K.R. MANGALAM UNIVERSITY

THE COMPLETE WORLD OF EDUCATION

DATA STRUCTURES AND ALGORITHMS

COURSE CODE:- ENCS205

SUBMITTED TO:-

Dr. Swati Gupta

(Associate Professor)

(School of Engineering and technology)

SUBMITTED BY:

Muskan Aggarwal

(RollNo.-2401010155)

(BTech CSE Core)

INDEX

7	**Understand and implement it and Analyze its best and worst case of linear search**	**18**	
8	**Program to implement Inserting and Deleting on Circular Queue**	**19**	
9	**Perform the Bubbe sort.**	**22**	
10	**Binary Search on Array**	**23**	
11	**Merge Sort On Array**	**24**	
12	**Implement Deletion from beginning and end on Circular Linked List**	**26**	

Q1: Write a Python program to create a menu-driven Inventory Stock Management System using functions.

Your program must perform the following tasks:

1. **Insert New Product**
   - Accept product details:
     1. SKU (Stock Keeping Unit)
     2. Product Name
     3. Quantity
   - Ensure that **SKU is unique**. If the SKU already exists, display an appropriate message.
   - Use **exception handling** to validate that quantity entered is a number.

2. **Store Products**
   - Store each product as a **dictionary** containing SKU, product name, and quantity.
   - Maintain all product records inside a **list called `inventory`**.

3. **Display Inventory**
   - Display all stored product records in a **tabular format**.
   - If no products are available, display "Inventory is empty."

4. **Menu-Driven Program**
   - Create a menu with the following options:
     1. Insert New Product
     2. Display Inventory
     3. Exit
   - Continuously prompt the user until they choose to exit.

TEST CASES

Test Case No.	Test Case Description	Input (User Entry)	Expected Behavior
TC01	Insert a valid product	SKU: `P101`Name: `Pencil`Quantity: `50`	Product is inserted successfully.
TC02	Attempt to insert product with duplicate SKU)	SKU: `P101`	rejecting.
TC03	Insert multiple products	SKU: `P102`, `P103`Name: `Pen`, `Eraser`Qty: `20`, `10`	All products added correctly.
TC04	Enter invalid (non-numeric) quantity	SKU: `P104`, Name: `Marker`, Quantity: `ten`	Show error: "Invalid quantity."
TC05	Search product by SKU	Search for SKU: `P101`	Details of product `Pencil` displayed.
TC06	Search product by Name	Search for Name: `Eraser`	Details of product `Eraser` displayed.
TC07	Try inserting a product with empty name	SKU: `P105`, Name: *(blank)*, Quantity: `5`	Error: Product name cannot be empty.
TC08	Insert product with negative quantity	SKU: `P106`, Name: `Notebook`, Quantity: `-10`	Error: Quantity must be positive.
TC9	Delete an existing product	Action: Delete SKU: `P102`	Product `Pen` removed from inventory.
TC10	Display inventory after multiple operations	Products: `P101`, `P103`, `P104` (some updated, some added)	Show updated inventory table with all changes.

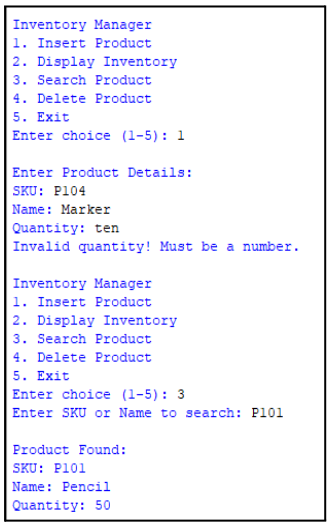
Solution Code In Python

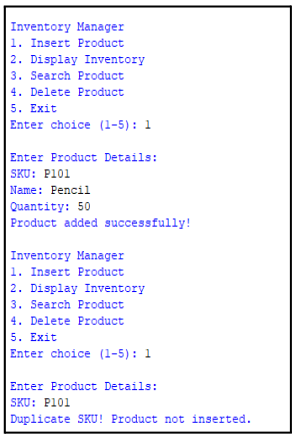
```
inventory = []
def insert_product():
    print("\nEnter Product Details:")
    sku = input("SKU: ").strip()
    if sku == "":
        print("SKU cannot be blank!")
        return
    if any(item['sku'].lower() == sku.lower() for item in inventory):
        print("Duplicate SKU! Product not inserted.")
        return
    name = input("Name: ").strip()
    if name == "":
        print("Product name cannot be blank!")
        return
    try:
        quantity = int(input("Quantity: "))
        if quantity <= 0:
            print("Quantity must be a positive number!")
            return
    except:
        print("Invalid quantity! Must be a number.")
        return
    inventory.append({"sku": sku, "name": name, "quantity": quantity})
    print("Product added successfully!")
def display_inventory():
    if not inventory:
        print("Inventory is empty.")
        return
    print("\nCurrent Inventory:")
    print("SKU\t\tName\t\tQuantity")
    print("----------------------------------------")
    for item in inventory:
        print(f"{item['sku']}\t\t{item['name']}\t\t{item['quantity']}")
    print()
def search_product():
    if not inventory:
        print("Inventory empty!")
        return
    key = input("Enter SKU or Name to search: ").strip().lower()
    found = False
    for item in inventory:
        if item['sku'].lower() == key or item['name'].lower() == key:
            print("\nProduct Found:")
            print(f"SKU: {item['sku']}")
            print(f"Name: {item['name']}")
            print(f"Quantity: {item['quantity']}")
            found = True
    if not found:
        print("No matching product found!")
def delete_product():
    if not inventory:
        print("Inventory empty!")
        return
    sku = input("Enter SKU to delete: ").strip().lower()
    for item in inventory:
```

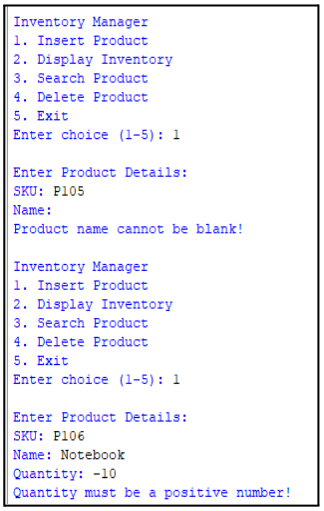
```
        if item['sku'].lower() == sku:
            inventory.remove(item)
            print("Product deleted successfully!")
            return
    print("No product found with that SKU!")
def main():
    while True:
        print("\nInventory Manager")
        print("1. Insert Product")
        print("2. Display Inventory")
        print("3. Search Product")
        print("4. Delete Product")
        print("5. Exit")
        choice = input("Enter choice (1-5): ")
        if choice == '1':
            insert_product()
        elif choice == '2':
            display_inventory()
        elif choice == '3':
            search_product()
        elif choice == '4':
            delete_product()
        elif choice == '5':
            print("Exiting...")
            break
        else:
            print("Invalid choice! Select 1–5.")

main()
```

OUTPUT







Q2:- This program is an **Inventory Stock Manager** that does two main tasks:

1. **Process Sales** – reduces the stock of an item (SKU) when a sale happens.
   - o If stock is enough → reduces it.
   - o If stock is not enough → warns about insufficient stock.
   - o If SKU doesn't exist → shows an error message.
2. **Identify Zero Stock** – checks which items are **out of stock** and lists them for easy Monitoring

Process Sales:

Decrease stock of specific SKUs based on sales, ensuring updates in real time.

Identify Zero Stock:

Detect out-of-stock items and move them to the end for easy monitoring

TEST CASES

Test Case	Input	Expected Output
TC1 – Normal Sale	Inventory: `[(101, 50)]`, SKU: `101`, Qty: `30`	Inventory: `[(101, 20)]`, message: "Sale processed: 30 units of SKU 101."
TC2 – Insufficient Stock	Inventory: `[(102, 20)]`, SKU: `102`, Qty: `25`	Inventory remains `[(102, 20)]`, message: "Insufficient stock for SKU 102. Available: 20"
TC3 – SKU Not Found	Inventory: `[(101, 50)]`, SKU: `104`, Qty: `10`	Inventory remains `[(101, 50)]`, message: "SKU 104 not found in inventory."
TC4 – Zero Stock Detection	Inventory: `[(101, 0), (102, 5), (103, 0)]`	Zero stock SKUs: `[101, 103]`
TC5 – No Zero Stock	Inventory: `[(101, 10), (102, 5)]`	Zero stock SKUs: `[]`, message: "No zero stock items found."
TC6 – Sale Reducing to Zero	Inventory: `[(101, 10)]`, SKU: `101`, Qty: `10`	Inventory: `[(101, 0)]`, message: "Sale processed: 10 units of SKU 101."

---------- Solution Code In Python ----------

```
def process_sale(inventory, sku, qty):
    if sku not in inventory:
        return inventory, f"SKU {sku} not found in inventory."
    if qty <= inventory[sku]:
        inventory[sku] -= qty
        return inventory, f"Sale processed: {qty} units of SKU {sku}."
    else:
        return inventory, f"Insufficient stock for SKU {sku}. Available: {inventory[sku]}"
def find_zero_stock(inventory):
    zero_stock_list = [sku for sku, qty in inventory.items() if qty == 0]
    if zero_stock_list:
        return zero_stock_list, f"Zero stock SKUs: {zero_stock_list}"
    else:
        return [], "No zero stock items found."
def main():
    inventory ={101:50, 102:20,103:0}
    while True:
        print("\n===== Inventory Stock Manager =====")
        print("1. Add/Update Item in Inventory")
        print("2. Process Sale")
        print("3. Check Zero Stock Items")
        print("4. Display Inventory")
        print("5. Exit")
        choice = input("Enter choice (1-5): ")
        if choice == "1":
            try:
                sku = int(input("Enter SKU: "))
                qty = int(input("Enter Quantity: "))
                if qty < 0:
                    print(" Quantity cannot be negative!")
                    continue
                inventory[sku] = qty
                print(" Inventory updated successfully.")

            except ValueError:
                print(" Invalid input. SKU and Quantity must be numbers.")
        elif choice == "2":
            try:
                sku = int(input("Enter SKU for sale: "))
                qty = int(input("Enter quantity to sell: "))
                updated_inventory, message = process_sale(inventory, sku, qty)
                inventory = updated_inventory
                print(message)
            except ValueError:
                print(" Invalid input. SKU and Quantity must be numbers.")
        elif choice == "3":
            zero_list, message = find_zero_stock(inventory)
            print(message)
        elif choice == "4":
            if not inventory:
```

```
            print("Inventory is empty.")
        else:
            print("\nCurrent Inventory:")
            for sku, qty in inventory.items():
                print(f"SKU: {sku}, Qty: {qty}")
    elif choice == "5":
        print("Exiting Program...")
        break
    else:
        print(" Invalid choice. Try again.")
main()
```

---------- OUTPUT ----------

```
===== Inventory Stock Manager =====
1. Add/Update Item in Inventory
2. Process Sale
3. Check Zero Stock Items
4. Display Inventory
5. Exit
Enter choice (1-5): 4

Current Inventory:
SKU: 101, Qty: 20
SKU: 102, Qty: 20
SKU: 103, Qty: 0

===== Inventory Stock Manager =====
1. Add/Update Item in Inventory
2. Process Sale
3. Check Zero Stock Items
4. Display Inventory
5. Exit
Enter choice (1-5): 2
Enter SKU for sale: 101
Enter quantity to sell: 10
Sale processed: 10 units of SKU 101.
```

```
===== Inventory Stock Manager =====
1. Add/Update Item in Inventory
2. Process Sale
3. Check Zero Stock Items
4. Display Inventory
5. Exit
Enter choice (1-5): 2
Enter SKU for sale: 104
Enter quantity to sell: 10
SKU 104 not found in inventory.

===== Inventory Stock Manager =====
1. Add/Update Item in Inventory
2. Process Sale
3. Check Zero Stock Items
4. Display Inventory
5. Exit
Enter choice (1-5): 3
Zero stock SKUs: [103]
```

```
===== Inventory Stock Manager =====
1. Add/Update Item in Inventory
2. Process Sale
3. Check Zero Stock Items
4. Display Inventory
5. Exit
Enter choice (1-5): 2
Enter SKU for sale: 101
Enter quantity to sell: 30
Sale processed: 30 units of SKU 101.

===== Inventory Stock Manager =====
1. Add/Update Item in Inventory
2. Process Sale
3. Check Zero Stock Items
4. Display Inventory
5. Exit
Enter choice (1-5): 2
Enter SKU for sale: 102
Enter quantity to sell: 25
Insufficient stock for SKU 102. Available: 20
```

Q3:-

Application of STACK
Browser Back Button Simulation

Problem Statement:- Modern web browsers allow users to go back to the previous page. We want to **simulate a browser's back button** using a **stack**:

Requirements:

1. User can visit a new webpage → it gets pushed onto the stack.
2. User can press the "Back" button → last visited page is removed from the stack and
shown.
3. User can see the **current page**.

Step	User Input	Action Performed	Expected Outcome
1	1 (Visit Page) Google	Calls visit_page("Google")	Output: Visited: Google history = ["Google"]
2	1 (Visit Page) YouTube	Calls visit_page("YouTube")	Output: Visited: YouTube history = ["Google", "YouTube"]
3	1 (Visit Page) GitHub	Calls visit_page("GitHub")	Output: Visited: GitHub history = ["Google", "YouTube", "GitHub"]
4	3 (Show History)	Calls show_history()	Output: History: Google -> YouTube -> GitHub
5	2 (Back)	Calls go_back()	Output: Going back from: GitHub Current page: YouTube history = ["Google", "YouTube"]
6	3 (Show History)	Calls show_history()	Output: History: Google -> YouTube
7	2 (Back)	Calls go_back()	Output: Going back from: YouTube Current page: Google history = ["Google"]
8	2 (Back)	Calls go_back()	Output: Going back from: Google No pages left in history. history = []
9	3 (Show History)	Calls show_history()	Output: History is empty.
10	4 (Exit)	Ends program	Output: Exiting browser simulation.

—---------- Solution Code In Python ----------

```
history = []
def visit_page(page):
    history.append(page)
    print(f"Visited: {page}")
def go_back():
    if not history:
        print("No pages in history.")
        return
    last_page = history.pop()
    print(f"Going back from: {last_page}")
    if history:
        print(f"Current page: {history[-1]}")
    else:
```

```
        print("No pages left in history.")
def show_history():
    if not history:
        print("History is empty.")
    else:
        print("History:", " -> ".join(history))
while True:
    print("\n1. Visit Page")
    print("2. Back")
    print("3. Show History")
    print("4. Exit")
    choice = input("Enter choice: ")
    if choice == "1":
        page = input("Enter page name: ")
        visit_page(page)
    elif choice == "2":
        go_back()
    elif choice == "3":
        show_history()
    elif choice == "4":
        print("Exiting browser simulation.")
        break
    else:
        print("Invalid choice.")
```

OUTPUT

```
1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 1
Enter page name: Google
Visited: Google

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 1
Enter page name: Youtube
Visited: Youtube

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 1
Enter page name: GitHub
Visited: GitHub

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 3
History: Google -> Youtube -> GitHub
```

```
1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 2
Going back from: GitHub
Current page: Youtube

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 3
History: Google -> Youtube

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 2
Going back from: Youtube
Current page: Google

1. Visit Page
2. Back
3. Show History
4. Exit
Enter choice: 2
Going back from: Google
No pages left in history.
```

Q4:- An online ticketing system manages ticket requests in the order they are received. Each request has a unique request ID. New requests are added to the end of the queue and requests are processed from the front of the queue. Implement a fixed - size linear queue to handle these ticket requests.

OPERATIONS to be performed

1) Insertion (enqueue): Write a function to add a new ticket request to the queue.
2) Deletion (dequeue): Write a function to remove the next ticket request from the queue when it is processed.
3) Size: Write a function to return the number of ticket requests currently in the queue.
4) IsFull: Write a function to check if the queue is full (assuming a fixed-size array implementation).

—---------- SOLUTION CODE IN PYTHON —----------

```
class TicketQueue:
    def __init__(self, capacity):
        self.capacity = capacity
        self.queue = [None] * capacity
        self.front = 0
        self.rear = -1
        self.count = 0
    def enqueue(self, request_id):
        if self.isFull():
            print("Queue is full. Cannot add new ticket request.")
            return
        self.rear += 1
        self.queue[self.rear] = request_id
        self.count += 1
        print(f"Ticket Request {request_id} added successfully.")
    def dequeue(self):
        if self.isEmpty():
            print("Queue is empty. No ticket request to process.")
            return
        removed = self.queue[self.front]
        print(f"Processed Ticket Request: {removed}")
        for i in range(1, self.count):
            self.queue[i - 1] = self.queue[i]
        self.queue[self.count - 1] = None
        self.rear -= 1
        self.count -= 1
    def size(self):
        return self.count
    def isFull(self):
        return self.count == self.capacity
```

```
    def isEmpty(self):
        return self.count == 0
    def display(self):
        if self.isEmpty():
            print("Queue is empty.")
        else:
            print("Current Ticket Queue:", end=" ")
            for i in range(self.count):
                print(self.queue[i], end=" ")
            print()
def main():
    print("=== Online Ticket Request Queue System ===")
    capacity = int(input("Enter queue capacity: "))
    tq = TicketQueue(capacity)
    while True:
        print("\n1. Enqueue (Add Ticket Request)")
        print("2. Dequeue (Process Ticket Request)")
        print("3. Queue Size")
        print("4. Check if Queue is Full")
        print("5. Check if Queue is Empty")
        print("6. Display Queue")
        print("7. Exit")
        choice = input("Enter your choice: ")
        if choice == '1':
            request_id = input("Enter Ticket Request ID: ")
            tq.enqueue(request_id)
        elif choice == '2':
            tq.dequeue()
        elif choice == '3':
            print("Number of ticket requests in queue:", tq.size())
        elif choice == '4':
            print("Queue Full?" , tq.isFull())
        elif choice == '5':
            print("Queue Empty?" , tq.isEmpty())
        elif choice == '6':
            tq.display()
        elif choice == '7':
            print("Exiting Program.")
            break
        else:
            print("Invalid choice. Try again.")
main()
```

OUTPUT

```
1. Enqueue (Add Ticket Request)
2. Dequeue (Process Ticket Request)
3. Queue Size
4. Check if Queue is Full
5. Check if Queue is Empty
6. Display Queue
7. Exit
Enter your choice: 4
Queue Full? False

1. Enqueue (Add Ticket Request)
2. Dequeue (Process Ticket Request)
3. Queue Size
4. Check if Queue is Full
5. Check if Queue is Empty
6. Display Queue
7. Exit
Enter your choice: 6
Current Ticket Queue: 55
```

```
1. Enqueue (Add Ticket Request)
2. Dequeue (Process Ticket Request)
3. Queue Size
4. Check if Queue is Full
5. Check if Queue is Empty
6. Display Queue
7. Exit
Enter your choice: 4
Queue Full? True

1. Enqueue (Add Ticket Request)
2. Dequeue (Process Ticket Request)
3. Queue Size
4. Check if Queue is Full
5. Check if Queue is Empty
6. Display Queue
7. Exit
Enter your choice: 5
Queue Empty? False
```

Q5:- Program to perform following functionality on Single Linked List

1. Create a Node
2. Delete from beginning
3. Delete from end

—----------Solution Code In Python----------

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class LinkedList:
    def __init__(self):
        self.head = None
    def insert(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        temp = self.head
        while temp.next:
            temp = temp.next
        temp.next = new_node
    def delete_from_beginning(self):
        if self.head is None:
            print("List is empty!")
            return
        self.head = self.head.next
        print("Node deleted from beginning.")
    def delete_from_end(self):
        if self.head is None:
            print("List is empty!")
            return
        if self.head.next is None:
            self.head = None
            print("Last node deleted.")
            return
        temp = self.head
        while temp.next.next:
            temp = temp.next
```

```
        temp.next = None
        print("Node deleted from end.")
    def display(self):
        if self.head is None:
            print("List is empty!")
            return
        temp = self.head
        while temp:
            print(temp.data, end=" -> ")
            temp = temp.next
        print("None")
ll = LinkedList()
ll.insert(10)
ll.insert(20)
ll.insert(30)
ll.insert(40)
print("Original List:")
ll.display()
ll.delete_from_beginning()
print("After deleting from beginning:")
ll.display()
ll.delete_from_end()
print("After deleting from end:")
ll.display()
```

OUTPUT

```
Original List:
10 -> 20 -> 30 -> 40 -> None
Node deleted from beginning.
After deleting from beginning:
20 -> 30 -> 40 -> None
Node deleted from end.
After deleting from end:
20 -> 30 -> None
```

Q6:-Program: Insertion in a Circular Linked List (Beginning and End)

Operations To Be Performed:

1. Create a Circular Linked List **2. Insert nodes at the end**

3. Insert a node at the beginning **4. Display the list after each operation**

—---------- Solution Code In Python —----------

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class CircularLinkedList:
    def __init__(self):
        self.head = None
    def insert_at_end(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            new_node.next = self.head
            return
        temp = self.head
        while temp.next != self.head:
            temp = temp.next
        temp.next = new_node
        new_node.next = self.head
    def insert_at_beginning(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            new_node.next = self.head
            return
        temp = self.head
        while temp.next != self.head:
            temp = temp.next
        new_node.next = self.head
        temp.next = new_node
        self.head = new_node
    def display(self):
        if self.head is None:
            print("List is empty")
```

```
            return
        temp = self.head
        while True:
            print(temp.data, end=" -> ")
            temp = temp.next
            if temp == self.head:
                break
        print("(back to head)")
cll = CircularLinkedList()
cll.insert_at_end(10)
cll.insert_at_end(20)
cll.insert_at_end(30)
print("Original list:")
cll.display()
cll.insert_at_beginning(5)
print("\nAfter inserting 5 at beginning:")
cll.display()
cll.insert_at_end(40)
print("\nAfter inserting 40 at end:")
cll.display()
```

OUTPUT

```
Original list:
10 -> 20 -> 30 -> (back to head)

After inserting 5 at beginning:
5 -> 10 -> 20 -> 30 -> (back to head)

After inserting 40 at end:
5 -> 10 -> 20 -> 30 -> 40 -> (back to head)
```

Q7:- Understand the concept of Linear Search, perform its implementation and Analyze its best and worst case

—---------- Solution Code In C++ —----------

```
#include <iostream>
using namespace std;
int main(){
   int arr[10],i,num,index=-1;
   cout<<"Enter 10 Numbers: ";
   for(i=0;i<10;i++){
      cin>>arr[i];
   }
   cout<<"\nEnter a number to search: ";
   cin>>num;
   for(i=0;i<10;i++){
      if(arr[i]==num){
         index=i;
         break;
      }
   }
   cout<<"\nFound at index No."<<index;
   cout<<endl;
   return 0;
}
```

—---------- OUTPUT ----------

```
Enter 10 Numbers: 1 2 3 4 5 6 7 8 9 10

Enter a number to search: 5

Found at index No.4


=== Code Execution Successful ===
```

Q8:- Write a program to implement Circular Queue by performing following operations:

1. **Inserting an Element**
2. **Deleting an Element**

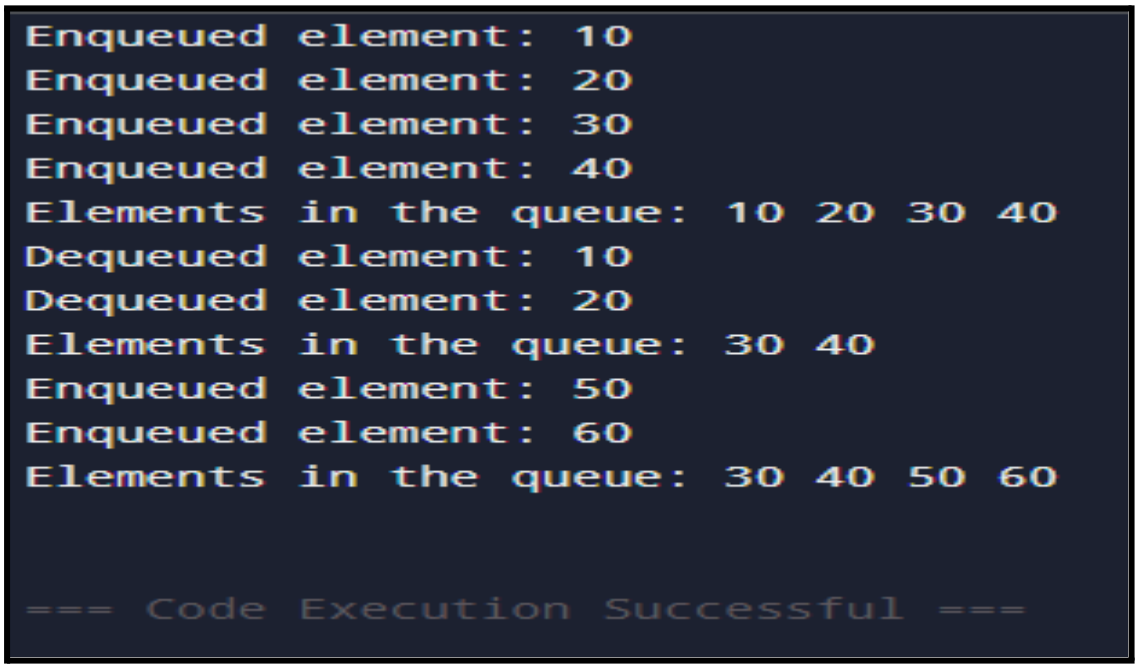
—---------- Solution Code In C++ ----------

```
#include<iostream>
#define MAX_SIZE 100
using namespace std;
class CircularQueue{
   private:
   int front, rear;
   int arr[MAX_SIZE];
   public:
   CircularQueue (){
      front = -1;
      rear = -1;
   }
   bool isFull (){
      if ((front == 0 && rear == MAX_SIZE - 1)
      || (rear == (front - 1) % (MAX_SIZE - 1))){
         return true;
      }
      return false;
   }
   bool isEmpty (){
      if (front == -1){
         return true;
      }
      return false;
   }
   void enQueue (int value){
      if (isFull ()){
         cout << "Queue is full." << endl;
      }
```

```
    else{
        if (front == -1){
            front = 0;
        }
                rear = (rear + 1) % MAX_SIZE;
                arr[rear] = value;
                cout << "Enqueued element: " << value << endl;
    }
}
int deQueue (){
    int element;
    if (isEmpty ()){
        cout << "Queue is empty." << endl;
        return -1;
    }
    else{
        element = arr[front];
        if (front == rear){
            front = -1;
            rear = -1;
        }
                else{
                    front = (front + 1) % MAX_SIZE;
                }
                cout << "Dequeued element: " << element << endl;
            return element;
    }
}
void display (){
    if (isEmpty ()){
        cout << "Queue is empty." << endl;
    }
    else{
        cout << "Elements in the queue: ";
                int i;
                for (i = front; i != rear; i = (i + 1) % MAX_SIZE){
                    cout << arr[i] << " ";
                }
```

```
            cout << arr[i] << endl;
        }
    }
};
int main ()
{
  CircularQueue q;
  q.enQueue (10);
  q.enQueue (20);
  q.enQueue (30);
  q.enQueue (40);
  q.display ();
  q.deQueue ();
  q.deQueue ();
  q.display ();
  q.enQueue (50);
  q.enQueue (60);
  q.display ();
  return 0;
}
```

—---------- OUTPUT —----------



Q9:- Performing the Bubbe sort.

—---------- Solution Code In C++ —----------

```
#include <bits/stdc++.h>
using namespace std;
void bubbleSort(int arr[], int n){
    int i, j;
    for (i=0; i<n-1; i++){
        for (j=0; j<n-i-1; j++){
            if (arr[j]> arr[j + 1]) swap(arr[j], arr[j + 1]);
        }
    }
}
void printArray(int arr[], int size) {
    int i;
    for (i = 0; i < size; i++){
        cout << arr[i] <<" ";
    cout << endl;
    }
}
int main(){
    int arr[]={5,1,4,2,8};
    int N=sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr,N);
    cout<<"Sorted Array: \n";
    printArray(arr,N);
    return 0;
}
```

—---------- OUTPUT —----------

```
Sorted Array:
1
2
4
5
8


=== Code Execution Successful ===
```

Q10:- : Implement Binary Search on Array.

—---------- Solution Code In C++ —----------

```
#include <bits/stdc++.h>
using namespace std;
int binarySearch(int array[], int x,int low,int high){
    if(high>=low){
        int mid=low+(high-low)/2;
        if (array[mid]==x){
            return mid;
        }
        if (array[mid]>x){
            return binarySearch(array,x,low,mid-1);
        }
        return binarySearch(array,x,mid+1,high);
    }
    return -1;
}
int main(void){
    int array[]={3,4,5,6,7,8,9};
    int x=4;
    int n=sizeof(array)/sizeof(array[0]);
    int result=binarySearch(array,x,0,n-1);
    if(result==-1){
        printf("Not Found");
    }else{
        printf("Element is found at index %d", result);
    }
}
```

—---------- OUTPUT —----------

```
Element is found at index 1

=== Code Execution Successful ===
```

Q11:- Perform Merge Sort on Array.

—---------- Solution Code In Python —----------

```
def merge(arr, left, mid, right):
    L = arr[left:mid + 1]
    R = arr[mid + 1:right + 1]
    i = j = 0
    k = left
    while i < len(L) and j < len(R):
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1
    while i < len(L):
        arr[k] = L[i]
        i += 1
        k += 1
    while j < len(R):
        arr[k] = R[j]
        j += 1
        k += 1
def merge_sort(arr, left, right):
    if left >= right:
        return
    mid = (left + right) // 2
    merge_sort(arr, left, mid)
    merge_sort(arr, mid + 1, right)
    merge(arr, left, mid, right)
n = int(input("Enter number of elements: "))
arr = []
print("Enter elements:")
for _ in range(n):
    arr.append(int(input()))
merge_sort(arr, 0, n - 1)
print("Sorted array:", arr)
```

OUTPUT

```
Enter number of elements: 9
Enter elements:
5
2
7
55
11
99
66
0
666
Sorted array: [0, 2, 5, 7, 11, 55, 66, 99, 666]
```

Q12:- To implement a Circular Linked List in Python and perform:

a)Deletion of a node at the beginning.

b).Deletion of a node at the end.

—---------- Solution Code In Python ----------

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class CircularLinkedList:
    def __init__(self):
        self.head = None
    def insert(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            self.head.next = self.head
        else:
            temp = self.head
            while temp.next != self.head:
                temp = temp.next
            temp.next = new_node
            new_node.next = self.head
    def display(self):
        if self.head is None:
            print("List is empty")
            return
        temp = self.head
        print("Circular Linked List:", end=" ")
        while True:
            print(temp.data, end=" -> ")
            temp = temp.next
            if temp == self.head:
                break
        print("(back to head)")
    def delete_begin(self):
        if self.head is None:
            print("List is empty, nothing to delete.")
```

```
            return
        temp = self.head
        if self.head.next == self.head:
            self.head = None
            print("Deleted the only node in the list.")
            return
        last = self.head
        while last.next != self.head:
            last = last.next
        last.next = self.head.next
        self.head = self.head.next
        print("Node deleted from beginning.")
    def delete_end(self):
        if self.head is None:
            print("List is empty, nothing to delete.")
            return
        if self.head.next == self.head:
            self.head = None
            print("Deleted the only node in the list.")
            return
        prev = None
        temp = self.head
        while temp.next != self.head:
            prev = temp
            temp = temp.next
        prev.next = self.head
        print("Node deleted from end.")
cll = CircularLinkedList()
cll.insert(10)
cll.insert(20)
cll.insert(30)
cll.insert(40)
print("Initial List:")
cll.display()
cll.delete_begin()
cll.display()
cll.delete_end()
cll.display()
```

OUTPUT

```
Initial List:
Circular Linked List: 10 -> 20 -> 30 -> 40 -> (back to head)
Node deleted from beginning.
Circular Linked List: 20 -> 30 -> 40 -> (back to head)
Node deleted from end.
Circular Linked List: 20 -> 30 -> (back to head)
```