DATA STRUCTURE AND ALGORITHMS LAB

(Course Code: 22UPCSC1C10)

A programming laboratory record submitted to Periyar University, Salem

In partial fulfillment of the requirements for the degree of

MASTER OF COMPUTER APPLICATIONS

By

ELANCHEZHAIN M.

[Reg. No.: U22PG507CAP006]

DEPARTMENT OF COMPUTER SCIENCE

PERIYAR UNIVERSITY

(NAAC `A++` Grade with CGPA 3.61) – NIRF RANK 59 – ARIIA RANK 10

PERIYAR PALKALAI NAGAR,

SALEM – 636 011.

(APRIL - 2023)

CERTIFICATE

This is to certify that the Programming Laboratory entitled "**DATA STRUCTURE AND ALGORITHMS LAB (22UPCSC1C10)**" is a bonafide record work done by Mr. /Ms. ____________________________________ Register No: ___________________________ as partial fulfillment of the requirements for the degree of Master of Computer Applications, in the Department of Computer Science, Periyar University, Salem, during the Academic Year 2023-2024.

Staff In-charge　　　　　　　　　　　　Head of the Department

Submitted for the practical examination held on……...………………

Internal Examiner　　　　　　　　　　　　External Examiner

CONTENTS

SOURCE CODE:

i) Linear recursion:

```
prime_numbers = []
First_number=int(input("Enter the starting number : "))
Last_number=int(input("Enter the last number : "))
for i in (range(First_number, Last_number)):
    for j in range(2,i-1):
        if i%j==0:
            break
    else:
        prime_numbers.append(i)
print(prime_numbers)

def my_prime_search_index(prime_numbers, current_prime):
    myprime = current_prime
    for i in list(range(len(prime_numbers))):
        if prime_numbers[i] == myprime:
            print("Found at : ", i)
            break
    else :
        print("It is not a prime number")

x=int(input("Enter the prime number to find index : "))
my_prime_search_index( prime_numbers, x )
```

OUTPUT (Linear recursion**):**

```
Enter the starting number : 5
Enter the last number : 29
[5, 7, 11, 13, 17, 19, 23]
Enter the prime number to find index : 11
Found at :  2
```

SOURCE CODE:

ii) Binary recursion:

```
def binary_search(arr, low, high, key):
    if low > high:
        return -1
    mid = (low + high) // 2
    if arr[mid] == key:
        return mid
    elif arr[mid] < key:
        return binary_search(arr, mid+1, high, key)
    else:
        return binary_search(arr, low, mid-1, key)

arr = list(map(int, input("Enter a sorted list of numbers: ").split()))
key = int(input("Enter the number to be searched: "))
result = binary_search(arr, 0, len(arr)-1, key)
if result == -1:
    print(key, "not found in the list.")
else:
    print(key, "found at index", result)
```

OUTPUT (Binary recursion**):**

```
Enter a sorted list of numbers: 8 1 3 4 7 6
Enter the number to be searched: 3
3 found at index 2
```

SOURCE CODE:

```
queue=[]
while True:
    print("Enter your choice:\n","1.Enqueue\n","2.Dequeue\n",
    "3.queue is   empty or not \n","4.Display queue\n",
    "5.Exit\n")
    choice=int(input())
    if choice==1:
        item=input("enter the item to be enqueued:")
        queue.append(item)
    elif choice==2:
        if not queue:
            print("queue is empty")
        else:
            item=queue.pop(0)
            print("dequeue item:",item)
    elif choice==3:
        if not queue:
            print("queue is empty")
        else:
            print("queue is not empty")
    elif choice==4:
        if not queue:
            print("queue is not empty")
        else:
            print("queue=",queue)
```

```
elif choice==5:
    break
else:
    print("Invalid choice")
```

OUTPUT:

```
Enter your choice:
 1.Enqueue
 2.Dequeue
 3.queue is empty or not
 4.Display queue
 5.Exit

1
enter the item to be enqueued:A
Enter your choice:
 1.Enqueue
 2.Dequeue
 3.queue is empty or not
 4.Display queue
 5.Exit

1
enter the item to be enqueued:B
Enter your choice:
 1.Enqueue
 2.Dequeue
 3.queue is empty or not
 4.Display queue
 5.Exit

1
enter the item to be enqueued:C
Enter your choice:
 1.Enqueue
 2.Dequeue
 3.queue is empty or not
 4.Display queue
 5.Exit

3
queue is not empty
Enter your choice:
 1.Enqueue
 2.Dequeue
 3.queue is empty or not
 4.Display queue
 5.Exit
```

```
2
dequeue item: A
Enter your choice:
 1.Enqueue
 2.Dequeue
 3.queue is empty or not
 4.Display queue
 5.Exit

4
queue= ['B', 'C']
Enter your choice:
 1.Enqueue
 2.Dequeue
 3.queue is empty or not
 4.Display queue
 5.Exit

5
```

SOURCE CODE:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None

class DoublyLinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            curr_node = self.head
            while curr_node.next:
                curr_node = curr_node.next
            curr_node.next = new_node
            new_node.prev = curr_node

    def prepend(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
```

```
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node

    def delete(self, key):
        if self.head is None:
            return

        if self.head.data == key:
            self.head = self.head.next
            if self.head:
                self.head.prev = None
        else:
            curr_node = self.head
            while curr_node.next:
                if curr_node.next.data == key:
                    curr_node.next = curr_node.next.next
                    if curr_node.next:
                        curr_node.next.prev = curr_node
                    break
                curr_node = curr_node.next

    def print_list(self):
        curr_node = self.head
        while curr_node:
            print(curr_node.data)
```

```python
        curr_node = curr_node.next

doubly_linked_list = DoublyLinkedList()

while True:
    print("1. Append to list")
    print("2. Prepend to list")
    print("3. Delete from list")
    print("4. Print list")
    print("5. Exit")
    user_choice = input("Enter your choice: ")
    if user_choice == "1":
        data = input("Enter the element to append: ")
        doubly_linked_list.append(data)
    elif user_choice == "2":
        data = input("Enter the element to prepend: ")
        doubly_linked_list.prepend(data)
    elif user_choice == "3":
        data = input("Enter the element to delete: ")
        doubly_linked_list.delete(data)
    elif user_choice == "4":
        doubly_linked_list.print_list()
    elif user_choice == "5":
        break
    else:
        print("Invalid choice. Please try again.")
```

OUTPUT:

```
1. Append to list
2. Prepend to list
3. Delete from list
4. Print list
5. Exit
Enter your choice: 1
Enter the element to append: Dhanush
1. Append to list
2. Prepend to list
3. Delete from list
4. Print list
5. Exit
Enter your choice: 1
Enter the element to append: Vijay
1. Append to list
2. Prepend to list
3. Delete from list
4. Print list
5. Exit
Enter your choice: 2
Enter the element to prepend: Ajith
1. Append to list
2. Prepend to list
3. Delete from list
4. Print list
5. Exit
Enter your choice: 4
Ajith
Dhanush
Vijay
1. Append to list
2. Prepend to list
3. Delete from list
4. Print list
5. Exit
Enter your choice: 3
Enter the element to delete: Dhanush
1. Append to list
2. Prepend to list
3. Delete from list
4. Print list
5. Exit
```

```
Enter your choice: 4
Ajith
Vijay
1. Append to list
2. Prepend to list
3. Delete from list
4. Print list
5. Exit
Enter your choice: 5
```

SOURCE CODE:

```
class Stack:
    def __init__(self):
        self.stack = []

    def push(self, element):
        self.stack.append(element)

    def pop(self):
        if not self.is_empty():
            last_element = self.stack[-1]
            del self.stack[-1]
            return last_element
        else:
            return "Stack Already Empty"

    def is_empty(self):
        return self.stack == []

    def print_stack(self):
        print(self.stack)
if __name__ == "__main__":
    s = Stack()
    while True:
        var = int(input("1. PUSH\n2. POP\n3. IS EMPTY\n4. PRINT
        STACK\n5. EXIT\n"))
```

```
if var == 1:
    item = input("Enter element to push in stack: ")
    s.push(item)
elif var == 2:
    print(s.pop())
elif var == 3:
    print(s.is_empty())
elif var == 4:
    s.print_stack()
elif var == 5:
    break
```

OUTPUT:

```
1. PUSH
2. POP
3. IS EMPTY
4. PRINT STACK
5. EXIT
1
Enter element to push in stack: Deva
1. PUSH
2. POP
3. IS EMPTY
4. PRINT STACK
5. EXIT
1
Enter element to push in stack: Meera
1. PUSH
2. POP
3. IS EMPTY
4. PRINT STACK
5. EXIT
1
Enter element to push in stack: Dhas
1. PUSH
2. POP
3. IS EMPTY
4. PRINT STACK
5. EXIT
1
Enter element to push in stack: Sree
1. PUSH
2. POP
3. IS EMPTY
4. PRINT STACK
5. EXIT
4
['Deva', 'Meera', 'Dhas', 'Sree']
1. PUSH
2. POP
3. IS EMPTY
4. PRINT STACK
5. EXIT
3
False
```

```
1. PUSH
2. POP
3. IS EMPTY
4. PRINT STACK
5. EXIT
2
Sree
1. PUSH
2. POP
3. IS EMPTY
4. PRINT STACK
5. EXIT
4
['Deva', 'Meera', 'Dhas']
1. PUSH
2. POP
3. IS EMPTY
4. PRINT STACK
5. EXIT
5
```

SOURCE CODE:

```
import heapq


class MaxHeap:
    def __init__(self):
        self._heap = []


    def insert(self, item):
        heapq.heappush(self._heap, -item)


    def delete(self):
        if len(self._heap) > 0:
            return -heapq.heappop(self._heap)
        else:
            raise Exception("Heap is empty")


    def peek(self):
        if len(self._heap) > 0:
            return -self._heap[0]
        else:
            raise Exception("Heap is empty")


    def size(self):
        return len(self._heap)
```

```python
heap = MaxHeap()

input_list = input("Enter integers to insert into the heap, separated by spaces: ").split()
for item in input_list:
    heap.insert(int(item))

print("The highest priority element in the heap is:", heap.peek())
print("The deleted element from the heap is:", heap.delete())
print("The new highest priority element in the heap is:", heap.peek())
```

OUTPUT:

```
Enter integers to insert into the heap, separated by spaces: 5 3 4 7 8 2
The highest priority element in the heap is: 8
The deleted element from the heap is: 8
The new highest priority element in the heap is: 7
```

SOURCE CODE:

```
def merge_sort(arr):
    if len(arr)<=1:
        return arr
    mid=len(arr)//2
    left=arr[:mid]
    right=arr[mid:]

    left=merge_sort(left)
    right=merge_sort(right)
    return merge(left,right)

def  merge(left,right):
    result=[]
    i,j=0,0
    while i<len(left) and j<len(right):
        if left[i]<right[j]:
            result.append(left[i])
            i+=1
        else:
            result.append(right[j])
            j+=1

    result+=left[i:]
    result+=right[j:]
    return result
```

```
arr=input("Enter the list of numbers separated by space:").split()
arr=[int(num) for num in arr]
print("Input Array:",arr)
sorted_arr=merge_sort(arr)
print("Sorted Array:",sorted_arr)
```

OUTPUT:

```
Enter the list of numbers separated by space:23 45 12 56 34 1
Input Array: [23, 45, 12, 56, 34, 1]
Sorted Array: [1, 12, 23, 34, 45, 56]
```

SOURCE CODE:

```
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)

arr=[]
n=int(input("Enter number of elements: "))
for i in range(n):
    element=int(input("Enter element {}: ".format(i+1)))
    arr.append(element)

print("Unsorted array: ",arr)
sorted_arr=quick_sort(arr)
print("Sorted array: ",sorted_arr)
```

OUTPUT:

```
Enter number of elements: 5
Enter element 1: 23
Enter element 2: 76
Enter element 3: 48
Enter element 4: 56
Enter element 5: 1
Unsorted array:  [23, 76, 48, 56, 1]
Sorted array:  [1, 23, 48, 56, 76]
```

SOURCE CODE:

```
class DisjointSet:
    def __init__(self, size):
        self.parent = list(range(size))
        self.rank = [0] * size

    def find(self, i):
        if self.parent[i] != i:
            self.parent[i] = self.find(self.parent[i])
        return self.parent[i]

    def union(self, i, j):
        root_i = self.find(i)
        root_j = self.find(j)
        if root_i != root_j:
            if self.rank[root_i] < self.rank[root_j]:
                self.parent[root_i] = root_j
            elif self.rank[root_i] > self.rank[root_j]:
                self.parent[root_j] = root_i
            else:
                self.parent[root_j] = root_i
                self.rank[root_i] += 1

class KruskalMST:
    def __init__(self, edges, num_vertices):
        self.edges = edges
```

```
        self.num_vertices = num_vertices

    def find_mst(self):
        result = []
        disjoint_set = DisjointSet(self.num_vertices)
        for u, v, weight in sorted(self.edges, key=lambda e: e[2]):
            if disjoint_set.find(u) != disjoint_set.find(v):
                result.append((u, v, weight))
                disjoint_set.union(u, v)
        return result

if __name__ == '__main__':
    edges = [(0, 1, 8), (0, 2, 5), (1, 2, 9), (1, 3, 11), (2, 3, 15), (2, 4, 10),
    (3, 4, 7)]
    kruskal = KruskalMST(edges, num_vertices=5)
    mst = kruskal.find_mst()
    for u, v, weight in mst:
        print(f"Edge: {u}-{v} weight: {weight}")
```

OUTPUT:

```
Edge: 0-2 weight: 5
Edge: 3-4 weight: 7
Edge: 0-1 weight: 8
Edge: 2-4 weight: 10
```

SOURCE CODE:

```
class Node:
    def __init__(self,value):
        self.value=value
        self.left=None
        self.right=None

def depth_first_search(root):
    if root is None:
        return
    print(root.value)
    depth_first_search(root.left)
    depth_first_search(root.right)

root=Node(50)
root.left=Node(10)
root.right=Node(25)
root.left.left=Node(18)
root.left.right=Node(36)
root.right.left=Node(22)
root.right.right=Node(46)

depth_first_search(root)
```

OUTPUT:

```
50
10
18
36
25
22
46
```

SOURCE CODE:

```
class Node:
    def __init__(self, val=None):
        self.value = val
        self.left_child = None
        self.right_child = None

class BST:
    def __init__(self):
        self.root = None

    def insert(self, val):
        if not self.root:
            self.root = Node(val)
        else:
            self._insert(val, self.root)

    def _insert(self, val, current_node):
        if val < current_node.value:
            if not current_node.left_child:
                current_node.left_child = Node(val)
            else:
                self._insert(val, current_node.left_child)
        elif val > current_node.value:
            if not current_node.right_child:
                current_node.right_child = Node(val)
```

```
            else:
                self._insert(val, current_node.right_child)
        else:
            print("Value already exists in the tree.")

    def find(self, val):
        if self.root:
            res = self._find(val, self.root)
            if res:
                return True
            else:
                return False
        else:
            return False

    def _find(self, val, current_node):
        if val == current_node.value:
            return True
        elif val < current_node.value and current_node.left_child:
            return self._find(val, current_node.left_child)
        elif val > current_node.value and current_node.right_child:
            return self._find(val, current_node.right_child)
        return False
```

```
    def inorder_traversal(self, node):
        if node is not None:
            self.inorder_traversal(node.left_child)
            print(node.value, end=" ")
            self.inorder_traversal(node.right_child)

    def preorder_traversal(self, node):
        if node is not None:
            print(node.value, end=" ")
            self.preorder_traversal(node.left_child)
            self.preorder_traversal(node.right_child)

    def postorder_traversal(self, node):
        if node is not None:
            self.postorder_traversal(node.left_child)
            self.postorder_traversal(node.right_child)
            print(node.value, end=" ")

if __name__ == "__main__":
    bst = BST()
    while True:
        print("\nBinary Search Tree Operations:")
        print("1. Insert a node")
        print("2. Search for a node")
        print("3. Traverse the tree (inorder)")
        print("4. Traverse the tree (preorder)")
```

```
print("5. Traverse the tree (postorder)")
print("6. Exit")

choice = int(input("\nEnter your choice: "))
if choice == 1:
    val = int(input("Enter value to be inserted: "))
    bst.insert(val)
elif choice == 2:
    val = int(input("Enter value to be searched: "))
    if bst.find(val):
        print("Value found in the tree.")
    else:
        print("Value not found in the tree.")
elif choice == 3:
    print("\nInorder Traversal: ", end="")
    bst.inorder_traversal(bst.root)
elif choice == 4:
    print("\nPreorder Traversal: ", end="")
    bst.preorder_traversal(bst.root)
elif choice == 5:
    print("\nPostorder Traversal: ", end="")
    bst.postorder_traversal(bst.root)
elif choice == 6:
    break
else:
    print("Invalid choice. Try again.")
```

OUTPUT:

```
Binary Search Tree Operations:
1. Insert a node
2. Search for a node
3. Traverse the tree (inorder)
4. Traverse the tree (preorder)
5. Traverse the tree (postorder)
6. Exit

Enter your choice: 1
Enter value to be inserted: 50

Binary Search Tree Operations:
1. Insert a node
2. Search for a node
3. Traverse the tree (inorder)
4. Traverse the tree (preorder)
5. Traverse the tree (postorder)
6. Exit

Enter your choice: 1
Enter value to be inserted: 30

Binary Search Tree Operations:
1. Insert a node
2. Search for a node
3. Traverse the tree (inorder)
4. Traverse the tree (preorder)
5. Traverse the tree (postorder)
6. Exit

Enter your choice: 1
Enter value to be inserted: 70

Binary Search Tree Operations:
1. Insert a node
2. Search for a node
3. Traverse the tree (inorder)
4. Traverse the tree (preorder)
5. Traverse the tree (postorder)
6. Exit

Enter your choice: 2
Enter value to be searched: 30
Value found in the tree.
```

```
Binary Search Tree Operations:
1. Insert a node
2. Search for a node
3. Traverse the tree (inorder)
4. Traverse the tree (preorder)
5. Traverse the tree (postorder)
6. Exit

Enter your choice: 3

Inorder Traversal: 30 50 70
Binary Search Tree Operations:
1. Insert a node
2. Search for a node
3. Traverse the tree (inorder)
4. Traverse the tree (preorder)
5. Traverse the tree (postorder)
6. Exit

Enter your choice: 4

Preorder Traversal: 50 30 70
Binary Search Tree Operations:
1. Insert a node
2. Search for a node
3. Traverse the tree (inorder)
4. Traverse the tree (preorder)
5. Traverse the tree (postorder)
6. Exit

Enter your choice: 5

Postorder Traversal: 30 70 50
Binary Search Tree Operations:
1. Insert a node
2. Search for a node
3. Traverse the tree (inorder)
4. Traverse the tree (preorder)
5. Traverse the tree (postorder)
6. Exit

Enter your choice: 6
```