LINEAR SEARCH:

PROGRAM:

```
def linear_search(arr, target):
    for i, value in enumerate(arr):
        if value == target:
            return i  # Return the index of the target element
    return -1  # Return -1 if target is not found
input_list = list(map(int, input("Enter numbers (separated by space): ").split()))
target = int(input("Enter the number to search for: "))
index = linear_search(input_list, target)
if index != -1:
    print(f"Element found at index: {index}")
else:
    print("Element not found in the list.")
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
Enter numbers (separated by space): 5 8 4 2 3
Enter the number to search for: 2
Element found at index: 3
```

BINARY SEARCH:

PROGRAM:

```
def binary_search(arr, target):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
input_list = list(map(int, input("Enter numbers to sort and search (separated by space): ").split()))
target = int(input("Enter the number to search for: "))
input_list.sort()
index = binary_search(input_list, target)
print("Sorted list:", input_list)
if index != -1:
    print(f"Element found at index: {index}")
else:
    print("Element not found in the list.")
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
Enter numbers to sort and search (separated by space): 5 6 4 2 1
Enter the number to search for: 1
Sorted list: [1, 2, 4, 5, 6]
Element found at index: 0
```

FIBONACCI SEARCH:

PROGRAM:

```
def fibonacci_search(arr, target):
    fib2 = 0
    fib1 = 1
    fib = fib2 + fib1
    n = len(arr)
    while fib < n:
        fib2, fib1 = fib1, fib
        fib = fib2 + fib1
    offset = -1
    while fib > 1:
        i = min(offset + fib2, n - 1)
        if arr[i] < target:
            fib, fib1 = fib1, fib2
            fib2 = fib - fib1
            offset = i
        elif arr[i] > target:
            fib, fib1 = fib2, fib1 - fib2
            fib2 = fib - fib1
        else:
            return i
    if fib1 and offset < n - 1 and arr[offset + 1] == target:
        return offset + 1
    return -1
input_list = list(map(int, input("Enter numbers to sort and search :").split()))
target = int(input("Enter the number to search for: "))
input_list.sort()
index = fibonacci_search(input_list, target)
```

```
print("Sorted list:", input_list)
if index != -1:
    print(f"Element found at index: {index}")
else:
    print("Element not found in the list.")
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
Enter numbers to sort and search :5 7 8 6 2
Enter the number to search for: 2
Sorted list: [2, 5, 6, 7, 8]
Element found at index: 0
>>>
```

INSERTION SORT:

PROGRAM:

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
arr = list(map(int, input("Enter an array of integers : ").split()))
insertion_sort(arr)
print("Sorted array is:", arr)
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
Enter an array of integers : 6 5 8 2 4
Sorted array is: [2, 4, 5, 6, 8]
>>>
```

SELECTION SORT:

PROGRAM:

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]
arr = list(map(int, input("Enter an array of integers : ").split()))
selection_sort(arr)
print("Sorted array is:", arr)
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
Enter an array of integers : 5 4 8 6 2
Sorted array is: [2, 4, 5, 6, 8]
>>>
```

BUBBLE SORT:

PROGRAM:

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
arr = list(map(int, input("Enter an array of integers (space-separated): ").split()))
bubble_sort(arr)
print("Sorted array is:", arr)
```

OUTPUT:

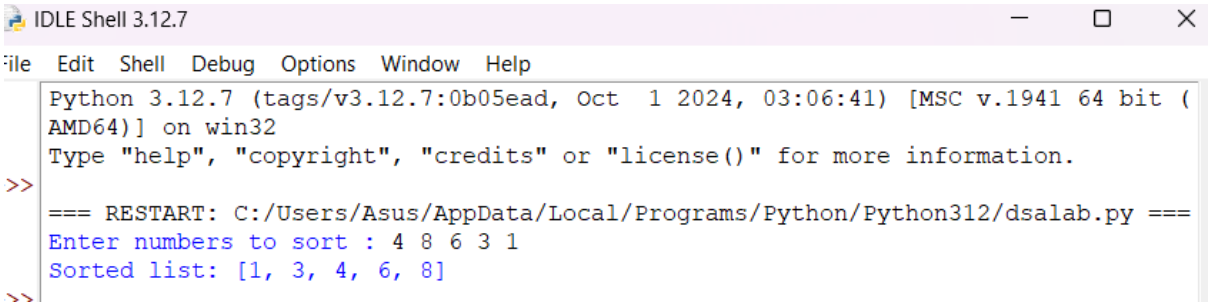
```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
Enter an array of integers (space-separated): 5 4 8 6 2
Sorted array is: [2, 4, 5, 6, 8]
>>
```

QUICK SORT:

PROGRAM:

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)
input_list = list(map(int, input("Enter numbers to sort : ").split()))
sorted_list = quick_sort(input_list)
print("Sorted list:", sorted_list)
```

OUTPUT:


IDLE Shell 3.12.7
File Edit Shell Debug Options Window Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct 1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
Enter numbers to sort : 4 8 6 3 1
Sorted list: [1, 3, 4, 6, 8]
>>

HEAP SORT:

PROGRAM

```
import heapq
input_list = list(map(int, input("Enter numbers to sort : ").split()))
heapq.heapify(input_list)  # Transform list into a heap
sorted_list = [heapq.heappop(input_list) for _ in range(len(input_list))]
print("Sorted list:", sorted_list):
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
Enter numbers to sort : 5 4 8 6 1
Sorted list: [1, 4, 5, 6, 8]
>>>
```

MERGE SORT:

PROGRAM:

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)
def merge(left, right):
    sorted_list = []
    while left and right:
        if left[0] < right[0]:
            sorted_list.append(left.pop(0))
        else:
            sorted_list.append(right.pop(0))
    sorted_list.extend(left or right)
    return sorted_list
input_list = list(map(int, input("Enter numbers to sort : ").split()))
sorted_list = merge_sort(input_list)
print("Sorted list:", sorted_list)
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
Enter numbers to sort : 5 4 8 6 21 1
Sorted list: [1, 4, 5, 6, 8, 21]
```

STACK:

PROGRAM:

```
class Stack:
    def __init__(self):
        self.stack = []
    def push(self, item):
        self.stack.append(item)
        print(f"Item '{item}' pushed to stack.")
    def pop(self):
        if not self.is_empty():
            item = self.stack.pop()
            print(f"Item '{item}' popped from stack.")
            return item
        else:
            print("Stack is empty. Cannot pop.")
            return None
    def peek(self):
        if not self.is_empty():
            print(f"Top item is '{self.stack[-1]}'.")
            return self.stack[-1]
        else:
            print("Stack is empty.")
            return None
    def is_empty(self):
        return len(self.stack) == 0
    def display(self):
        print("Current stack:", self.stack)
if __name__ == "__main__":
    stack = Stack()
```

```
while True:
    print("\nChoose an operation:")
    print("1. Push")
    print("2. Pop")
    print("3. Peek")
    print("4. Display stack")
    print("5. Exit")
    choice = input("Enter your choice (1-5): ")
    if choice == '1':
        item = input("Enter the item to push: ")
        stack.push(item)
    elif choice == '2':
        stack.pop()
    elif choice == '3':
        stack.peek()
    elif choice == '4':
        stack.display()
    elif choice == '5':
        print("Exiting program.")
        break
    else:
        print("Invalid choice. Please enter a number between 1 and 5.")
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===

Choose an operation:
1. Push
2. Pop
3. Peek
4. Display stack
5. Exit
Enter your choice (1-5): 1
Enter the item to push: 8
Item '8' pushed to stack.

Choose an operation:
1. Push
2. Pop
3. Peek
4. Display stack
5. Exit
Enter your choice (1-5): 1
Enter the item to push: 17
Item '17' pushed to stack.

Choose an operation:
1. Push
2. Pop
3. Peek
4. Display stack
5. Exit
Enter your choice (1-5): 5
Exiting program.
>>>
```

POSTFIX:

PROGRAM:

```
class Stack:
    def __init__(self):
        self.stack = []
    def push(self, item):
        self.stack.append(item)
    def pop(self):
        if not self.is_empty():
            return self.stack.pop()
        else:
            return None
    def is_empty(self):
        return len(self.stack) == 0
def evaluate_postfix(expression):
    stack = Stack()
    for char in expression:
        if char.isdigit():
            stack.push(int(char))
        else:
            val1 = stack.pop()
            val2 = stack.pop()
            if val1 is None or val2 is None:
                raise ValueError("Insufficient operands in the expression.")
            if char == '+':
                stack.push(val2 + val1)
            elif char == '-':
                stack.push(val2 - val1)
            elif char == '*':
```

```
            stack.push(val2 * val1)
        elif char == '/':
            if val1 == 0:
                raise ValueError("Division by zero is undefined.")
            stack.push(val2 // val1)
        else:
            raise ValueError(f"Unknown operator '{char}' encountered.")
    result = stack.pop()
    if not stack.is_empty():
        raise ValueError("The expression has too many operands.")
    return result
if __name__ == "__main__":
    postfix_expr = input("Enter a postfix expression: ")
    try:
        result = evaluate_postfix(postfix_expr)
        print(f"Result of postfix evaluation: {result}")
    except ValueError as e:
        print(f"Error: {e}")
```

OUTPUT :

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
Enter a postfix expression: 36+3*
Result of postfix evaluation: 27
>>>
```

QUEUE:

PROGRAM:

```
from collections import deque
def main():
    queue = deque()
    while True:
        user_input = input("Enter an element (or 'q' to quit): ")
        if user_input.lower() == 'q':
            break
        queue.append(user_input)
    print("\nElements dequeued from the queue:")
    while queue:
        print(queue.popleft())
if __name__ == "__main__":
    main()
```

Output:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
Enter an element (or 'q' to quit): d
Enter an element (or 'q' to quit): s
Enter an element (or 'q' to quit): a
Enter an element (or 'q' to quit): l
Enter an element (or 'q' to quit): a
Enter an element (or 'q' to quit): b
Enter an element (or 'q' to quit): q

Elements dequeued from the queue:
d
s
a
l
a
b
>>
```

CIRCULAR:

PROGRAM:

```
class CircularQueue:
    def __init__(self, size):
        self.maxSize = size
        self.queueArray = [None] * size
        self.front = -1
        self.rear = -1
    def enqueue(self, item):
        if (self.rear + 1) % self.maxSize == self.front:
            print("The circular queue is full. Cannot enqueue more items.")
        elif self.front == -1:
            self.front = 0
            self.rear = 0
            self.queueArray[self.rear] = item
            print(f"Enqueued: {item}")
        else:
            self.rear = (self.rear + 1) % self.maxSize
            self.queueArray[self.rear] = item
            print(f"Enqueued: {item}")
    def dequeue(self):
        if self.front == -1:
            print("The circular queue is empty. Cannot dequeue any items.")
            return None
        item = self.queueArray[self.front]
        if self.front == self.rear:  # Only one element left
            self.front = -1
            self.rear = -1
        else:
```

```
            self.front = (self.front + 1) % self.maxSize

        return item
def main():
    queue_size = int(input("Enter the size of the circular queue: "))
    queue = CircularQueue(size=queue_size)
    while True:
        user_input = input("Enter a value to enqueue (or 'q' to quit): ")
        if user_input.lower() == 'q':
            break
        queue.enqueue(user_input)
    print("Circular Queue elements:")
    while True:
        item = queue.dequeue()
        if item is None:
            break
        print(item, end=", ")
    print("\nQueue is now empty.")

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

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
Enter the size of the circular queue: 6
Enter a value to enqueue (or 'q' to quit): d
Enqueued: d
Enter a value to enqueue (or 'q' to quit): s
Enqueued: s
Enter a value to enqueue (or 'q' to quit): a
Enqueued: a
Enter a value to enqueue (or 'q' to quit): l
Enqueued: l
Enter a value to enqueue (or 'q' to quit): a
Enqueued: a
Enter a value to enqueue (or 'q' to quit): b
Enqueued: b
Enter a value to enqueue (or 'q' to quit): q
Circular Queue elements:
d, s, a, l, a, b, The circular queue is empty. Cannot dequeue any items.

Queue is now empty.
>>>
```

PRIORITY:

PROGRAM:

```
import heapq
priority_queue = []
def insert(element, priority):
    heapq.heappush(priority_queue, (priority, element))
    print(f"Inserted: {element} with priority {priority}")
def remove_highest_priority():
    if priority_queue:
        element = heapq.heappop(priority_queue)
        print(f"Removed element with highest priority: {element[1]} (priority: {element[0]})")
    else:
        print("The queue is empty.")
def display_queue():
    if priority_queue:
        print("Current Priority Queue:")
        for priority, element in priority_queue:
            print(f"Element: {element}, Priority: {priority}")
    else:
        print("The queue is empty.")
if __name__ == "__main__":
    while True:
        print("\nPriority Queue Operations:")
        print("1. Insert")
        print("2. Remove Highest Priority")
        print("3. Display Queue")
        print("4. Exit")
        choice = int(input("Enter your choice (1-4): "))
        if choice == 1:
```

```
    element = input("Enter the element: ")
    priority = int(input("Enter the priority (lower number = higher priority): "))
    insert(element, priority)
elif choice == 2:
    remove_highest_priority()
elif choice == 3:
    display_queue()
elif choice == 4:
    print("Exiting program.")
    break
else:
    print("Invalid choice, try again.")
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===

Priority Queue Operations:
1. Insert
2. Remove Highest Priority
3. Display Queue
4. Exit
Enter your choice (1-4): 1
Enter the element: 8
Enter the priority (lower number = higher priority): 11
Inserted: 8 with priority 11

Priority Queue Operations:
1. Insert
2. Remove Highest Priority
3. Display Queue
4. Exit
Enter your choice (1-4): 4
Exiting program.
>>>
```

DEQUE:

PROGRAM:

```
from collections import deque
# Initialize deque
dq = deque()
def insert_front(element):
    dq.appendleft(element)
    print(f"Inserted {element} at the front.")
def insert_rear(element):
    dq.append(element)
    print(f"Inserted {element} at the rear.")
def delete_front():
    if dq:
        element = dq.popleft()
        print(f"Removed {element} from the front.")
    else:
        print("Deque is empty.")
def delete_rear():
    if dq:
        element = dq.pop()
        print(f"Removed {element} from the rear.")
    else:
        print("Deque is empty.")
def display_deque():
    if dq:
        print("Current Deque:", list(dq))
    else:
        print("Deque is empty.")
```

```
# Main loop for deque operations
while True:
    print("\nDeque Operations:")
    print("1. Insert at Front")
    print("2. Insert at Rear")
    print("3. Delete from Front")
    print("4. Delete from Rear")
    print("5. Display Deque")
    print("6. Exit")
    try:
        choice = int(input("Enter your choice (1-6): "))
        if choice == 1:
            element = input("Enter the element: ")
            insert_front(element)
        elif choice == 2:
            element = input("Enter the element: ")
            insert_rear(element)
        elif choice == 3:
            delete_front()
        elif choice == 4:
            delete_rear()
        elif choice == 5:
            display_deque()
        elif choice == 6:
            print("Exiting program.")
            break
        else:
            print("Invalid choice, try again.")
    except ValueError:
```

```
print("Please enter a valid number between 1 and 6.")
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===

Deque Operations:
1. Insert at Front
2. Insert at Rear
3. Delete from Front
4. Delete from Rear
5. Display Deque
6. Exit
Enter your choice (1-6): 1
Enter the element: 8
Inserted 8 at the front.

Deque Operations:
1. Insert at Front
2. Insert at Rear
3. Delete from Front
4. Delete from Rear
5. Display Deque
6. Exit
Enter your choice (1-6): 2
Enter the element: 17
Inserted 17 at the rear.

Deque Operations:
1. Insert at Front
2. Insert at Rear
3. Delete from Front
4. Delete from Rear
5. Display Deque
6. Exit
Enter your choice (1-6): 6
Exiting program.
>>>
```

SINGLE LINK LIST:

PROGRAM:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class SinglyLinkedList:
    def __init__(self):
        self.head = None
    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new_node
        print(f"Inserted {data}.")
    def delete(self, key):
        temp = self.head
        if not temp:
            print("List is empty.")
            return
        if temp.data == key:
            self.head = temp.next
            temp = None
            print(f"Deleted {key}.")
            return
```

```
        prev = None
        while temp and temp.data != key:
            prev = temp
            temp = temp.next
        if not temp:
            print(f"{key} not found in the list.")
            return
        prev.next = temp.next
        temp = None
        print(f"Deleted {key}.")
    def display(self):
        if not self.head:
            print("List is empty.")
        else:
            temp = self.head
            print("Linked List:", end=" ")
            while temp:
                print(temp.data, end=" -> ")
                temp = temp.next
            print("None")
if __name__ == "__main__":
    sll = SinglyLinkedList()
    while True:
        print("\nSingly Linked List Operations:")
        print("1. Insert")
        print("2. Delete")
        print("3. Display List")
        print("4. Exit")
        choice = int(input("Enter your choice (1-4): "))
```

```
if choice == 1:
    data = int(input("Enter the element: "))
    sll.insert(data)
elif choice == 2:
    key = int(input("Enter the element to delete: "))
    sll.delete(key)
elif choice == 3:
    sll.display()
elif choice == 4:
    print("Exiting program.")
    break
else:
    print("Invalid choice, try again.")
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===

Singly Linked List Operations:
1. Insert
2. Delete
3. Display List
4. Exit
Enter your choice (1-4): 1
Enter the element: 8
Inserted 8.

Singly Linked List Operations:
1. Insert
2. Delete
3. Display List
4. Exit
Enter your choice (1-4): 1
Enter the element: 11
Inserted 11.

Singly Linked List Operations:
1. Insert
2. Delete
3. Display List
4. Exit
Enter your choice (1-4): 4
Exiting program.
>>
```

DOUBLE LINK LIST:

PROGRAM:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None
class DoublyLinkedList:
    def __init__(self):
        self.head = None
    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new_node
            new_node.prev = temp
        print(f"Inserted {data}.")
    def delete(self, key):
        if not self.head:
            print("List is empty.")
            return
        temp = self.head
        if temp.data == key:
            if temp.next:
                self.head = temp.next
```

```
            self.head.prev = None
        else:
            self.head = None
        temp = None
        print(f"Deleted {key}.")
        return
    while temp and temp.data != key:
        temp = temp.next
    if not temp:
        print(f"{key} not found in the list.")
        return
    if temp.next:
        temp.next.prev = temp.prev
    if temp.prev:
        temp.prev.next = temp.next
    temp = None
    print(f"Deleted {key}.")
  def display(self):
    if not self.head:
        print("List is empty.")
    else:
        temp = self.head
        print("Doubly Linked List:", end=" ")
        while temp:
            print(temp.data, end=" <-> ")
            temp = temp.next
        print("None")
if __name__ == "__main__":
  dll = DoublyLinkedList()
```

```
while True:
    print("\nDoubly Linked List Operations:")
    print("1. Insert")
    print("2. Delete")
    print("3. Display List")
    print("4. Exit")
    choice = int(input("Enter your choice (1-4): "))
    if choice == 1:
        data = int(input("Enter the element: "))
        dll.insert(data)
    elif choice == 2:
        key = int(input("Enter the element to delete: "))
        dll.delete(key)
    elif choice == 3:
        dll.display()
    elif choice == 4:
        print("Exiting program.")
        break
    else:
        print("Invalid choice, try again.")
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===

Doubly Linked List Operations:
1. Insert
2. Delete
3. Display List
4. Exit
Enter your choice (1-4): 1
Enter the element: 8
Inserted 8.

Doubly Linked List Operations:
1. Insert
2. Delete
3. Display List
4. Exit
Enter your choice (1-4): 3
Doubly Linked List: 8 <-> None

Doubly Linked List Operations:
1. Insert
2. Delete
3. Display List
4. Exit
Enter your choice (1-4): 4
Exiting program.
>>>
```

CIRCULAR LINK LSIT:

PROGRAM:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class CircularLinkedList:
    def __init__(self):
        self.last = None
    def insert(self, data):
        new_node = Node(data)
        if self.last is None:
            self.last = new_node
            self.last.next = self.last
        else:
            new_node.next = self.last.next
            self.last.next = new_node
            self.last = new_node
        print(f"Inserted {data}.")

    def delete(self, key):
        if self.last is None:
            print("List is empty.")
            return
        temp = self.last.next
        prev = self.last
        if self.last == temp and temp.data == key:
            self.last = None
            print(f"Deleted {key}.")
```

```
        return
    if temp.data == key:
        prev.next = temp.next
        self.last.next = temp.next
        print(f"Deleted {key}.")
        return
    while temp != self.last:
        if temp.data == key:
            break
        prev = temp
        temp = temp.next
    if temp.data == key:
        prev.next = temp.next
        if temp == self.last:
            self.last = prev
        print(f"Deleted {key}.")
    else:
        print(f"{key} not found in the list.")
def display(self):
    if self.last is None:
        print("List is empty.")
    else:
        temp = self.last.next
        print("Circular Linked List:", end=" ")
        while True:
            print(temp.data, end=" -> ")
            temp = temp.next
            if temp == self.last.next:
                break
```

```
        print("(back to start)")
cll = CircularLinkedList()
while True:
    print("\nCircular Linked List Operations:")
    print("1. Insert")
    print("2. Delete")
    print("3. Display List")
    print("4. Exit")
    choice = int(input("Enter your choice (1-4): "))
    if choice == 1:
        data = int(input("Enter the element: "))
        cll.insert(data)
    elif choice == 2:
        key = int(input("Enter the element to delete: "))
        cll.delete(key)
    elif choice == 3:
        cll.display()
    elif choice == 4:
        print("Exiting program.")
        break
    else:
        print("Invalid choice, try again.")
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===

Circular Linked List Operations:
1. Insert
2. Delete
3. Display List
4. Exit
Enter your choice (1-4): 1
Enter the element: 8
Inserted 8.

Circular Linked List Operations:
1. Insert
2. Delete
3. Display List
4. Exit
Enter your choice (1-4): 1
Enter the element: 17
Inserted 17.

Circular Linked List Operations:
1. Insert
2. Delete
3. Display List
4. Exit
Enter your choice (1-4): 4
Exiting program.
>>>
```

TREE TRAVERSAL:

PROGRAM:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
class BinaryTree:
    def __init__(self):
        self.root = None
    def insert(self, data):
        new_node = Node(data)
        if self.root is None:  # If tree is empty, set root to new node
            self.root = new_node
        else:
            queue = [self.root]
            while queue:
                temp = queue.pop(0)
                if not temp.left:  # Insert in the first empty left spot
                    temp.left = new_node
                    break
                else:
                    queue.append(temp.left)  # Add left node to the queue
                if not temp.right:  # Insert in the first empty right spot
                    temp.right = new_node
                    break
                else:
                    queue.append(temp.right)  # Add right node to the queue
```

```
    def inorder(self, node):
        if node:
            self.inorder(node.left)
            print(node.data, end=" ")
            self.inorder(node.right)
    def preorder(self, node):
        if node:
            print(node.data, end=" ")
            self.preorder(node.left)
            self.preorder(node.right)
    def postorder(self, node):
        if node:
            self.postorder(node.left)
            self.postorder(node.right)
            print(node.data, end=" ")
# Create an instance of BinaryTree
bt = BinaryTree()
# Main loop for interacting with the user
while True:
    print("\nBinary Tree Operations:")
    print("1. Insert Node")
    print("2. In-order Traversal")
    print("3. Pre-order Traversal")
    print("4. Post-order Traversal")
    print("5. Exit")
    # Try to handle the user's choice and inputs
    try:
        choice = int(input("Enter your choice (1-5): "))
        if choice == 1:
```

```
        data = int(input("Enter the node value: "))
        bt.insert(data)
    elif choice == 2:
        print("In-order Traversal: ", end="")
        bt.inorder(bt.root)
        print()
    elif choice == 3:
        print("Pre-order Traversal: ", end="")
        bt.preorder(bt.root)
        print()
    elif choice == 4:
        print("Post-order Traversal: ", end="")
        bt.postorder(bt.root)
        print()
    elif choice == 5:
        print("Exiting program.")
        break
    else:
        print("Invalid choice, try again.")
except ValueError:
    print("Invalid input. Please enter a number between 1 and 5.")
```

OUTPUT:

```
IDLE Shell 3.12.7
e  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===

Binary Tree Operations:
1. Insert Node
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice (1-5): 1
Enter the node value: 8

Binary Tree Operations:
1. Insert Node
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice (1-5): 1
Enter the node value: 5

Binary Tree Operations:
1. Insert Node
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice (1-5): 4
Post-order Traversal: 5 8

Binary Tree Operations:
1. Insert Node
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice (1-5): 5
Exiting program.
```

BFS:

PROGRAM:

```
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
visited = []  # List to keep track of visited nodes
queue = []  # Initialize a queue
def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)
    while queue:
        s = queue.pop(0)
        print(s, end=" ")
        for neighbour in graph[s]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)
# Start BFS from node 'A'
bfs(visited, graph, 'A')
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
A B C D E F
>>>
```

DFS:

PROGRAM:

```
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
visited = set()  # Set to keep track of visited nodes
def dfs(visited, graph, node):
    if node not in visited:
        print(node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
# Driver Code
dfs(visited, graph, 'A')
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
A
B
D
E
F
C
>>>
```

DIJKSTRA:

PROGRAM:

```
import heapq
def dijkstra(graph, start):
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    priority_queue = [(0, start)]
    while priority_queue:
        current_distance, current_node = heapq.heappop(priority_queue)
        if current_distance > distances[current_node]:
            continue
        for neighbor, weight in graph[current_node]:
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(priority_queue, (distance, neighbor))
    return distances
def build_graph():
    graph = {}
    num_nodes = int(input("Enter the number of nodes: "))
    for i in range(num_nodes):
        node = input(f"Enter the name of node {i+1}: ")
        graph[node] = []
        num_edges = int(input(f"Enter the number of edges from {node}: "))
        for _ in range(num_edges):
            neighbor = input("Enter the destination node: ")
            weight = int(input(f"Enter the weight to {neighbor}: "))
            graph[node].append((neighbor, weight))
     return graph
```

```
if __name__ == "__main__":
    graph = build_graph()
    start_node = input("Enter the starting node: ")
    distances = dijkstra(graph, start_node)
    print("\nShortest distances from node", start_node)
    for node, distance in distances.items():
        print(f"Node {node}: Distance {distance}")
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
Enter the number of nodes: 4
Enter the name of node 1: A
Enter the number of edges from A: 2
Enter the destination node: B
Enter the weight to B: 1
Enter the destination node: C
Enter the weight to C: 4
Enter the name of node 2: B
Enter the number of edges from B: 1
Enter the destination node: C
Enter the weight to C: 2
Enter the name of node 3: C
Enter the number of edges from C: 1
Enter the destination node: D
Enter the weight to D: 1
Enter the name of node 4: D
Enter the number of edges from D: 0
Enter the starting node: A

Shortest distances from node A
Node A: Distance 0
Node B: Distance 1
Node C: Distance 3
Node D: Distance 4
>>>
```

HASH TABLE:

PROGRAM:

```
class HashTable:
    def __init__(self, size):
        self.size = size
        self.table = [[] for _ in range(size)]
    def hash_function(self, key):
        return hash(key) % self.size
    def insert(self, key, value):
        index = self.hash_function(key)
        for pair in self.table[index]:
            if pair[0] == key:
                pair[1] = value
                return
        self.table[index].append([key, value])
    def search(self, key):
        index = self.hash_function(key)
        for pair in self.table[index]:
            if pair[0] == key:
                return pair[1]
        return None
    def delete(self, key):
        index = self.hash_function(key)
        for i, pair in enumerate(self.table[index]):
            if pair[0] == key:
                del self.table[index][i]
                return True
        return False
if __name__ == "__main__":
```

```
size = int(input("Enter the size of the hash table: "))
ht = HashTable(size)
while True:
    operation = input("Choose operation: insert, search, delete, or exit: ").lower()
    if operation == "insert":
        key = input("Enter key: ")
        value = input("Enter value: ")
        ht.insert(key, value)
        print(f"Inserted ({key}, {value})")
    elif operation == "search":
        key = input("Enter key to search: ")
        result = ht.search(key)
        if result:
            print(f"Found: {result}")
        else:
            print("Key not found")
    elif operation == "delete":
        key = input("Enter key to delete: ")
        if ht.delete(key):
            print(f"Deleted key: {key}")
        else:
            print("Key not found")
    elif operation == "exit":
        break
    else:
        print("Invalid operation. Please choose insert, search, delete, or exit.")
```

OUTPUT:

```
IDLE Shell 3.12.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:/Users/Asus/AppData/Local/Programs/Python/Python312/dsalab.py ===
Enter the size of the hash table: 4
Choose operation: insert, search, delete, or exit: INSERT
Enter key: DSA
Enter value: DSA LAB
Inserted (DSA, DSA LAB)
Choose operation: insert, search, delete, or exit: SEARCH
Enter key to search: DSA
Found: DSA LAB
Choose operation: insert, search, delete, or exit: EXIT
>>>
```