UNIVERSITI TEKNOLOGI MALAYSIA
Department of Computer Science
Faculty of Computing

DATA STRUCTURE AND ALGORITHMS
ASSIGNMENT 1

Programme	:	Bachelor of Computer Science (*Data Engineering*)
Subject Code	:	SECJ2013
Session-Sem	:	2023/2024-1

Prepared by :
1) LOO JIA CHANG (A22EC0074)
2) GOH JING YANG (A22EC0052)
3) LOW JIE SHENG(A22EC0075)

Section : 02

Group : Nothing

Topic : Inventory Management System

Lecturer : Dr. LIZAWATI BINTI MI YUSUF

Date : 20/12/2023

TABLE OF CONTENTS

1.0 Objective

The main objective of our theme Inventory Management System is to give a structured and effective way to manage information about colorful particulars in stores. The system trials to give a thorough result for effectively managing colorful rudiments of each item in the force through the relinquishment of a well- organized and effective system. It encompasses essential information pertaining to each item.

Data association: To store detailed information about each storehouse object. System designers must efficiently store and manage information related to item number, item, description, quantity, price, and item location.

Sorting Capability: Applies sorting algorithms to organize force lots. The system offers druggies the occasion to sort the force grounded on colorful characteristics in thrusting or descending order.

Search function: Allows druggies to search for specific products in stock. druggies can search for particulars using attributes similar as number, title or product position.

In achieving these pretensions, the purpose of the storehouse operation system is to give a comprehensive and dependable result to manage storehouse information, support effective decision- making and contribute to the effectiveness of store operations.

2.0 Synopsis

The C++ Inventory Management System is a robust operation designed to facilitate efficient organization and retrieval data to inventory. This system effectively manages inventory items, each of which is characterized by various attributes such as item number, name, description, quantity, cost and location. Users can interact with the system through options for sorting and searching based on different criteria.

Inventory Management:

1. Item Number

- A unique identifier is assigned to each item store in the inventory.

2.Item Name

- The name of the item provides a clear and human-readable identification for easy recognition.

3.Description of item

- A brief description of the item provides additional details that help in understanding the characteristics of the product.

4.Quantity of item

- The quantity of each item stored in the inventory, indicating the stock level and helping in inventory control.

5.Cost

- The value of each item in terms of money enables effective financial management and facilitates the analysis of costs.

6. Location at inventory

- The location or storage position of the item within the store

Sorting Mechanisms:

Merge Sort: This algorithm is implemented to sort the inventory in ascending or descending order based on different attributes.

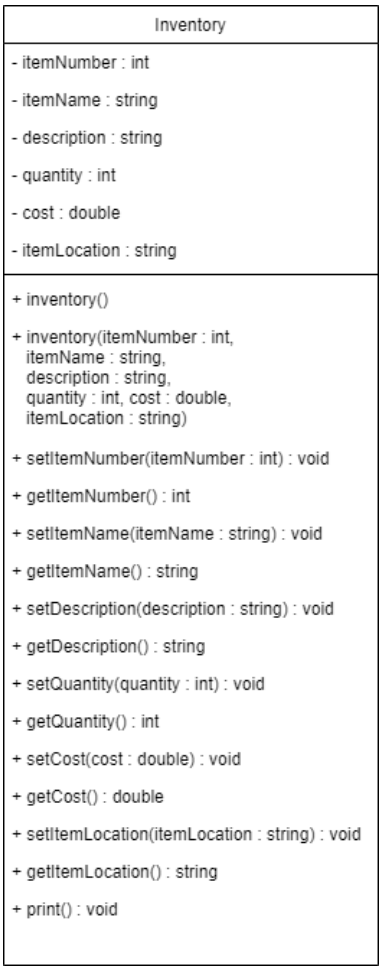
Quicksort: Another sorting algorithm utilized for efficient sorting of inventory items.

Searching Functionality:

Binary Search: Users can search for specific items based on their item number, using the binary search algorithm. This ensures quick and accurate retrieval of information.

Sequential Search: Searching by item name or item location is supported using the sequential search method. It enables users to find specific items within the inventory efficiently.

3.0 Class Diagram



4.0 Pseudo Code

```
Main and other function:

Function main()
1. inventory inv[MAX]
2. string inp
3. Open file "inventory.txt"
   1. If file fails to open, print error message, pause the system,
exit
4. int i = 0
5. While not end of the file, read the inventory data into `inv`
array, increment i.
6. Close the file.
7. Clear the screen, print welcome message.
8. Delay half a second.
9. While true:
   1. user to choose an option from the main menu.
   2. If choice is 1 (Sort), enter a loop to get the sorting criteria
and order.
      1. If the user's choice is valid, sort the inventory based on
the chosen criteria and order using either `mergeSort` or `quickSort`.
      2. Print the sorted inventory in a table format.
   3. If choice is 2 (Search), enter a loop, get the search criteria.
      1. If the user's choice is valid, search the inventory based on
the chosen criteria using either `binarySearch` or `SeqSearch`.
      2. Print the found item or a message indicating that the item
was not found.
   4. If choice is 3 (Exit), break
   5. If choice is invalid, print error message.
10. Clear the console and print exit message.
11. Delay one second.
12. Clear the console and exit the program.
```

```
Function isNumber(string s)
1:     For each character in s:
2:         If the character is not a digit then
3:             Output "Invalid input, enter a numeric input!"
4:             Pause the system
5:             Return false
```

```
6:          End If
7:      End For
8:      Return true
End Function
```

Binary Search:

```
Function binarySearch(inventory inv[], int low, int high, int target)
1. If low > high
   1.1. Return -1
   1.2. End If
2. integer mid = (low + high) / 2
3. If inv[mid].getItemNumber() = target
   3.1. Return mid
   3.2. End If
4. If inv[mid].getItemNumber() > target
   4.1. Return binarySearch(inv, low, mid - 1, target)
   4.2. Else
   4.3. Return binarySearch(inv, mid + 1, high, target)
   4.4. End If
5. End Function
```

Sequential Search:

```
Function SeqSearch(inventory inv[], int n, string target, bool
isLocation)
1. For i, from 0 to n – 1:
   1.1. If isLocation:
       1.1.1. If `inv[i].getItemLocation()` == `target`:
           1.1.1.1. Return `i`.
           1.1.1.2. End If
       1.1.2. End If
   1.2. If isLocation false:
       1.2.1. If `inv[i].getItemName()` equal to `target`:
           1.2.1.1. Return `i`.
           1.2.1.2. End If
       1.2.2. End If
   1.3. End For
2. Return `-1`
3. End Function
```

QuickSort:

Function quickSort(inventory inv[], int first, int last, int choice)

1. If `first < last`:
   1.1. int cut = `partition(inv, first, last, choice)`.
   1.2. Call `quickSort(inv, first, cut, choice)`.
   1.3. Call `quickSort(inv, cut + 1, last, choice)`.
   1.4. End If
2. End Function

Function partition(inventory inv[], int first, int last, int choice)

1. int cutPoint, bottom and top and set them as first and last respectively.
2. inventory temp
3. If choice == 1:
   3.1. pivot = inv[first].getItemNumber()
   3.2. While true:
       3.2.1. While inv[bottom].getItemNumber() > pivot:
           3.2.1.1. Increment `bottom`
           3.2.1.2. End While
       3.2.2. While `inv[top].getItemNumber() < pivot`:
           3.2.2.1. Decrement `top`.
           3.2.2.2. End While
       3.2.3. If `bottom < top`:
           3.2.3.1. Swap `inv[bottom]`, `inv[top]`.
           3.2.3.2. Increment `bottom` and decrement `top`.
           3.2.3.3. Else
           3.2.3.4. `cutPoint` = `top`, break the loop.

            3.2.3.5. End If
        3.2.4. End While
    3.3. End If
4. Repeat step 3 for choice `2`, `3`, `4`, and `5`
5. Return `cutPoint`
6. End Function

Merge sort:

Function mergeSort(inventory inv[], int low, int high, int choice)

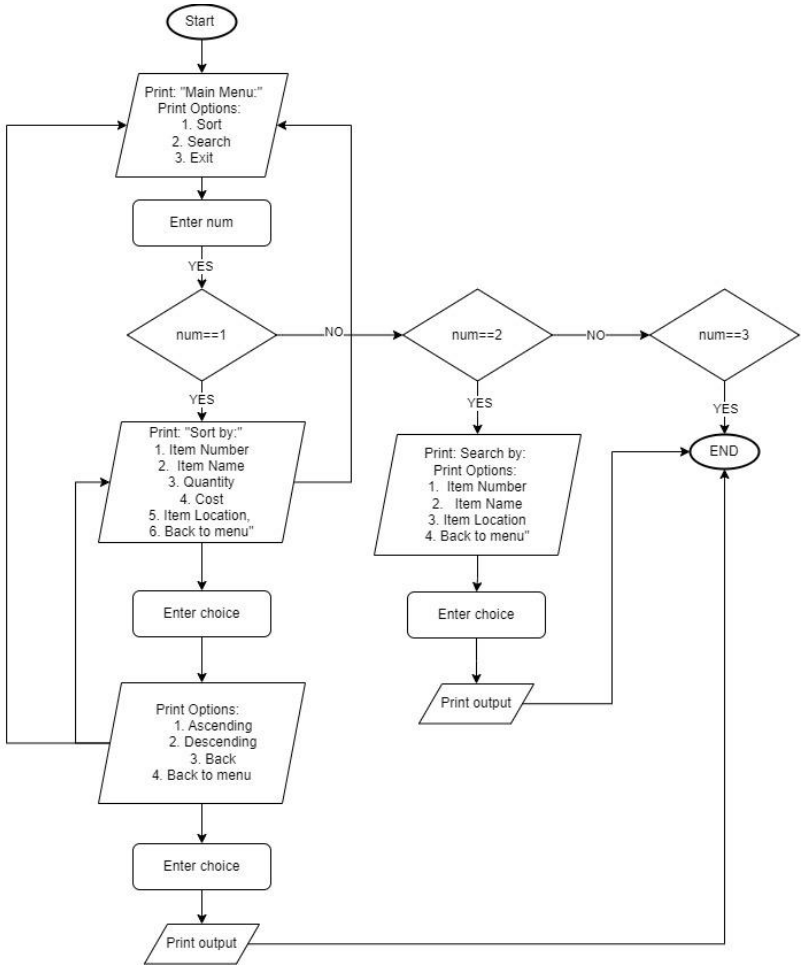
1. If `low < high`
   1.1. int `mid` = `(low + high) / 2`.
   1.2. Call `mergeSort(inv, low, mid, choice)`.
   1.3. Call `mergeSort(inv, mid + 1, high, choice)`.
   1.4. Call `merge(inv, low, mid, high, choice)`.
   1.5. End If
2. End Function

Function merge(inventory inv[], int first, int mid, int last, int choice)

1. int `first1`, `last1`, `first2`, `last2`, `index` = `first`, `mid`, `mid + 1`, `last`, `first` respectively.
2. inventory `temp[MAX]`.
3. If `choice` == `1`:
   3.1. While `first1 <= last1` and `first2 <= last2` :
      3.1.1. If `inv[first1].getItemNumber()` <= `inv[first2].getItemNumber()`:
         3.1.1.1. `temp[index]` = `inv[first1]`.
         3.1.1.2. Increment `first1`.
         3.1.1.3. Else
         3.1.1.4. `temp[index]` = `inv[first2]`.
         3.1.1.5. Increment `first2`.
         3.1.1.6. End If
      3.1.2. Increment `index`.
      3.1.3. End While
   3.2. End If

4. Repeat step 3 for `choice` == 2, 3, 4, 5
5. For first1 to last1:
   5.1. Set `temp[index]` as `inv[first1]`.
   5.2. Increment `first1` and `index`.
   5.3. End For
6. For `first2` to `last2`:
   6.1. Set `temp[index]` as `inv[first2]`.
   6.2. Increment `first2` and `index`.
   6.3. End For
7. For `index` from `first` to `last` do
   7.1. Set `inv[index]` as `temp[index]`.
   7.2. End For
8. End Function

5.0 Flow Chart



6.0 Description of how data structure operations

Sorting Merge Sort: The merge sort algorithm implements sorting the inventory items in ascending and descending order based on the specified attribute.

1. MergeSort Part: The inventory array is recursively divided into two halves until individual elements reach. The mid-point of the array is calculated to determine the split point.

2.Conquer Phase (Merge): The divided subarrays merge in a sorted manner. A temporary array is used to merge the subarrays while maintaining the sorting order.

Quicksort: The quicksort algorithm employs for sorting in descending order. It involves partitioning the array and recursively sorting the subarrays. Key steps include:

1.Partitioning: A pivot element selected from the array. Elements can be rearranged such that elements smaller than the pivot are on the right, and elements greater than the pivot are on the left.

For Partition Function: This function chooses a pivot and rearranges the elements in accordance with that pivot. Smaller elements are moved to the right, and larger elements are moved to the left of the pivot.

Function for Binary Search: The function computes the midpoint and then makes a comparison between the target and the element located at the midpoint. The search is restricted to the left or right subarray if the target equals the mid-point element; otherwise, the index can be returned.

Function for Sequential Search: Iteratively going through the array, the function compares the target with each element until a match is found or the array's end is reached.