.pdf#page=230 [Accessed 2 Apr. 2024].

Lanaro, G. (2017). *Python High Performance*. [online] *Google Books*. Packt Publishing Ltd. Available at: https://books.google.co.uk/books?hl=en&lr=&id=aHc5DwAAQBAJ&oi=fnd&pg=PP1&dq=queue+in+python&ots=8kF1_ESnDE&sig=Jcssuu2TKsUr-8TuqD6ZJuwZJfI&redir_esc=y#v=onepage&q=queue%20in%20python&f=false [Accessed 12 Apr. 2024].

Van Rossum, G. (2023). *Python Tutorial*. [online] Available at: https://scicomp.ethz.ch/public/manual/Python/3.9.9/tutorial.pdf [Accessed 29 Mar. 2024].

Baka, B. (2017b). *Python Data Structures and Algorithms*. [online] *Google Books*, Packt Publishing Ltd, p.38. Available at: https://books.google.co.uk/books?hl=en&lr=&id=oHc5DwAAQBAJ&oi=fnd&pg=PP1&dq=Tuples+python+immutable&ots=TveCJc4heC&sig=izqpfomxa1GfldgKCT4Zd5jZxZc&redir_esc=y#v=onepage&q=Tuples%20python%20immutable&f=false [Accessed 3 Apr. 2024].

Shaffer, C. (2011). *A Practical Introduction to Data Structures and Algorithm Analysis Edition 3.2 (C++ Version)*. [online] p.68. Available at: https://people.cs.vt.edu/~shaffer/Book/C++3e20110520.pdf [Accessed 12 Apr. 2024].

Joyce, J. (2003). Bayes' Theorem. *seop.illc.uva.nl*. [online] Available at: https://seop.illc.uva.nl/entries/bayes-theorem/ [Accessed 2 Apr. 2024].