

# 1 Cover Page (With GitHub Link to your Source Code)

**INTI International College Penang**  
**School of Computing**

**3+0 Bachelor of Science (Hons) in Computer Science, in collaboration with Coventry University, UK**

## **Coursework cover sheet**

### **Section A - To be completed by the student**

<b>Full Name: Lee Je Rel</b>	
<b>INTI Student ID Number: P22014086</b>	
<b>CU Student ID Number: 16929706</b>	
<b>Semester: AUGUST 2025</b>	
Lecturer: Khor Jia Yun	
Module Code and Title: 5002CMD Advanced Algorithms	
Assignment No. / Title: Individual Report	50% of Module Mark
Hand out date: 10-Sept-2025 (Wednesday of Week 4)	Due date: 15-Nov-2025 (Saturday of Week 13)

**GitHub Link:**

<https://github.com/Alust0/Advance-Algorythm-August2025>

Penalties: No late work will be accepted. If you are unable to submit coursework on time due to extenuating circumstances you may be eligible for an extension. Please consult the lecturer.

**Declaration:**

*I the undersigned **confirm** that I have **read and agree to abide** by the University **regulations on plagiarism and cheating** and Faculty coursework policies and procedures. I confirm that this piece of **work is my own**. I consent to appropriate storage of our work for plagiarism checking.*

Signature(s): \_\_\_\_\_*leejarel*\_\_\_\_\_

**Section B - To be completed by the module leader**

Intended learning outcomes (LO) assessed by this work:

1. Evaluate the complexity and efficiency of algorithms for solving a range of problems.
2. Apply algorithms and data structures to solve novel problems.
3. Demonstrate solutions to intractable problems by considering computational complexity and algorithm theory.
4. Analyse the issue of data consistency and multi-threading in a concurrent application.

Marking scheme	Max	Mark
Refer to the attached rubric.		
Total		

Lecturer's Feedback

Internal Moderator's Feedback

## 2 TurnItIn Result Summary Page



## Digital Receipt

This receipt acknowledges that **Turnitin** received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author: LEE JE REL  
Assignment title: ASSIGNMENT 1 TURNITIN CHECKING  
Submission title: LeeJeRel\_16929706\_Advanced Algorithmm.pdf  
File name: LeeJeRel\_16929706\_Advanced\_Algorithm.pdf  
File size: 1.3M  
Page count: 38  
Word count: 2,824  
Character count: 16,220  
Submission date: 15-Nov-2025 10:29PM (UTC+0800)  
Submission ID: 2815704215

1 Cover Page (With GitHub Link to your Source Code)

INT International College Penang  
School of Computing  
3rd Bachelor of Science (Hons) in Computer Science, in collaboration with Coventry University, UK

Coursework cover sheet

Section A - To be completed by the student

Full Name: Lee Je Rel	
INT Student ID Number: P2314066	
CU Student ID Number: 16929706	
Semester: AUGUST 2025	
Lecturer: Khair Jia Yee	
Module Code and Title: SSC20MD Advanced Algorithms	
Assignment No. / Title: Individual Report	50% of Module Mark
Hand out date: 10-Sep-2025 (Wednesday of Week 4)	Due date: 15-Nov-2025 (Saturday of Week 12)

# LeeJeRel\_16929706\_Advanced Algorythm.pdf

## ORIGINALITY REPORT

7%

SIMILARITY INDEX

5%

INTERNET SOURCES

0%

PUBLICATIONS

7%

STUDENT PAPERS

## PRIMARY SOURCES

1

Submitted to INTI Universal Holdings SDM  
BHD

Student Paper

7%

2

Submitted to Colorado State University,  
Global Campus

Student Paper

<1%

Exclude quotes Off

Exclude bibliography On

Exclude matches Off

### 3 Table of Contents

<b>1 Cover Page (With GitHub Link to your Source Code)</b>	<b>1</b>
<b>2 TurnItIn Result Summary Page</b>	<b>5</b>
<b>3 Table of Contents</b>	<b>8</b>
<b>4 Question 1</b>	<b>9</b>
4.1 Hash table with separate chaining	9
4.2 product retail shop entity class	10
4.3 Command line menu ,Insert and search function	13
4.4 Separate chaining and one dimensional array performance comparison	16
<b>5 Question 2</b>	<b>19</b>
5.1 unweighted directed graph data structure	19
5.2 Person domain / entity class	19
5.3 Person object creation	20
5.4 Graph object creation	20
5.5 mandatory features	21
5.6 Additional Features and Output screenshots	23
5.6.1 Additional Features	23
5.6.2 Output screenshots	25
<b>6 Question 3</b>	<b>31</b>
6.1 If python is concurrent processing or parallel processing	31
6.2 Big-O function	31
6.3 Multithreading	32
6.4 Single threading	34
6.5 Discussion and output	35
6.5.1 comparison of steps 3 and 4 (total and average time taken)	35
6.5.2 Why multithreading did shorten the time taken for this experiment	35
6.5.3 Example situation where multithreading will improve the performance	36
<b>7 Challenges faced and personal reflections</b>	<b>37</b>
<b>8 References</b>	<b>40</b>

## 4 Question 1

### 4.1 Hash table with separate chaining

The HashTableArray class will implement a hash table that will create a list that uses separate chaining like a bucket in the table that can store multiple values. With using the Entry class each key value pair will be represented. Once the hashtable bowl is initialized it will have a default capacity of 10 buckets, and then by applying Python's built in hash we can use a `_hash` method that computes and index. An insert method will be created to add new items or update existing items, then the search method will retrieve values by going to their corresponding buckets, the `__len__` method is used to verify the number of items that is stored in there, the delete button will remove key value pair, and lastly a print table method is used to provide debugging:

```
import time
#1
class HashTableArray: # separate chaining class
    class Entry: # bucket entry type
        def __init__(self, key, value):
            self.key = key # unique identifier
            self.value = value # stored data

    def __init__(self, capacity=10): # Initialize hash table with capacity
        self.capacity = capacity # Number of buckets/slots
        self.table = [[] for _ in range(capacity)] # Array of empty lists (chains)
        self.size = 0 # Counter for total items stored

    def _hash(self, key): # Hash function to compute bucket index
        return hash(key) % self.capacity # Use modulo to get array index

    def insert(self, key, value): # Insert/update key-value pair
        index = self._hash(key) # Get bucket index using hash function
        bucket = self.table[index] # Access the bucket (linked list)

        for entry in bucket: # Check if key already exists
            if entry.key == key: # Key found
                entry.value = value # Update existing value
                return # Exit early

        bucket.append(self.Entry(key, value)) # New key: add to bucket
        self.size += 1 # Increment item count

    def search(self, key): # Search for value by key (returns None if not found)
        index = self._hash(key) # Compute bucket index
        bucket = self.table[index] # Get the bucket

        for entry in bucket: # Linear search within bucket
            if entry.key == key: # Key found
                return entry.value # Return the associated value

        return None # Key not found in hash table

    def delete(self, key): # delete method
        index = self._hash(key)
        bucket = self.table[index]
        for i, e in enumerate(bucket):
            if e.key == key:
                del bucket[i]
                self.size -= 1
                return True
        return False
```

```

def __len__(self): # length method for size of hash table
    return self.size

def print_table(self): # print method (for debugging)
    for i, bucket in enumerate(self.table):
        print(f'{i}: ', end='')
        for entry in bucket:
            print(f'[{entry.key}, {entry.value}]', end=' ')
        print()

```

## 4.2 product retail shop entity class

The entity class BabyProduct is defined. It stores product data such as product ID, name, price, stock quantity and category:

```

# Entity class for baby shop products
class BabyProduct:
    def __init__(self, productId, name, price, stock, category):
        self.productId = productId
        self.name = name
        self.price = price
        self.stock = stock
        self.category = category

    def __repr__(self):
        return f"{self.productId} - {self.name} | RM{self.price:.2f} | Stock: {self.stock} | Category: {self.category}"

```

The `local_storage_system` method creates a hash table with the capacity of 10 buckets. It prepares a one dimensional array for the comparison between hash table and one dimensional array. I have created a set of 50 baby product records covering diapers, feeding items, skin care, and other necessities baby products:

```

#2
def local_storage_system():
    hashTable = HashTableArray(capacity=10)
    arrayStorage = [] # 1D array #4

    # Pre-defined products (ALL UPPERCASE)
    p1 = BabyProduct("D001", "DRYPERS WEE WEE DIAPERS", 29.90, 100, "DIAPERS")
    p2 = BabyProduct("B002", "PIGEON BABY BOTTLE 240ML", 24.50, 50, "FEEDING")
    p3 = BabyProduct("W003", "PUREEN BABY WIPES 80S", 9.90, 200, "CARE")
    p4 = BabyProduct("L004", "JOHNSON'S BABY LOTION 200ML", 16.90, 80, "SKINCARE")
    p5 = BabyProduct("D005", "MAMYPOKO EXTRA DRY DIAPERS M SIZE 60PCS", 45.90, 120, "DIAPERS")
    p6 = BabyProduct("D006", "HUGGIES DRY PANTS L SIZE 50PCS", 39.90, 90, "DIAPERS")
    p7 = BabyProduct("F007", "AVENT NATURAL FEEDING BOTTLE 260ML", 34.90, 40, "FEEDING")
    p8 = BabyProduct("S013", "SEBAMED BABY LOTION 200ML", 29.90, 70, "SKINCARE")
    p9 = BabyProduct("F008", "TOMMEE TIPSEE CLOSER TO NATURE BOTTLE 150ML", 29.90, 30, "FEEDING")
    p10 = BabyProduct("C009", "DR BROWN'S BABY PACIFIER 2PCS", 18.90, 60, "CARE")
    p11 = BabyProduct("C010", "PUKU BABY COMB AND BRUSH SET", 12.50, 80, "CARE")
    p12 = BabyProduct("W011", "HUGGIES BABY WIPES 80S", 11.90, 150, "CARE")
    p13 = BabyProduct("W012", "MAMYPOKO BABY WIPES 100S", 13.90, 140, "CARE")
    p14 = BabyProduct("S014", "BUDS ORGANICS BABY SHAMPOO 230ML", 32.90, 50, "SKINCARE")
    p15 = BabyProduct("H015", "PANADOL CHILDREN SUSPENSION 100ML", 14.90, 60, "HEALTH")
    p16 = BabyProduct("H016", "NUBY BABY NASAL ASPIRATOR", 19.90, 40, "HEALTH")
    p17 = BabyProduct("T017", "FISHER-PRICE BABY RATTLE SET", 22.90, 35, "TOYS")
    p18 = BabyProduct("T018", "FISHER-PRICE TEETHING RING", 15.90, 55, "TOYS")
    p19 = BabyProduct("B019", "BABY COTTON BIB 3PCS PACK", 9.90, 120, "CLOTHING")
    p20 = BabyProduct("B020", "BABY MITTENS AND BOOTIES SET", 7.90, 200, "CLOTHING")
    p21 = BabyProduct("C021", "PUREEN BABY HEAD-TO-TOE WASH 750ML", 18.50, 90, "CARE")
    p22 = BabyProduct("C022", "JOHNSON'S BABY POWDER 500G", 12.90, 110, "CARE")
    p23 = BabyProduct("S023", "BEPANTHEN BABY DIAPER CREAM 30G", 17.90, 70, "SKINCARE")
    p24 = BabyProduct("S024", "SUDOCREM DIAPER RASH CREAM 60G", 24.90, 65, "SKINCARE")
    p25 = BabyProduct("D025", "DRYPERS DRY PANTS XL SIZE 42PCS", 46.90, 80, "DIAPERS")
    p26 = BabyProduct("D026", "PETPET TAPE DIAPERS S SIZE 72PCS", 28.90, 140, "DIAPERS")
    p27 = BabyProduct("D027", "PETPET DAYPANTS XL SIZE 52PCS", 39.90, 90, "DIAPERS")
    p28 = BabyProduct("F028", "PIGEON WIDE NECK TEAT SIZE M", 16.90, 60, "FEEDING")
    p29 = BabyProduct("F029", "PIGEON BREAST PUMP MANUAL", 89.90, 25, "FEEDING")
    p30 = BabyProduct("F030", "TOMMEE TIPSEE MILK POWDER DISPENSER", 19.90, 75, "FEEDING")
    p31 = BabyProduct("C031", "PIGEON BABY NAIL CLIPPER", 15.50, 85, "CARE")
    p32 = BabyProduct("C032", "SAFERCARE BABY NASAL DROPS 28ML", 8.90, 55, "CARE")
    p33 = BabyProduct("H033", "APOLLO BABY FEVER COOLING PATCH 6PCS", 12.90, 90, "HEALTH")
    p34 = BabyProduct("H034", "PUREEN ANTIBACTERIAL WIPES 40S", 6.90, 130, "HEALTH")
    p35 = BabyProduct("T035", "LAMAZE BABY SOFT BOOK", 29.90, 40, "TOYS")
    p36 = BabyProduct("T036", "BABY PLAY MAT FOLDABLE 180CM", 79.90, 20, "TOYS")
    p37 = BabyProduct("C037", "BABY COTTON SWABS 200PCS", 5.90, 160, "CARE")
    p38 = BabyProduct("C038", "PUREEN BABY LAUNDRY DETERGENT 1L", 18.90, 45, "CARE")
    p39 = BabyProduct("B039", "BABY ROMPER SHORT SLEEVE 3PCS", 25.90, 70, "CLOTHING")
    p40 = BabyProduct("B040", "BABY BLANKET 100% COTTON", 18.90, 50, "CLOTHING")
    p41 = BabyProduct("S041", "CALIFORNIA BABY SHAMPOO & BODYWASH 251ML", 49.90, 30, "SKINCARE")
    p42 = BabyProduct("S042", "MUSTELA NO-RINSE CLEANSING WATER 300ML", 44.90, 35, "SKINCARE")
    p43 = BabyProduct("F043", "NAN OPTIPRO INFANT FORMULA 800G", 75.90, 45, "FEEDING")
    p44 = BabyProduct("F044", "ENFAGROW A+ Step 3 FORMULA 1.3KG", 89.90, 55, "FEEDING")
    p45 = BabyProduct("D045", "GENKI TAPE MEGA PACK M SIZE 96PCS", 54.90, 100, "DIAPERS")
    p46 = BabyProduct("D046", "MERRIES TAPE DIAPERS L SIZE 56PCS", 77.90, 45, "DIAPERS")
    p47 = BabyProduct("T047", "STACKING CUPS BABY TOY SET", 14.90, 85, "TOYS")
    p48 = BabyProduct("T048", "BABY TEETHING TOY BANANA SILICONE", 12.90, 95, "TOYS")
    p49 = BabyProduct("C049", "PUREEN BABY OIL 250ML", 10.90, 60, "CARE")
    p50 = BabyProduct("C050", "JOHNSON'S BABY OIL 300ML", 12.90, 70, "CARE")

```

A unique product ID is used as the key for when each product is inserted into the hash table, the keys are also being inserted into the array ascending order the which they are added:

```
products = [
    p1, p2, p3, p4, p5, p6, p7, p8,
    p9, p10, p11, p12, p13, p14, p15, p16,
    p17, p18, p19, p20, p21, p22, p23, p24,
    p25, p26, p27, p28, p29, p30, p31, p32,
    p33, p34, p35, p36, p37, p38, p39, p40,
    p41, p42, p43, p44, p45, p46, p47, p48,
    p49, p50]

# Insert into hash table and array for dual storage #4
for product in products: # Loop through all products
    hashTable.insert(product.productId, product) # Add to hash table
    arrayStorage.append(product) # Add to array

return hashTable, arrayStorage # Return both data structures
```

Output:

Display of all products:

```
==== BABY PRODUCTS RETAIL SHOP ====
1. Insert new product
2. Search product by Product ID
3. Show all products
4. Delete product by Product ID (optional)
5. Performance comparison: Hash Table vs Array (search)
0. Exit
Enter your choice: 5
Enter Product ID to test performance search: h033

==== Performance Test Result ====
Search key: H033

Hash table search result:
H033 - APOLLO BABY FEVER COOLING PATCH 6PCS | RM12.90 | Stock: 90 | Category: HEALTH
2: [D005, D005 - MAMPOKO EXTRA DRY DIAPERS M SIZE 60PCS | RM45.90 | Stock: 120 | Category: DIAPERS] [F028, F028 - PIGEON WIDE NECK TEAT SIZE M | RM16.90 | Stock: 60 | Category: FEEDING] [T035, T035 - LAMAZE BABY SOFT BOOK | RM29.90 | Stock: 40 | Category: TOYS] [B039, B039 - BABY ROMPER SHORT SLEEVE 3PCS | RM25.90 | Stock: 70 | Category: CLOTHING] [F044, F044 - ENFAGROW A+ Ste
p 3 FORMULA 1-3KG | RM89.90 | Stock: 55 | Category: FEEDING]
3: [F007, F007 - AVENT NATURAL FEEDING BOTTLE 260ML | RM34.90 | Stock: 40 | Category: FEEDING] [C009, C009 - DR BROWN'S BABY PACIFIER 2PCS | RM18.90 | Stock: 60 | Category: CARE] [T018, T018 - FISHER-PRICE TEETHING RING | RM15.90 | Stock: 55 | Category: TOYS] [C022, C022 - JOHNSON'S BABY POWDER 500G | RM12.90 | Stock: 110 | Category: CARE] [S023, S023 - BEPANTHEN BABY DIAPER CR
EAM 30G | RM17.90 | Stock: 70 | Category: SKINCARE] [D027, D027 - PETPET DAYPANTS XL SIZE 52PCS | RM39.90 | Stock: 90 | Category: DIAPERS] [D045, D045 - GENKI TAPE MEGA PACK M SIZE 96PCS | R
M54.90 | Stock: 100 | Category: DIAPERS] [T047, T047 - STACKING CUPS BABY TOY SET | RM14.90 | Stock: 85 | Category: TOYS]
4: [D006, D006 - HUGGIES DRY PANTS L SIZE 50PCS | RM39.90 | Stock: 90 | Category: DIAPERS] [S024, S024 - SUDDCREM DIAPER RASH CREAM 60G | RM24.90 | Stock: 65 | Category: SKINCARE] [H033, H033 - APOLLO BABY FEVER COOLING PATCH 6PCS | RM12.90 | Stock: 90 | Category: HEALTH] [S041, S041 - CALIFORNIA BABY SHAMPOO & BODYWASH 251ML | RM49.90 | Stock: 30 | Category: SKINCARE] [S042, S
042 - MUSTELA NO-RINSE CLEANSING WATER 300ML | RM44.90 | Stock: 35 | Category: SKINCARE] [C049, C049 - PUREEN BABY OIL 250ML | RM10.90 | Stock: 60 | Category: CARE]
5: [F008, F008 - TOMMEE TIPPEE CLOSER TO NATURE BOTTLE 150ML | RM29.90 | Stock: 30 | Category: FEEDING] [C010, C010 - PUKU BABY COMB AND BRUSH SET | RM12.50 | Stock: 80 | Category: CARE] [W012, W012 - MAMPOKO BABY WIPES 1005 | RM13.90 | Stock: 140 | Category: CARE] [C021, C021 - PUREEN BABY HEAD-TO-TOE WASH 750ML | RM18.50 | Stock: 90 | Category: CARE] [C032, C032 - SAFERCARE BABY NASAL DROPS 28ML | RM8.90 | Stock: 55 | Category: CARE] [C038, C038 - PUREEN BABY LAUNDRY DETERGENT 1L | RM18.90 | Stock: 45 | Category: CARE]
6: [B002, B002 - PIGEON BABY BOTTLE 240ML | RM24.50 | Stock: 50 | Category: FEEDING]
7: [B020, B020 - BABY MITTENS AND BOOTIES SET | RM7.90 | Stock: 200 | Category: CLOTHING] [D026, D026 - PETPET TAPE DIAPERS S SIZE 72PCS | RM28.90 | Stock: 140 | Category: DIAPERS] [H034, H034 - PUREEN ANTIBACTERIAL WIPES 485 | RM6.90 | Stock: 130 | Category: HEALTH] [T036, T036 - BABY PLAY MAT FOLDABLE 180CM | RM09.90 | Stock: 20 | Category: TOYS] [C037, C037 - BABY COTTON SWA
BS 200PCS | RM5.90 | Stock: 160 | Category: CARE] [C050, C050 - JOHNSON'S BABY OIL 300ML | RM12.90 | Stock: 70 | Category: CARE]
8: [S014, S014 - BUDS ORGANICS BABY SHAMPOO 230ML | RM32.90 | Stock: 50 | Category: SKINCARE] [D025, D025 - DRYPERS DRY PANTS XL SIZE 42PCS | RM46.90 | Stock: 80 | Category: DIAPERS] [F029, F029 - PIGEON BREAST PUMP MANUAL | RM89.90 | Stock: 25 | Category: FEEDING]
9: [H015, H015 - PANADOL CHILDREN SUSPENSION 100ML | RM14.90 | Stock: 60 | Category: HEALTH] [B019, B019 - BABY COTTON BIB 3PCS PACK | RM9.90 | Stock: 120 | Category: CLOTHING] [C031, C031 - PIGEON BABY NAIL CLIPPER | RM15.50 | Stock: 85 | Category: CARE] [F043, F043 - NAN OPTIPRO INFANT FORMULA 800G | RM05.90 | Stock: 45 | Category: FEEDING] [T048, T048 - BABY TEETHING TOY BAN
ANA SILICONE | RM12.90 | Stock: 95 | Category: TOYS]
```

## 4.3 Command line menu ,Insert and search function

The menu will display options that the user can input:

```
#3
def menu():
    print("\n=== BABY PRODUCTS RETAIL SHOP ===")
    print("1. Insert new product")
    print("2. Search product by Product ID")
    print("3. Show all products")
    print("4. Delete product by Product ID (optional)")
    print("5. Performance comparison: Hash Table vs Array (search)")
    print("0. Exit")
    return input("Enter your choice: ")
```

A method for mapping what input does what function is created in the name `inventory_system`:

```
def inventory_system(): # Main system - handles user interactions
    hashTable, arrayStorage = local_storage_system() # Load initial data
    print("Welcome to Baby Shop Inventory System (Hash Table + Array)") # Welcome message
```

For choice one(insert) products can be inserted by entering their ID, product name in full, price Quantity in stock and product category. Product objects will then be inserted into the insert function of the hash table and into the one dimensional array. Next for choice two (search) It were us for the input of product to search for a product the hash table using the method search, It found product details will be displayed and if an error message product not found will be displayed:

```
while True: # Main loop - keep running until exit
    choice = menu() # Display menu and get choice

    if choice == "1": # INSERT NEW PRODUCT
        productId = input("Enter Product ID (e.g., D001): ").upper() # Get ID
        name = input("Enter Full Product Name (e.g., Drypers Wee Wee Diapers): ").upper() # Get name
        price = float(input("Enter Product Price in RM (e.g., 29.90): ")) # Get price
        stock = int(input("Enter Quantity in Stock (e.g., 100): ")) # Get quantity
        category = input("Enter Product Category (e.g., Diapers / Feeding / Care / Skincare): ").upper() # Get category

        new_product = BabyProduct(productId, name, price, stock, category) # Create product object
        hashTable.insert(productId, new_product) # Add to hash table
        arrayStorage.append(new_product) # Add to array
        print("Product inserted successfully.")

    elif choice == "2": # SEARCH PRODUCT
        productId = input("Enter Product ID to search: ").upper() # Get ID to search
        result = hashTable.search(productId) # Search in hash table
        if result: # If found
            print("Product found in hash table:") # Confirmation
            print(result) # Display product details
        else: # If not found
            print("Product NOT found.") # Error message

    elif choice == "3": # DISPLAY ALL PRODUCTS
        print("\nAll products in hash table:") # Header
        hashTable.print_table() # Print table structure

    elif choice == "4": # DELETE PRODUCT
        productId = input("Enter Product ID to delete: ").upper() # Get ID
        deleted = hashTable.delete(productId) # Delete from hash table

        if deleted: # If deletion successful
            # Remove from array as well (for consistency)
            arrayStorage[:] = [p for p in arrayStorage if p.productId != productId] # Filter array
            print("Product deleted from hash table and array.") # Confirm deletion
        else: # If product not found
            print("Product not found; nothing deleted.") # Error message
```

Output:

Insert function:

```
=== BABY PRODUCTS RETAIL SHOP ===
1. Insert new product
2. Search product by Product ID
3. Show all products
4. Delete product by Product ID (optional)
5. Performance comparison: Hash Table vs Array (search)
0. Exit
Enter your choice: 1
Enter Product ID (e.g., D001): 1200
Enter Full Product Name (e.g., Drypers Wee Wee Diapers): crozz
Enter Product Price in RM (e.g., 29.90): 100.90
Enter Quantity in Stock (e.g., 100): 30
Enter Product Category (e.g., Diapers / Feeding / Care / Skincare): toy
Product inserted successfully.
```

Search function:

```
=== BABY PRODUCTS RETAIL SHOP ===
1. Insert new product
2. Search product by Product ID
3. Show all products
4. Delete product by Product ID (optional)
5. Performance comparison: Hash Table vs Array (search)
0. Exit
Enter your choice: 2
Enter Product ID to search: 1200
Product found in hash table:
1200 - CROZZ | RM100.90 | Stock: 30 | Category: TOY

=== BABY PRODUCTS RETAIL SHOP ===
1. Insert new product
2. Search product by Product ID
3. Show all products
4. Delete product by Product ID (optional)
5. Performance comparison: Hash Table vs Array (search)
0. Exit
Enter your choice: █
```

## 4.4 Separate chaining and one dimensional array performance comparison

Choice 5 is for comparing the time performance of separate chaining and one dimensional array:

```
elif choice == "5": # PERFORMANCE TEST
    run_performance_test(hashTable, arrayStorage) # Run comparison test

elif choice == "0": # EXIT PROGRAM
    print("Exiting system...") # Goodbye message
    break # Exit while loop

else: # Invalid choice
    print("Invalid choice, please try again.") # Error message
```

The `run_performance_test` method will first check whether any product data exists, If the product exists then it will continue if not it will be stopped. Product ID wouldn't be asked to enter and the system we'll verify if ID exists or not. When product is found the function will run the performance test iteration is 100,000 times so that the timing noise will be reduced and differences can be more clearer when measured between hash table lookup and linear array search:

```
#4
def run_performance_test(hashTable, arrayStorage): # Compare hash table vs array search performance
    if not arrayStorage: # Check if data exists
        print("No data available for performance test.") # Error if no data
        return # Exit function

    # User chooses the Product ID to test performance
    targetProductId = input("Enter Product ID to test performance search: ").upper() # Get search target

    # Check if the product exists at least once in the array
    exists = any(product.productId == targetProductId for product in arrayStorage) # Verify existence
    if not exists: # If not found
        print("Product ID not found in the system. Cannot run performance test.") # Error message
        return # Exit function

    iterations = 100000 # Repeat searches to measure clearer timing differences

    # Hash table search timing
    startHash = time.perf_counter() # Start timer
    for _ in range(iterations): # Repeat search
        hashTable.search(targetProductId) # Hash table lookup
    endHash = time.perf_counter() # End timer
    hashTime = endHash - startHash # Calculate elapsed time
```

This part of the method Performs the measurement of time for the hashtable and array the program will lookups using `hashTable.search` method and Measure the start and end times using `time.perf_counter`. The rail search use the same number of iterations but each search

will require the array from beginning to the end until the product ID is found:

```
# Array search timing
startArray = time.perf_counter() # Start timer
for _ in range(iterations): # Repeat search
    for product in arrayStorage: # Linear search through array
        if product.productId == targetProductId: # Find match
            break # Exit inner loop when found
endArray = time.perf_counter() # End timer
arrayTime = endArray - startArray # Calculate elapsed time

# Single search to OUTPUT the dataset for this Product ID
hashResult = hashTable.search(targetProductId) # Get product from hash table
arrayResult = None # Initialize
for product in arrayStorage: # Search array
    if product.productId == targetProductId: # Match found
        arrayResult = product # Store result
        break # Exit loop
```

This part of the method will display product that then the total search time for hash table and array. The method will then compare both times and print out the conclusion that explains the structure that is faster:

```
print("\n=== Performance Test Result ===") # Header
print(f"Search key: {targetProductId}\n") # Show search target
print("Hash table search result:") # Label
print(hashResult) # Display product
print(f"Hash table search time: {hashTime:.6f} seconds") # Show hash table time
print("\nArray search result:") # Label
print(arrayResult) # Display product
print(f"Array search time: {arrayTime:.6f} seconds") # Show array time

if hashTime < arrayTime: # HashTime is faster
    print("=> Hash table is faster for searching in this test.(it jumps straight to where the product is stored)")
else: # Array is faster
    print("=> Array is faster in this specific run (variable is in the earlier/front part of the array).")
```

Output:

Performance comparison:

Hash table is faster than array due to hash table directly jumping straight into its product store location, while one-dimensional array has to search from where the search key is inputted.

nt/question1.py

Welcome to Baby Shop Inventory System (Hash Table + Array)

=== BABY PRODUCTS RETAIL SHOP ===

1. Insert new product
2. Search product by Product ID
3. Show all products
4. Delete product by Product ID (optional)
5. Performance comparison: Hash Table vs Array (search)
0. Exit

Enter your choice: 5

Enter Product ID to test performance search: h033

=== Performance Test Result ===

Search key: H033

Hash table search result:

H033 - APOLLO BABY FEVER COOLING PATCH 6PCS | RM12.90 | Stock: 90 | Category: HEALTH

Hash table search time: 0.032673 seconds

Array search result:

H033 - APOLLO BABY FEVER COOLING PATCH 6PCS | RM12.90 | Stock: 90 | Category: HEALTH

Array search time: 0.101526 seconds

=> Hash table is faster for searching in this test.(it jumps straight to where the product is stored)

=== BABY PRODUCTS RETAIL SHOP ===

1. Insert new product
2. Search product by Product ID
3. Show all products
4. Delete product by Product ID (optional)
5. Performance comparison: Hash Table vs Array (search)
0. Exit

Enter your choice: █

## 5 Question 2

### 5.1 unweighted directed graph data structure

- addVertex:  
Will check vertex key exists and then we'll initialize an empty list to add vertex into the graph:

```
#1
def addVertex(self, vertex): # add a vertex to the graph
    if vertex not in self.graph: # check if vertex(key) already exists
        self.graph[vertex] = [] # initialize an empty list for the vertex
```

- addEdge:  
Will check for both vertices if they exist in the graph and we'll add an to that graph, if it does not exist raise an error value:

```
#1
def addEdge(self, from_vertex, to_vertex): #add an edge to the graph
    # check if both vertices exist in the graph
    if from_vertex in self.graph and to_vertex in self.graph:
        self.graph[from_vertex].append(to_vertex)
    else:
        #one or both vertices do not exist
        raise ValueError("One or both vertices not found in the graph.")
```

- listOutgoingAdjacentVertex:  
We list all the neighbors of the vertex which are the edges are going from the vertex:

```
def listOutgoingAdjacentVertex(self, vertex): #return the neighbours of a vertex
    return self.graph.get(vertex,[] ) |
```

### 5.2 Person domain / entity class

The Person class will serve as a entity for representing the user profile details, As it stores the username, full name, gender biography and privacy settings inside of it. all the other all the data are string except for privacy that is boolean to indicate whether the user's profile is public or private:

```
#2
class Person:
    username: str
    name: str
    gender: str
    biography: str
    privacy: bool # True = public, False = private
```

## 5.3 Person object creation

This is a dictionary named `people` that stores the user accounts of the graph. For every key that is a username there is a person that has a value which contains the profile details of the user such as their gender, their full name, whether they are in privacy mode, or or their the code also has a safeguard We'll limit the graph to only a maximum of 10. Using the `addVertex` method the program will add us into the graph as a vertex:

```
#3
people = {
    "erin": Person("erin", "Erin Tan", "F", "Cats & coffee", True),
    "jack": Person("jack", "Jack Lim", "M", "Gaming & streaming", True),
    "dan": Person("dan", "Dan Ho", "M", "Photography", True),
    "fahmi": Person("fahmi", "A. Fahmi", "M", "Running & podcasts", False),
    "haris": Person("haris", "Haris Teh", "M", "Car reviews & lifestyle", True),
    "gina": Person("gina", "Gina Wong", "F", "Art & sketching", True),
}

if len(people) > 10:
    people = dict(list(people.items())[:10])

# add vertices as usernames
for uname in people.keys():
    follow.addVertex(uname)
```

## 5.4 Graph object creation

The `follow` object is created from the `UDGraph` class and By using `addEdge` method we can create a direct connection to show who will follow who for example in the picture below Erin is following Gina or then following Jack And Jack following Dan back as shown mutual following is also not required. With this set of edges it will form a collective follower following structure that allows the graph to store relationships like on social media apps where user has their own list of followers or people that are following:

```
follow = UDGraph()
```

```
# 4 "someone follows someone" (not necessarily mutual)
follow.addEdge("erin", "gina")
follow.addEdge("jack", "dan")
follow.addEdge("dan", "jack")
follow.addEdge("jack", "erin")
follow.addEdge("dan", "gina")
follow.addEdge("fahmi", "erin")
follow.addEdge("haris", "jack")
follow.addEdge("gina", "erin")
follow.addEdge("gina", "haris")
```

## 5.5 mandatory features

This method is to show users their options:

```
#5(mandatory)
def display(self):
    print("*****")
    print("Follow Gram, Who is Following Who Media App:")
    print("*****")
    print("1. View names of all profiles")
    print("2. View details for any profiles")
    print("3. View followers of any profile")
    print("4. View followed accounts of any profile")
    print("5. Add a new profile")
    print("6. Follow someone")
    print("7. Unfollow someone")
    print("8. Quit")
    print("*****")
    choice = input("Enter your choice (1 - 8): ")
    return choice
```

This method is the menu method where there are multiple cases from one to eight. Case one will list all profiles. Case two will show details of the profile from the username, and case 3 asks for the user to enter their ID to view their followers:

```
def menuSelection(self, people, choice):
    choice = str(choice).strip()
    match choice:
        case "1": # (1) View names of all profiles
            print("\nList of all profiles:")
            for uname in sorted(people.keys()):
                print(f"> {uname}")

        case "2": # (2) View details for any profile
            uname = input("\nEnter username to view profile: ").strip()
            p = people.get(uname)
            if uname not in people:
                print("User not found.")
            #6
            else:
                if p.privacy: # True = public
                    print(f"\n@{p.username}")
                    print(f"Name      : {p.name}")
                    print(f"Gender   : {p.gender}")
                    print(f"Privacy  : Public")
                    print(f"Biography: {p.biography}")
                else: # False = private
                    print(f"\n@{p.username}")
                    print("Privacy : Private")

        case "3": # (3) View followers (incoming)
            uname = input("\nEnter username to view followers: ").strip()
            if uname not in people:
                print("User not found.")
            else:
                followers = []
                # Check every user u: if uname is in u's outgoing list, then u follows uname
                for u in sorted(self.graph.keys()):
                    if uname in self.listOutgoingAdjacentVertex(u): # get u's outgoing list
                        followers.append(u)

                print(f"\nFollowers of @{uname} ({len(followers)}):")
                if followers:
                    for u in followers:
                        print(f"{uname} <- {u}")
                else:
                    print("No followers yet.")
```

Case 4 is for users to enter their username to view who they followed:

```
case "4": # (4) View followed accounts (outgoing)
    uname = input("\nEnter username to view followed accounts: ").strip()
    if uname not in people:
        print("User not found.")
    else:
        following = self.listOutgoingAdjacentVertex(uname) # get username's outgoing list
        print(f"\n@{uname} is following ({len(following)}):")
        if following:
            for v in following:
                print(f" {uname} -> {v}")
        else:
            print("This user isn't following anyone yet.")
```

If users have entered incorrectly they will be prompted to enter again, and if the user wants to exit they will have to press number 8:

```
print(e)

case "8": # (8) Quit
    print("\nThank you for using Slow Gram. Goodbye!")
    sys.exit(0)

case _:
    print("\nInvalid choice. Please enter 1 - 8.")

# small pause so CMD users can see output before next menu
input("\nPress Enter to return to the menu...")
```

Displays the menu method and also the display method:

```
# show menu
while True:
    user_choice = follow.display()
    follow.menuSelection(people, user_choice)
```

## 5.6 Additional Features and Output screenshots

### 5.6.1 Additional Features

Case 5 (add profile) References the addVertex method and links to the Person data class constructor to create a new user profile object and then once new user profile object is created it is inserted to the graph as a new vertex:

```
case "5": # (5) Add a new profile #5(optional)
    print("\nAdd new profile:")
    new_username = input("Enter new username: ").strip()
    if not new_username:
        print("Username cannot be empty.")
    elif new_username in people:
        print("That username already exists.")
    else:
        name = input("Enter full name: ").strip()
        gender = input("Enter gender (M/F): ").strip()
        biography = input("Enter biography: ").strip()
        priv_input = input("Make profile public? (y/n): ").strip().lower()
        privacy = True if priv_input == "y" else False
        # create Person and add to dict + graph
        people[new_username] = Person(new_username, name, gender, biography, privacy)
        self.addVertex(new_username)
        print(f"Profile @{new_username} created successfully.")
```

Case 6 (follow someone) will use the check\_has\_edge method to verify if a connection exists and then the addEdge method to create edge to represent its relationship:

```
case "6": # (6) Follow someone #5(optional)
    print("\nFollow someone:")
    follower = input("Enter username of follower: ").strip()
    if follower not in people:
        print("User not found.")
    else:
        print("\nExisting profiles:")
        for uname in sorted(people.keys()):
            if uname != follower:
                print(f"> {uname}")
        followee = input("Enter username to follow: ").strip()
        if followee not in people:
            print("Target user not found.")
        elif followee == follower:
            print("A user cannot follow themselves.")
        else:
            try:
                if self.check_has_edge(follower, followee):
                    print(f"{follower} is already following {followee}.")
                else:
                    self.addEdge(follower, followee)
                    print(f"{follower} now follows {followee}.")
            except ValueError as e:
                print(e)
```

Case 7 (unfollow someone) will retrieve the user's outgoing edges with the `listOutgoingAdjacentVertex` method and then we'll remove the specific relationship that follows using `remove edge` method:

```
case "7": # (7) Unfollow someone #5(optional)
    print("\nUnfollow someone:")
    follower = input("Enter username of follower: ").strip()
    if follower not in people:
        print("User not found.")
    else:
        following = self.listOutgoingAdjacentVertex(follower)
        if not following:
            print(f"{follower} is not following anyone.")
        else:
            print(f"\n{follower} is currently following:")
            for uname in following:
                print(f" > {uname}")
            followee = input("Enter username to unfollow: ").strip()
            if followee not in following:
                print(f"{follower} is not following {followee}.")
            else:
                try:
                    self.remove_edge(follower, followee)
                    print(f"{follower} has unfollowed {followee}.")
                except ValueError as e:
                    print(e)
```

### 5.6.2 Output screenshots

List all profiles:

```
m(\tab> & C:/Users/Amb/AppData/Local/Programs/Python/P...
neDrive/OneDrive - student.newintil.edu.my/Degree/se...
nt/question2.py"
*****
Follow Gram, Who is Folloing Who Media App:
*****
1. View names of all profiles
2. View details for any profiles
3. View followers of any profile
4. View followed accounts of any profile
5. Add a new profile
6. Follow someone
7. Unfollow someone
8. Quit
*****
Enter your choice (1 - 8): 1

List of all profiles:
> dan
> erin
> fahmi
> gina
> haris
> jack

Press Enter to return to the menu...
*****
```

View profile details(public):

```
*****
Follow Gram, Who is Folloing Who Media App:
*****

1. View names of all profiles
2. View details for any profiles
3. View followers of any profile
4. View followed accounts of any profile
5. Add a new profile
6. Follow someone
7. Unfollow someone
8. Quit
*****

Enter your choice (1 - 8): 2

Enter username to view profile: dan

@dan
Name      : Dan Ho
Gender    : M
Privacy   : Public
Biography: Photography

Press Enter to return to the menu...|
```

View profile details (private):

```
*****
Follow Gram, Who is Folloing Who Media App:
*****
1. View names of all profiles
2. View details for any profiles
3. View followers of any profile
4. View followed accounts of any profile
5. Add a new profile
6. Follow someone
7. Unfollow someone
8. Quit
*****
Enter your choice (1 - 8): 2

Enter username to view profile: erin

@erin
Privacy : Private

Press Enter to return to the menu...
```

View followers of profile:

When the username of the profile is entered other usernames that follows the profile will be shown linked to it.

```
*****
Follow Gram, Who is Folloing Who Media App:
*****
1. View names of all profiles
2. View details for any profiles
3. View followers of any profile
4. View followed accounts of any profile
5. Add a new profile
6. Follow someone
7. Unfollow someone
8. Quit
*****
Enter your choice (1 - 8): 3

Enter username to view followers: jack

Followers of @jack (2):
jack <- dan
jack <- haris

Press Enter to return to the menu...
```

View followed accounts of profile:

When the username of the profile is entered other usernames that the profile follows will be shown linked to it.

```
*****
Follow Gram, Who is Folloing Who Media App:
*****
1. View names of all profiles
2. View details for any profiles
3. View followers of any profile
4. View followed accounts of any profile
5. Add a new profile
6. Follow someone
7. Unfollow someone
8. Quit
*****
Enter your choice (1 - 8): 4

Enter username to view followed accounts: haris

@haris is following (1):
  haris -> jack

Press Enter to return to the menu...
```

Add a new profile:

Once required data is successfully can be shown in the list of all profiles.

```
*****
Follow Gram, Who is Folloing Who Media App:
*****

1. View names of all profiles
2. View details for any profiles
3. View followers of any profile
4. View followed accounts of any profile
5. Add a new profile
6. Follow someone
7. Unfollow someone
8. Quit
*****

Enter your choice (1 - 8): 5

Add new profile:
Enter new username: simon
Enter full name: simon mon
Enter gender (M/F): f
Enter biography: Singer Dancer
Make profile public? (y/n): y
Profile @simon created successfully.

Press Enter to return to the menu...
*****
Follow Gram, Who is Folloing Who Media App:
*****

1. View names of all profiles
2. View details for any profiles
3. View followers of any profile
4. View followed accounts of any profile
5. Add a new profile
6. Follow someone
7. Unfollow someone
8. Quit
*****

Enter your choice (1 - 8): 1

List of all profiles:
> dan
> erin
> fahmi
> gina
> haris
> jack
> simon

Press Enter to return to the menu...
```

User following someone:

User will be presented with existing profiles to follow and after following can be seen in the view followed accounts of any profile section.

```
*****
Follow Gram, Who is Folloing Who Media App:
*****
1. View names of all profiles
2. View details for any profiles
3. View followers of any profile
4. View followed accounts of any profile
5. Add a new profile
6. Follow someone
7. Unfollow someone
8. Quit
*****
Enter your choice (1 - 8): 6

Follow someone:
Enter username of follower: dan

Existing profiles:
> erin
> fahmi
> gina
> haris
> jack
> simon
Enter username to follow: simon
dan now follows simon.

Press Enter to return to the menu...
*****
Follow Gram, Who is Folloing Who Media App:
*****
1. View names of all profiles
2. View details for any profiles
3. View followers of any profile
4. View followed accounts of any profile
5. Add a new profile
6. Follow someone
7. Unfollow someone
8. Quit
*****
Enter your choice (1 - 8): 4

Enter username to view followed accounts: dan

@dan is following (3):
dan -> jack
dan -> gina
dan -> simon

Press Enter to return to the menu...
```

User unfollowing someone:

Users will be presented with existing profiles to unfollow and after unfollowing will be removed in the view followed accounts of any profile section.

```
*****
Follow Gram, Who is Following Who Media App:
*****
1. View names of all profiles
2. View details for any profiles
3. View followers of any profile
4. View followed accounts of any profile
5. Add a new profile
6. Follow someone
7. Unfollow someone
8. Quit
*****
Enter your choice (1 - 8): 7

Unfollow someone:
Enter username of follower: dan

dan is currently following:
> jack
> gina
> simon
Enter username to unfollow: simon
dan has unfollowed simon.

Press Enter to return to the menu...
*****
Follow Gram, Who is Following Who Media App:
*****
1. View names of all profiles
2. View details for any profiles
3. View followers of any profile
4. View followed accounts of any profile
5. Add a new profile
6. Follow someone
7. Unfollow someone
8. Quit
*****
Enter your choice (1 - 8): 3

Enter username to view followers: dan

Followers of @dan (1):
dan <- jack
Press Enter to return to the menu...
```

## 6 Question 3

### 6.1 If python is concurrent processing or parallel processing

Python supposedly does not have multithreading; it is running on concurrent processing instead of parallel processing. Python just fakes multithreading by doing processes bit by bit at a time at a rapid pace that is invisible to the naked eye.

### 6.2 Big-O function

The function creates a basic loop between 1 and n where the current iteration of the result is multiplied by consecutive integers each. To compute time complexity, the number of executed primitive operations is counted.

- The value of assigned result = 1 adds to one operation, then loop with add n the counter.
- For each time the body is looped through, then adds one multiplication and one assignment to have 2n operations, once done will return a statement.
- Total will add up to 4n + 4 primitive operations that are linearly proportional to n.

$T(n) = 4n + 4$  after eliminating constants the time complexity is  $O(n)$

```
# Factorial function
def factorial(n):
    result = 1           # 1 primitive operation (assignment)
    for i in range(1, n+1): # Loop runs n times
        result = result * i # 2 primitive operations (multiplication + assignment)
    return result         # 1 primitive operation
```

## 6.3 Multithreading

```
import threading
import time
```

The FactorialThread class will extend threading.Thread (50,100,200), This will lead to the compute a factorial number of each thread while recording its own start and end times. When a thread starts it will stall the time stamp using time.sleep(0.1), that will perform a delay to or mimic IO wait time. After delay, thread will compute the factorial of its assigned value and then record another timestamp once it is finished The timestamps recorded will then later be used for determining how long the thread actually took to compile:

```
# Thread target wrapper that records its own start + end time
class FactorialThread(threading.Thread):
    def __init__(self, n):
        super().__init__()
        self.n = n
        self.start_time = 0
        self.end_time = 0

    def run(self):
        # record when this thread starts its work
        self.start_time = time.time_ns()

        time.sleep(0.1) # simulate waiting for I/O / network / disk
        factorial(self.n)

        # record when this thread finishes its work
        self.end_time = time.time_ns()
```

This method measures how long it will take to compute 50!, 100!, 200! into three separate threads throughout ten rounds of testing. It will run by FactorialThread method and will record the global start time then will start all threads simultaneously. Then the program uses .join to combine the true threats after waiting and then identifies the actual finish by taking the latest end time from the key threats and calculating the total time taken for the round in nanoseconds. The method will then compute and display the average execution time once all ten rounds are completed:

```
# Perform 10 rounds of timing
def multithread_factorial_test():
    rounds = 10
    times = []

    for r in range(1, rounds + 1):
        # Create threads for 50!, 100!, 200!
        t1 = FactorialThread(50)
        t2 = FactorialThread(100)
        t3 = FactorialThread(200)

        # Global start
        global_start = time.time_ns()

        # Start all threads
        t1.start()
        t2.start()
        t3.start()

        # Wait for completion
        t1.join()
        t2.join()
        t3.join()

        # Determine global end = last thread to finish
        global_end = max(t1.end_time, t2.end_time, t3.end_time)

        # Total time T for this round
        T = global_end - global_start
        times.append(T)

        print(f"Round {r}: Time Elapsed (T) = {T} ns")

    # Average time over 10 rounds
    avg_time = sum(times) / rounds
    print("\n-----")
    print("Average Time over 10 rounds:", avg_time, "ns")
    print("-----")

if __name__ == "__main__":
    multithread_factorial_test()
```

## 6.4 Single threading

This method also measures how long it will take to compute 50!, 100!, 200! but sequentially in a single thread across 10 rounds, the method will record a start time then we'll execute the three operations one after the other with a simulated delay followed by the calculations respectively until end time. The end time is then recorded and total time taken will be computed nanoseconds, after all 10 rounds are the method will display the average execution time calculated:

```
0
1
2 def singlethread_factorial_test():
3     rounds = 10
4     times = []
5
6     for r in range(1, rounds + 1):
7         start = time.time_ns()
8
9         time.sleep(0.1) # simulate waiting for I/O / network / disk
10        factorial(50)
11
12        time.sleep(0.1) # simulate waiting for I/O / network / disk
13        factorial(100)
14
15        time.sleep(0.1) # simulate waiting for I/O / network / disk
16        factorial(200)
17
18        end = time.time_ns()
19        T = end - start
20        times.append(T)
21
22        print(f"Round {r}: Time Elapsed (T) = {T} ns")
23
24    # Average time over 10 rounds
25    avg_time = sum(times) / rounds
26    print("\n-----")
27    print("Average Time over 10 rounds:", avg_time, "ns")
28    print("-----")
29
30 if __name__ == "__main__":
31     singlethread_factorial_test()
32
33
```

## 6.5 Discussion and output

### 6.5.1 comparison of steps 3 and 4 (total and average time taken)

Multithreading:

```
part1.py"
Mutlithreading
Round 1: Time Elapsed (T) = 121676300 ns
Round 2: Time Elapsed (T) = 101318400 ns
Round 3: Time Elapsed (T) = 100832500 ns
Round 4: Time Elapsed (T) = 100785500 ns
Round 5: Time Elapsed (T) = 100790000 ns
Round 6: Time Elapsed (T) = 100829600 ns
Round 7: Time Elapsed (T) = 100783600 ns
Round 8: Time Elapsed (T) = 100673200 ns
Round 9: Time Elapsed (T) = 100725300 ns
Round 10: Time Elapsed (T) = 101510900 ns

-----
Average Time over 10 rounds: 102992530.0 ns
```

Singlethreading:

```
hm/lab/assignment/question3part2.py"
Singlethreading
Round 1: Time Elapsed (T) = 301278000 ns
Round 2: Time Elapsed (T) = 300995700 ns
Round 3: Time Elapsed (T) = 301560900 ns
Round 4: Time Elapsed (T) = 301221400 ns
Round 5: Time Elapsed (T) = 301044500 ns
Round 6: Time Elapsed (T) = 300365400 ns
Round 7: Time Elapsed (T) = 301076300 ns
Round 8: Time Elapsed (T) = 301118200 ns
Round 9: Time Elapsed (T) = 300515000 ns
Round 10: Time Elapsed (T) = 301289900 ns

-----
Average Time over 10 rounds: 301046530.0 ns
-----
```

Multithreading is faster than singlethreading due to the use of time.sleep to simulate waiting for IO.

### 6.5.2 Why multithreading did shorten the time taken for this experiment

Because Python does not support true parallel process of CPU bound threads only can one concurrently do it during I urban operations multi threading has the shorter time taken because of a simulated IO delay to perform more realistic testing, thus the tree threats are waiting at the same time instead of waiting sequentially thus causing their delays to overlap, Making the waiting. Be shut across threads while in the single all delays will occur after the other delay is finished.

### 6.5.3 Example situation where multithreading will improve the performance

An example where multithreading will improve performance is in applications that will perform IO related tasks like downloading multiple files from the Internet or reading and writing data at the same time on a disk. Web scrapers Fetch hundreds of thousands of web also benefit from multithreading because most of their requests spend their time waiting for a network response. This allows the thread to wait while other threads can continue progress significantly reducing execution time in total. Or maybe server handling of multiple download requests waiting for database to respond, efficiently allowing many clients to communicate with the server without getting any blockage.

## 7 Challenges faced and personal reflections

Minor:

Realized in question one when I enter values that are not directly corresponding to the id values they would not come out, the solution was to make everything convert to upper case which saves testing time as there is no need for the care of using uppercase or lowercase.

Major:

- When testing question one timing comparison between hash table and one dimensional array search times, I realized that no matter how many tests were made I could not prove that hash table should be a faster timing comparison compared to one dimensional array results, After some research I found out that to have a more accurate timing having set an iteration of 100,000 could help with averaging out random delays and noise, as well as allow for better visibility of time difference to be seen this also makes the result more stable and accurate. From:

```
hashTable.search(targetProductId)
for product in arrayStorage:
    if product.productId == targetProductId:
        break
```

To:

```
227
228     iterations = 100000 # repeat to see clearer timing
229
230     # Hash table search timing
231     startHash = time.perf_counter()
232     for _ in range(iterations):
233         hashTable.search(targetProductId)
234     endHash = time.perf_counter()
235     hashTime = endHash - startHash
236
237     # Array search timing
238     startArray = time.perf_counter()
239     for _ in range(iterations):
240         for product in arrayStorage:
241             if product.productId == targetProductId:
242                 break
243     endArray = time.perf_counter()
244     arrayTime = endArray - startArray
245
```

- While testing the timings of multithreading and singlethreading I found out that singlethreading was actually faster than multithreading but that is due to as stated in 6.1 That Python is not doing parallel processing but concurrent processing. So to make it a more realistic situation I added a line of code to simulate IO waiting and simulate waiting that overlaps across threads.

From:

```
def run(self):  
    # record when this thread starts its work  
    self.start_time = time.time_ns()  
  
    factorial(self.n)  
  
    # record when this thread finishes its work  
    self.end_time = time.time_ns()
```

```
for r in range(1, rounds + 1):  
    start = time.time_ns()  
  
    factorial(50)  
  
    factorial(100)  
  
    factorial(200)  
  
    end = time.time_ns()  
    T = end - start  
    times.append(T)  
  
    print(f"Round {r}: Time Elapsed (T) = {T} ns")
```

To:

```
def run(self):  
    # record when this thread starts its work  
    self.start_time = time.time_ns()  
  
    time.sleep(0.1) # simulate waiting for I/O / network / disk  
    factorial(self.n)  
  
    # record when this thread finishes its work  
    self.end_time = time.time_ns()
```

```
for r in range(1, rounds + 1):
    start = time.time_ns()

    time.sleep(0.1) # simulate waiting for I/O / network / disk
    factorial(50)

    time.sleep(0.1) # simulate waiting for I/O / network / disk
    factorial(100)

    time.sleep(0.1) # simulate waiting for I/O / network / disk
    factorial(200)

    end = time.time_ns()
    T = end - start
    times.append(T)

    print(f"Round {r}: Time Elapsed (T) = {T} ns")
```

## 8 References

Adam (2013). *Why do multiple iterations of a block of code increase average running time so much?* [online] Stack Overflow. Available at:

<https://stackoverflow.com/questions/14529831/why-do-multiple-iterations-of-a-block-of-code-increase-average-running-time-so-much>

Witowski, S. (2022). *How to Benchmark (Python) Code*. [online] Sebastian Witowski. Available at:

<https://switowski.com/blog/how-to-benchmark-python-code>

Stack Overflow. (n.d.). *Multiprocessing vs Threading Python*. [online] Available at:

<https://stackoverflow.com/questions/3044580/multiprocessing-vs-threading-python>.

Serdar Yegulalp (2023). *Python concurrency and parallelism explained*. [online] InfoWorld. Available at:

<https://www.infoworld.com/article/2269314/python-concurrency-and-parallelism-explained.html>.