Advanced Data Structure and Algorithm Laboratory

(CS-514)

Laboratory Sessional Report

Submitted by
Aadarsh Gupta
(23203001)

for **Master of Technology** in

Department of Computer Science and Engineering

Submitted to
Dr. Urvashi Bansal
(Assistant Professor)

Department of Computer Science and Engineering

DR. B.R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY

JALANDHAR, PUNJAB-144001

Index

Page No.	Lab Task	Date	Faculty Sign
	Week 1		
5	1. Print all primes numbers from 1 to n where n will be entered by the user. 2. Print the location of elements in an array using linear search. 3. Print the location of elements in an array using Binary search. 4. Print the elements from a list of numbers whose sum is equal to a number given as input.	18/01/24	
	Week 2		
10	1. Insertion, Deletion, searching (Linear, binary), sorting (Bubble sort) from 1 D array. 2. Matrix operation (addition, subtraction, multiplication and transpose) from 2 D array.	24/01/24	
	Week 3		
15	1. Sort the elements using Insertion sort, selection sort, merge sort, and quick sort	31/02/24	
	Week 4		
20	1. Implement following algorithms wrt linked list: ◦ Insertion ◦ Deletion ◦ Searching ◦ Traversal	07/02/24	

Notes:

Index

Page No.	Lab Task	Date	Faculty Sign
	Week 5		
24	1. Implement following programs for doubly linked list: ◦ Insertion ◦ Deletion ◦ Traversing 2. Implement push and pop operations on stack. 3. Write a program to evaluate a postfix expression.	14/02/24	
	Week 6		
33	1. Write the program to print all possible paths between two vertices. 2. Write the program to print all possible paths between two vertices.	21/02/24	
	Week 7		
38	1. Write the program to check the presence of articulation points in the graph. 2. Write the program to construct a binary tree.	03/04/24	
	Week 8		
43	1. Implement following algorithms: ◦ BST insertion ◦ BST Deletion ◦ BST Traversal	20/04/24	

Notes:

Index

Page No.	Lab Task	Date	Faculty Sign
	Week 9		
46	1. Implement AVL tree insertion and deletion.	24/04/24	
	Week 10		
51	1. Implement following programs: ◦ String matching with a pattern using brute force approach. ◦ Max heap insertion, Deletion, and sorting. Print the numbers present in the heap in sorted order.	01/05/24	

Notes:

Practical 01:

Question 1: Print all primes numbers from 1 to n where n will be entered by the user.

Code:

```
def is_prime(num):
    if num < 2:
        return False
    for i in range(2, num):
        if num % i == 0:
            return False
    return True

# Get user input for the value of n
try:
    n = int(input("Enter the value of n: "))
except ValueError:
    print("Invalid input. Please enter an integer.")
    exit()

# Print prime numbers from 1 to n
print("Prime numbers from 1 to", n, "are:")
for number in range(2, n + 1):
    if is_prime(number):
        print(number)
```

Output:

```
aadarsh@Aadarshs-Maccy Lab % python3 program1.py
Enter the value of n: 7
Prime numbers from 1 to 7 are:
2
3
5
7
aadarsh@Aadarshs-Maccy Lab %
```

Question 2: **Print the location of elements in an array using linear search.**

Code:

```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1

# User input for the array
arr = list(map(int, input("Enter the array elements (space-separated): ").split()))

# User input for the element to search
target = int(input("Enter the element to search: "))

# Perform linear search
index = linear_search(arr, target)

# Display the result
if index != -1:
    print(f"The element {target} is found at index {index}.")
else:
    print(f"The element {target} is not present in the array.")
```

Output:

```
aadarsh@Aadarshs-Maccy Lab % python3 program2.py
Enter the array elements (space-separated): 2 56 21 23 44 56 78
Enter the element to search: 44
The element 44 is found at index 4.
aadarsh@Aadarshs-Maccy Lab %
```

Question 3: **Print the location of elements in an array using Binary search.**

Code:

```
def binary_search(arr, target):
    low, high = 0, len(arr) - 1

    while low <= high:
        mid = (low + high) // 2

        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1

    return -1  # Element not found

# User input for the array
arr = list(map(int, input("Enter the array elements (space-separated): ").split()))

# User input for the element to search
target = int(input("Enter the element to search: "))

# Perform Binary search
index = binary_search(arr, target)

# Display the result
if index != -1:
    print(f"The element {target} is found at index {index}.")
else:
    print(f"The element {target} is not present in the array.")
```

Output :

```
aadarsh@Aadarshs-Maccy Lab % python3 program3.py
Enter the array elements (space-separated): 5 7 32 123 321 323 453 555 678
Enter the element to search: 123
The element 123 is found at index 3.
aadarsh@Aadarshs-Maccy Lab %
```

Question 4: Print the elements from a list of numbers whose sum is equal to a number given as input.

Code:

```
def find_subsets_with_sum(numbers, target_sum):
    subsets = []
    def backtrack(start, target, current_subset):
        if target == 0:
            subsets.append(tuple(current_subset))
            return
        for i in range(start, len(numbers)):
            if target - numbers[i] >= 0:
                backtrack(i + 1, target - numbers[i], current_subset + [numbers[i]])

    backtrack(0, target_sum, [])
    return subsets

# Example list of numbers
numbers = list(map(int, input("Enter the array elements (space-separated): ").split()))

# Get user input for the target sum
target_sum = int(input("Enter the target sum: "))


# Find and print subsets with the target sum
result_subsets = find_subsets_with_sum(numbers, target_sum)

if result_subsets:
    print(f"Subsets with sum equal to {target_sum}:")
    for subset in result_subsets:
        print(subset)
```

```
else:
    print(f"No subsets found with sum equal to {target_sum}.")
```

Output:

```
aadarsh@Aadarshs-Maccy Lab % python3 program4.py
Enter the array elements (space-separated): 1 2 3 4 5 1
Enter the target sum: 5
Subsets with sum equal to 5:
(1, 3, 1)
(1, 4)
(2, 3)
(4, 1)
(5,)
aadarsh@Aadarshs-Maccy Lab %
```

Practical 02:

Question 1: Insertion, Deletion, searching (Linear, binary), sorting (Bubble sort) from 1 D array.

Code:

```
def insert_element(arr, element, position):
    new_array = []
    for i in range(len(arr) + 1):
        if i < position:
            new_array.append(arr[i])
        elif i == position:
            new_array.append(element)
        else:
            new_array.append(arr[i - 1])
    return new_array

def delete_element(arr, element):
    new_array = []
    element_found = False
    for value in arr:
        if value == element and not element_found:
            element_found = True
        else:
            new_array.append(value)
    if not element_found:
        print(f"{element} not found in the array.")
    return new_array


def linear_search(arr, key):
    for index, value in enumerate(arr):
        if value == key:
            return index
    return -1

def binary_search(arr, key):
    low, high = 0, len(arr) - 1

    while low <= high:
```

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

    return -1

def bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr

# Example usage:
my_array = [5, 2, 9, 1, 3, 6, 12, 43, 124, 54]

# Insertion
my_array = insert_element(my_array, 8, 2)
print("Array after insertion:", my_array)

# Deletion
my_array = delete_element(my_array, 9)
print("Array after deletion:", my_array)

# Linear Search
search_key = 12
result = linear_search(my_array, search_key)
print(f"Linear search: {search_key} found at index {result}" if result != -1 else
f"{search_key} not found")

# Binary Search (Note: Array must be sorted)
sorted_array = bubble_sort(my_array.copy())
search_key = 1
result = binary_search(sorted_array, search_key)
```

```
print(f"Binary search: {search_key} found at index {result}" if result != -1 else
f"{search_key} not found")

# Bubble Sort
sorted_array = bubble_sort(my_array.copy())
print("Array after bubble sort:", sorted_array)
```

Output:

```
aadarsh@Aadarshs-Maccy Lab_2 % python3 1d.py
Array after insertion: [5, 2, 8, 9, 1, 3, 6, 12, 43, 124, 54]
Array after deletion: [5, 2, 8, 1, 3, 6, 12, 43, 124, 54]
Linear search: 12 found at index 6
Binary search: 1 found at index 0
Array after bubble sort: [1, 2, 3, 5, 6, 8, 12, 43, 54, 124]
aadarsh@Aadarshs-Maccy Lab_2 %
```

Question 2: Matrix operation (addition, subtraction, multiplication and transpose) from 2 D array.

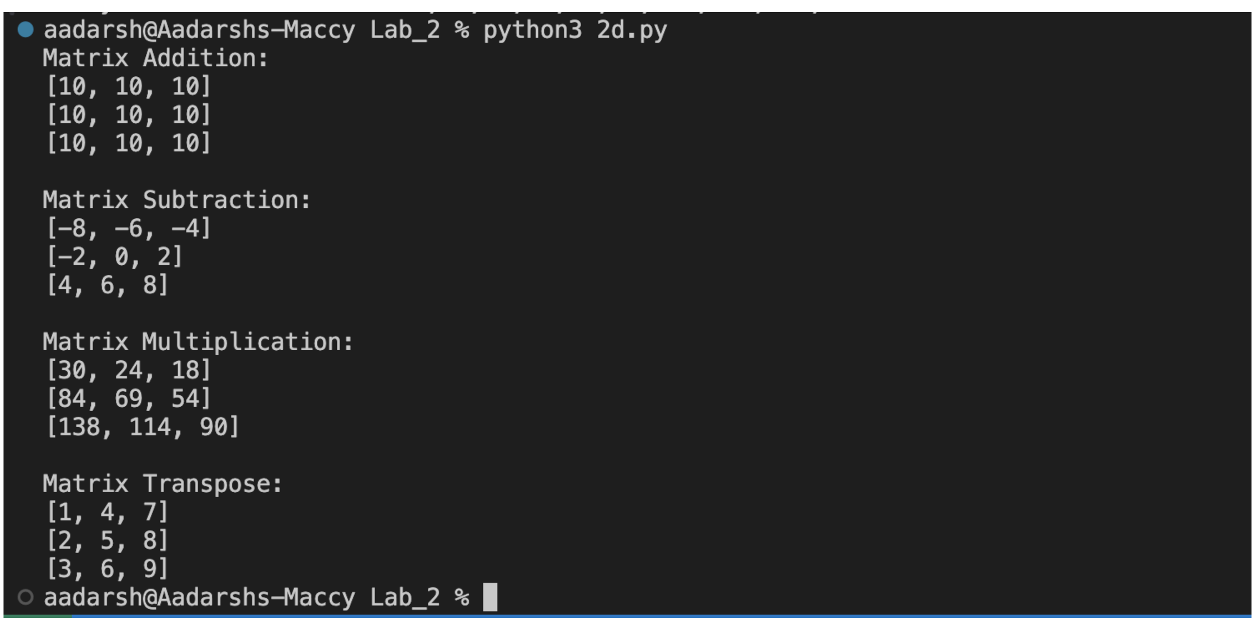
Code:

```
def add_matrices(matrix1, matrix2):
    result = []
    for i in range(len(matrix1)):
        row = []
        for j in range(len(matrix1[0])):
            row.append(matrix1[i][j] + matrix2[i][j])
        result.append(row)
    return result

def subtract_matrices(matrix1, matrix2):
    result = []
    for i in range(len(matrix1)):
        row = []
        for j in range(len(matrix1[0])):
            row.append(matrix1[i][j] - matrix2[i][j])
        result.append(row)
    return result
```

```
def multiply_matrices(matrix1, matrix2):
    result = []
    for i in range(len(matrix1)):
        row = []
        for j in range(len(matrix2[0])):
            element = 0
            for k in range(len(matrix2)):
                element += matrix1[i][k] * matrix2[k][j]
            row.append(element)
        result.append(row)
    return result

def transpose_matrix(matrix):
    result = []
    for j in range(len(matrix[0])):
        row = []
        for i in range(len(matrix)):
            row.append(matrix[i][j])
        result.append(row)
    return result

# Example matrices
matrix_a = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

matrix_b = [
    [9, 8, 7],
    [6, 5, 4],
    [3, 2, 1]
]

# Matrix addition


result_addition = add_matrices(matrix_a, matrix_b)
print("Matrix Addition:")
for row in result_addition:
```

```
    print(row)

# Matrix subtraction
result_subtraction = subtract_matrices(matrix_a, matrix_b)
print("\nMatrix Subtraction:")
for row in result_subtraction:
    print(row)

# Matrix multiplication
result_multiplication = multiply_matrices(matrix_a, matrix_b)
print("\nMatrix Multiplication:")
for row in result_multiplication:
    print(row)

# Matrix transpose
result_transpose = transpose_matrix(matrix_a)
print("\nMatrix Transpose:")
for row in result_transpose:
    print(row)
```

Output:



Practical 03:

Sort the elements using Insertion sort, selection sort, merge sort, and quick sort

Insertion Sort:

```
def insertionSort(arr):

    # Traverse through 1 to len(arr)
    for i in range(1, len(arr)):

        key = arr[i]
        # Move elements of arr[0..i-1], that are
        # greater than key, to one position ahead
        # of their current position
        j = i-1
        while j >= 0 and key < arr[j] :
                arr[j + 1] = arr[j]
                j -= 1
        arr[j + 1] = key


# Driver code to test above
arr = [12, 11, 13, 5, 6]
insertionSort(arr)
for i in range(len(arr)):
    print ("% d" % arr[i])
```

Output:

```
aadarsh@Aadarshs-Maccy Lab_3 % python3 insertion.py
 5
 6
 11
 12
 13
aadarsh@Aadarshs-Maccy Lab_3 %
```

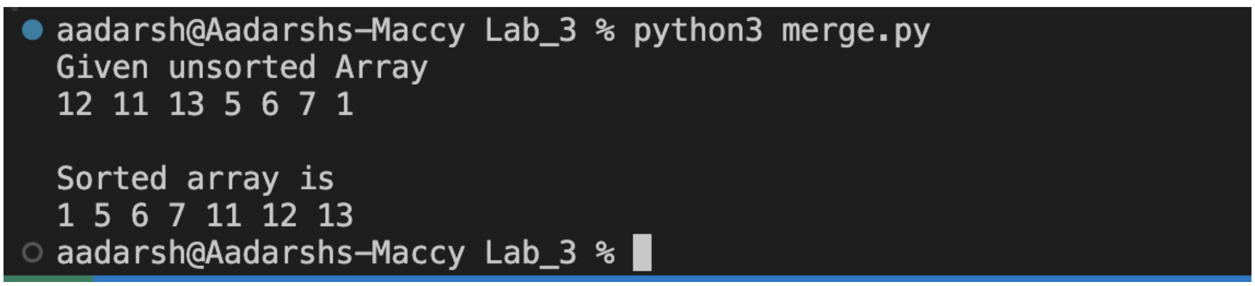
Merge Sort:

```
def mergeSort(arr):
```

```
    if len(arr) > 1:
        # Finding the mid of the array
        mid = len(arr)//2
        # Dividing the array elements
        L = arr[:mid]
        # Into 2 halves
        R = arr[mid:]
        # Sorting the first half
        mergeSort(L)
        # Sorting the second half
        mergeSort(R)
        i = j = k = 0
        # Copy data to temp arrays L[] and R[]
        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
        # Checking if any element was left
        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 printList(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()
```

```
# Driver Code

if __name__ == '__main__':
    arr = [12, 11, 13, 5, 6, 7, 1]
    print("Given unsorted Array")
    printList(arr)
    mergeSort(arr)
    print("\nSorted array is ")
    printList(arr)
```

Output:



Quick Sort:

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr

    # Taking first element as the pivot
    pivot = arr[0]
    less_than_pivot = []
    equal_to_pivot = []
    greater_than_pivot = []

    for num in arr:
        if num < pivot:
            less_than_pivot.append(num)
        elif num == pivot:
            equal_to_pivot.append(num)
        else:
            greater_than_pivot.append(num)

    return quick_sort(less_than_pivot) + equal_to_pivot + quick_sort(greater_than_pivot)

# Driver code to run the quick_sort
unsorted_array = [3, 6, 8, 10, 1, 2, 1]
sorted_array = quick_sort(unsorted_array)

print("Unsorted Array:", unsorted_array)
print("Sorted Array:", sorted_array)
```

Output:

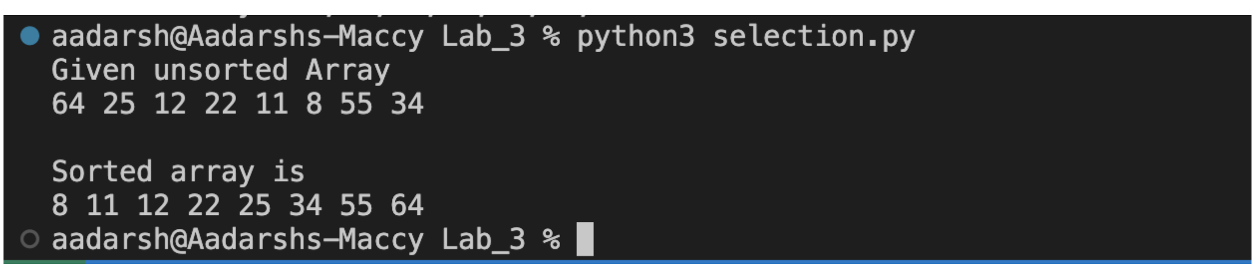
```
aadarsh@Aadarshs-Maccy Lab_3 % python3 quick.py
Unsorted Array: [3, 6, 8, 10, 1, 2, 1]
Sorted Array: [1, 1, 2, 3, 6, 8, 10]
aadarsh@Aadarshs-Maccy Lab_3 %
```

Selection Sort:

```
def selectionSort(A):
    for i in range(len(A)):
```

```
        # Find the minimum element in remaining
        # unsorted array
        min_idx = i
        for j in range(i+1, len(A)):

            if A[min_idx] > A[j]:

                min_idx = j
        # Swap the found minimum element with

        # the first element
        A[i], A[min_idx] = A[min_idx], A[i]


def printList(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()

# Driver Code

if __name__ == '__main__':
    A = [64, 25, 12, 22, 11, 8, 55, 34]
    print("Given unsorted Array")
    printList(A)
    selectionSort(A)
    print("\nSorted array is ")
    printList(A)
```

Output:



```
aadarsh@Aadarshs-Maccy Lab_3 % python3 selection.py
Given unsorted Array
64 25 12 22 11 8 55 34

Sorted array is
8 11 12 22 25 34 55 64
aadarsh@Aadarshs-Maccy Lab_3 %
```

Practical : 04

Implement following algorithms wrt linked list:

1. Insertion
2. Deletion
3. Searching
4. Traversal

```
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 not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
    def search(self, data):
        pos = 0
        current = self.head
        while current:
            if current.data == data:
                return pos
            current = current.next
            pos += 1
        return -1

    def delete(self, data):
        if not self.head:
            return
        if self.head.data == data:
            self.head = self.head.next
```

```
            return
        current = self.head
        while current.next:
            if current.next.data == data:
                current.next = current.next.next
                return
            current = current.next

    def display(self):
        elements = []
        current = self.head
        while current:
            elements.append(current.data)
            current = current.next
        return elements


if __name__ == "__main__":
    linked_list = LinkedList()
    while True:
        print("1. Insert")
        print("2. Delete")
        print("3. Display")
        print("4. Search Element")
        print("5. Quit")
        option = int(input("Enter your option: "))
        if option == 1:
            data = int(input("Enter the data to insert: "))
            linked_list.insert(data)
        elif option == 2:
            data = int(input("Enter the data to delete: "))
            linked_list.delete(data)
        elif option == 3:
            print("The linked list is: ", linked_list.display())
        elif option == 4:
            data = int(input("Enter the data to search for: "))
            result = linked_list.search(data)
            if result != -1:
                print("Element found at position: ", result)
            else:
```

```
            print("Element not found")
    elif option == 5:
        break
    else:
        print("Invalid option, please try again.")
```

Output:

```
○ aadarsh@Aadarshs-Maccy Lab_4 % python3 linkedlist.py
1. Insert
2. Delete
3. Display
4. Search Element
5. Quit
Enter your option: 1
Enter the data to insert: 45
1. Insert
2. Delete
3. Display
4. Search Element
5. Quit
Enter your option: 1
Enter the data to insert: 67
1. Insert
2. Delete
3. Display
4. Search Element
5. Quit
Enter your option: 3
The linked list is:  [45, 67]
1. Insert
2. Delete
3. Display
4. Search Element
5. Quit
Enter your option: 2
Enter the data to delete: 67
1. Insert
2. Delete
3. Display
4. Search Element
5. Quit
Enter your option: 3
The linked list is:  [45]
1. Insert
2. Delete
3. Display
4. Search Element
5. Quit
```

```
Enter your option: 1
Enter the data to insert: 34
1. Insert
2. Delete
3. Display
4. Search Element
5. Quit
Enter your option: 1
Enter the data to insert: 456
1. Insert
2. Delete
3. Display
4. Search Element
5. Quit
Enter your option: 4
Enter the data to search for: 34
Element found at position:  1
1. Insert
2. Delete
3. Display
4. Search Element
5. Quit
Enter your option: 
```

Practical : 05

Implement following programs for doubly linked list:

a. Insertion
b. Deletion
c. Traversing

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

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

    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
            new_node.prev = current

    def search(self, data):
        pos = 0
        current = self.head
        while current:
            if current.data == data:
                return pos
            current = current.next
            pos += 1
        return -1

    def delete(self, data):
            if not self.head:
                return
```

```
        if self.head.data == data:
            self.head = self.head.next
            if self.head:
                self.head.prev = None
            return
        current = self.head
        while current.next:
            if current.next.data == data:
                current.next = current.next.next
                if current.next:
                    current.next.prev = current
                return
            current = current.next

    def display(self):
        elements = []
        current = self.head
        while current:
            elements.append(current.data)
            current = current.next
        return elements

    def displayReserve(self):
        elements = []
        current = self.head
        while current.next:
            current = current.next
        previous = current
        while previous:
            elements.append(previous.data)
            previous = previous.prev
        return elements


if __name__ == "__main__":
    linked_list = LinkedList()
    while True:
        print("1. Insert")
        print("2. Delete")
        print("3. Display")
```

```
print("4. Reverse Display")
print("5. Search Element")
print("6. Quit")
option = int(input("Enter your option: "))
if option == 1:
    data = int(input("Enter the data to insert: "))
    linked_list.insert(data)
elif option == 2:
    data = int(input("Enter the data to delete: "))
    linked_list.delete(data)
elif option == 3:
    print("The linked list is: ", linked_list.display())
elif option == 4:
    print("The Reverse linked list is: ", linked_list.displayReserve())
elif option == 5:
    data = int(input("Enter the data to search for: "))
    result = linked_list.search(data)
    if result != -1:
        print("Element found at position: ", result)
    else:
        print("Element not found")
elif option == 6:
    break
else:
    print("Invalid option, please try again.")
```

Output:

```
aadarsh@Aadarshs-Maccy Lab_5 % python3 double_link.py
1. Insert
2. Delete
3. Display
4. Reverse Display
5. Search Element
6. Quit
Enter your option: 1
Enter the data to insert: 23
1. Insert
2. Delete
3. Display
4. Reverse Display
5. Search Element
6. Quit
Enter your option: 1
Enter the data to insert: 56
1. Insert
2. Delete
3. Display
4. Reverse Display
5. Search Element
6. Quit
Enter your option: 1
Enter the data to insert: 43
1. Insert
2. Delete
3. Display
4. Reverse Display
5. Search Element
6. Quit
Enter your option: 1
Enter the data to insert: 23
1. Insert
2. Delete
3. Display
4. Reverse Display
5. Search Element
6. Quit
Enter your option: 3
The linked list is:  [23, 56, 43, 23]
1. Insert
2. Delete
3. Display
4. Reverse Display
5. Search Element
6. Quit
Enter your option: 4
The Reverse linked list is:  [23, 43, 56, 23]
1. Insert
2. Delete
3. Display
4. Reverse Display
5. Search Element
6. Quit
```

```
Enter your option: 2
Enter the data to delete: 43
1. Insert
2. Delete
3. Display
4. Reverse Display
5. Search Element
6. Quit
Enter your option: 4
The Reverse linked list is:  [23, 56, 23]
1. Insert
2. Delete
3. Display
4. Reverse Display
5. Search Element
6. Quit
Enter your option: 5
Enter the data to search for: 56
Element found at position:  1
1. Insert
2. Delete
3. Display
4. Reverse Display
5. Search Element
6. Quit
Enter your option: 6
```

2. Implement push and pop operations on stack.

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

class Stack:
    def __init__(self):
        self.top = None

    def is_empty(self):
        return self.top is None

    def push(self, data):
        new_node = Node(data)
        new_node.next = self.top
        self.top = new_node

    def pop(self):
        if self.is_empty():
            print("Stack is empty. Cannot pop from an empty stack.")
            return None
        popped_data = self.top.data
        self.top = self.top.next
        return popped_data

    def peek(self):
        if self.is_empty():
            print("Stack is empty. Cannot peek from an empty stack.")
            return None
        return self.top.data

    def display(self):
        elements = []
        current = self.top
        while current:
            elements.append(current.data)
            current = current.next
        return elements
```

```python
if __name__ == "__main__":
    stack = Stack()

    while True:
        print("1. Push")
        print("2. Pop")
        print("3. Peek")
        print("4. Display")
        print("5. Quit")
        option = int(input("Enter your option: "))

        if option == 1:
            data = int(input("Enter the data to push: "))
            stack.push(data)
        elif option == 2:
            popped_data = stack.pop()
            if popped_data is not None:
                print("Popped element: ", popped_data)
        elif option == 3:
            peeked_data = stack.peek()
            if peeked_data is not None:
                print("Top element: ", peeked_data)
        elif option == 4:
            print("Stack elements: ", stack.display())
        elif option == 5:
            break
        else:
            print("Invalid option, please try again.")
```

Output:

```
aadarsh@Aadarshs-Maccy Lab_5 % python3 stack.py
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your option: 1
Enter the data to push: 23
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your option: 1
Enter the data to push: 34
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your option: 1
Enter the data to push: 56
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your option: 2
Popped element:  56
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your option: 4
Stack elements:  [34, 23]
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your option: 2
Popped element:  34
1. Push
2. Pop
3. Peek
4. Display
5. Quit
Enter your option: 4
Stack elements:  [23]
```

3. Write a program to evaluate a postfix expression.

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

class Stack:
    def __init__(self):
        self.top = None

    def is_empty(self):
        return self.top is None

    def push(self, data):
        new_node = Node(data)
        new_node.next = self.top
        self.top = new_node

    def pop(self):
        if self.is_empty():
            print("Stack is empty. Cannot pop from an empty stack.")
            return None
        popped_data = self.top.data
        self.top = self.top.next
        return popped_data

def evaluate_postfix(expression):
    stack = Stack()
    operators = set(['+', '-', '*', '/'])

    for char in expression:
        if char.isdigit():
            stack.push(int(char))
        elif char in operators:
            operand2 = stack.pop()
            operand1 = stack.pop()

            if char == '+':
                result = operand1 + operand2
```

```
            elif char == '-':
                result = operand1 - operand2
            elif char == '*':
                result = operand1 * operand2
            elif char == '/':
                if operand2 == 0:
                    print("Error: Division by zero.")
                    return None
                result = operand1 / operand2

            stack.push(result)

    if stack.is_empty():
        print("Error: Invalid postfix expression.")
        return None

    return stack.pop()

if __name__ == "__main__":
    postfix_expression = input("Enter a postfix expression: ")
    result = evaluate_postfix(postfix_expression)

    if result is not None:
        print("Result of the postfix expression is:", result)
```

Output:

```
aadarsh@Aadarshs-Maccy ADSA % cd Lab_5
aadarsh@Aadarshs-Maccy Lab_5 % python3 expression.py
Enter a postfix expression: 3 8 + 9 8 / -
Result of the postfix expression is: 9.875
aadarsh@Aadarshs-Maccy Lab_5 %
```

Practical : 06

Write the program to print all possible paths between two vertices.

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

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

    def add_edge(self, vertex):
        node = Node(vertex)
        node.next = self.head
        self.head = node

def find_all_paths(graph, start, end, path=[]):
    path = path + [start]
    print(path)
    if start == end:
        return [path]
    if not graph.get(start):
        return []
    paths = []
    current_node = graph[start].head
    print(f'current node: {current_node} for path: {path} where the accumalted final paths
are : {paths}')
    while current_node:
        if current_node.value not in path:
            new_paths = find_all_paths(graph, current_node.value, end, path)
            for p in new_paths:
                paths.append(p)
        current_node = current_node.next
    return paths

def print_all_paths(graph, start, end):
    paths = find_all_paths(graph, start, end)
    if not paths:
        print(f"No path found between {start} and {end}.")
```

```
    else:
        print(f"All paths between {start} and {end}:")
        for path in paths:
            print(" -> ".join(map(str, path)))

# Example usage:
graph = {
    'A': LinkedList(),
    'B': LinkedList(),
    'C': LinkedList(),
    'D': LinkedList(),
    'E': LinkedList(),
    'F': LinkedList(),
    'M': LinkedList(),
}

graph['A'].add_edge('B')
graph['A'].add_edge('C')
graph['A'].add_edge('M')
graph['B'].add_edge('D')
graph['B'].add_edge('E')
graph['B'].add_edge('M')
graph['C'].add_edge('F')
graph['C'].add_edge('M')
graph['D'].add_edge('B')
graph['E'].add_edge('F')
graph['F'].add_edge('E')
graph['F'].add_edge('C')
graph['M'].add_edge('A')
graph['M'].add_edge('B')
graph['M'].add_edge('C')
graph['M'].add_edge('F')

start = input('Please enter the starting point for the graph: ')
end = input('Please enter the ending point for the graph: ')
print_all_paths(graph, start, end)
```

Output:

```
(base) aadarsh@Aadarshs-Maccy Lab_6 % python3 paths_all.py
Please enter the starting point for the graph: A
Please enter the ending point for the graph: F
All paths between A and F:
A -> M -> F
A -> M -> C -> F
A -> M -> B -> E -> F
A -> C -> M -> F
A -> C -> M -> B -> E -> F
A -> C -> F
A -> B -> M -> F
A -> B -> M -> C -> F
A -> B -> E -> F
(base) aadarsh@Aadarshs-Maccy Lab_6 %
```

2. Write the program to print all possible paths between two vertices.

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

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

    def add_node(self, value):
        node = Node(value)
        node.next = self.head
        self.head = node

def add_edge(graph, u, v):
    if u not in graph:
        graph[u] = LinkedList()
    graph[u].add_node(v)

def print_equidistant_nodes(graph, root):
    if root not in graph:
        print(f"Node {root} not found in the graph.")
        return

    queue = [(root, 0)]  # Tuple containing current node and its distance from the root
    result = {}

    while queue:
        current_node, current_distance = queue.pop(0)

        if current_node in graph:
            current_neighbor = graph[current_node].head
            while current_neighbor:
                neighbor_value = current_neighbor.value
                queue.append((neighbor_value, current_distance + 1))

                if current_distance + 1 not in result:
                    result[current_distance + 1] = [neighbor_value]
```

```
                else:
                    result[current_distance + 1].append(neighbor_value)

                current_neighbor = current_neighbor.next

    for distance, nodes in result.items():
        print(f"Nodes at distance {distance} from root {root}: {', '.join(map(str, nodes))}")


graph = {}


# while True:
#     node = input('Add a New node to the graph')
#     check = 'Y'
#     while check == 'Y':
#         edge = input('Add a Edge to Node {node}: ')
#         add_edge(graph, node, edge)
#         check = input('Want to Add More  Edges? (y/n): ').upper()
#     cont = input('Want to Add More Nodes? (y/n): ').upper()
#     if (cont == 'N'):
#         break

add_edge(graph, 'A', 'B')
add_edge(graph, 'A', 'C')
add_edge(graph, 'B', 'D')
add_edge(graph, 'B', 'E')
add_edge(graph, 'C', 'F')
add_edge(graph, 'C', 'G')
add_edge(graph, 'E', 'M')

print_equidistant_nodes(graph, 'A')
```

```
(base) aadarsh@Aadarshs-Maccy Lab_6 % python3 distance_group.py
Nodes at distance 1 from root A: C, B
Nodes at distance 2 from root A: G, F, E, D
Nodes at distance 3 from root A: M
(base) aadarsh@Aadarshs-Maccy Lab_6 %
```

Practical : 07

Write the program to check the presence of articulation points in the graph.

Code:

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.adj = [[] for _ in range(vertices)]
        self.time = 0

    def add_edge(self, u, v):
        self.adj[u].append(v)
        self.adj[v].append(u)

    def APUtil(self, u, visited, disc, low, parent, ap):
        children = 0
        visited[u] = True
        disc[u] = self.time
        low[u] = self.time
        self.time += 1

        for v in self.adj[u]:
            if not visited[v]:
                children += 1
                parent[v] = u
                self.APUtil(v, visited, disc, low, parent, ap)
                low[u] = min(low[u], low[v])

                if parent[u] == -1 and children > 1:
                    ap[u] = True

                if parent[u] != -1 and low[v] >= disc[u]:
                    ap[u] = True
            elif v != parent[u]:
                low[u] = min(low[u], disc[v])

    def AP(self):
        visited = [False] * self.V
        disc = [float("inf")] * self.V
        low = [float("inf")] * self.V
```

```
        parent = [-1] * self.V
        ap = [False] * self.V

        for i in range(self.V):
            if not visited[i]:
                self.APUtil(i, visited, disc, low, parent, ap)

        print("Articulation Points in the Graph:")
        for i in range(self.V):
            if ap[i]:
                print(i, end=" ")


if __name__ == "__main__":
    g = Graph(6)
    g.add_edge(1, 0)
    g.add_edge(0, 2)
    g.add_edge(2, 1)
    g.add_edge(0, 3)
    g.add_edge(3, 4)
    g.add_edge(2, 5)
    print("Articulation Points in the given graph:")
    g.AP()
```

Output:

```
0 3 %
(base) aadarsh@Aadarshs-Maccy ADSA % /usr/local/bin/python3 "/Users/aadarsh/Documents/NITJ/Semester 2/ADSA/Lab_8/ap.py"
Articulation Points in the given graph:
Articulation Points in the Graph:
0 2 3 %
(base) aadarsh@Aadarshs-Maccy ADSA %
```

2. Write the program to construct a binary tree.

Code:
```
#include <iostream>
using namespace std;

class tree {
public:
    // Constructor to initialize the node with given data
    tree(int data) : data_(data), left_(nullptr), right_(nullptr) {}

    // Member function to access the data stored in the node
    int getData() const { return data_; }

    // Member function to access the left subtree
    tree* getLeft() const { return left_; }

    // Member function to access the right subtree
    tree* getRight() const { return right_; }

    // Setter function to set the left subtree
    void setLeft(tree* left) { left_ = left; }

    // Setter function to set the right subtree
    void setRight(tree* right) { right_ = right; }

private:
    // Data stored in the node
    int data_;

    // Pointers to the left and right subtrees
    tree* left_;
    tree* right_;
};

tree* constructTree() {
    int data;
    cout << "Enter data for the node (enter -1 for NULL): ";
    cin >> data;
```

```
    if (data == -1) {
        return nullptr;
    }

    tree* newNode = new tree(data);
    cout << "Enter left child of " << data << ":\n";
    newNode->setLeft(constructTree());
    cout << "Enter right child of " << data << ":\n";
    newNode->setRight(constructTree());

    return newNode;
}

void printInorder(tree* root) {
    if (root != nullptr) {
        printInorder(root->getLeft());
        cout << root->getData() << " ";
        printInorder(root->getRight());
    }
}

int main() {
    // Construct the binary tree based on user input
    cout << "Enter data for the root node:\n";
    tree* root = constructTree();

    // Print the binary tree in inorder traversal
    cout << "Inorder traversal of the binary tree:\n";
    printInorder(root);

    return 0;
}
```

Output:

```
(base) aadarsh@Aadarshs-Maccy output % ./"tree"
Enter data for the root node:
Enter data for the node (enter -1 for NULL): 10
Enter left child of 10:
Enter data for the node (enter -1 for NULL): 7
Enter left child of 7:
Enter data for the node (enter -1 for NULL): 3
Enter left child of 3:
Enter data for the node (enter -1 for NULL): -1
Enter right child of 3:
Enter data for the node (enter -1 for NULL): -1
Enter right child of 7:
Enter data for the node (enter -1 for NULL): 5
Enter left child of 5:
Enter data for the node (enter -1 for NULL): -1
Enter right child of 5:
Enter data for the node (enter -1 for NULL): -1
Enter right child of 10:
Enter data for the node (enter -1 for NULL): 12
Enter left child of 12:
Enter data for the node (enter -1 for NULL): 11
Enter left child of 11:
Enter data for the node (enter -1 for NULL): -1
Enter right child of 11:
Enter data for the node (enter -1 for NULL): -1
Enter right child of 12:
Enter data for the node (enter -1 for NULL): 15
Enter left child of 15:
Enter data for the node (enter -1 for NULL): -1
Enter right child of 15:
Enter data for the node (enter -1 for NULL): -1
Inorder traversal of the binary tree:
3 7 5 10 11 12 15 %
```

Practical : 08

Implement following algorithms:

- **BST insertion**
- **BST Deletion**
- **BST Traversal**

```
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None

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

    def insert(self, val):
        if not self.root:
            self.root = TreeNode(val)
        else:
            self._insert_helper(self.root, val)

    def _insert_helper(self, node, val):
        if val < node.val:
            if node.left is None:
                node.left = TreeNode(val)
            else:
                self._insert_helper(node.left, val)
        else:
            if node.right is None:
                node.right = TreeNode(val)
            else:
                self._insert_helper(node.right, val)

    def delete(self, val):
        self.root = self._delete_helper(self.root, val)

    def _delete_helper(self, root, val):
        if root is None:
            return root
```

```python
        if val < root.val:
            root.left = self._delete_helper(root.left, val)
        elif val > root.val:
            root.right = self._delete_helper(root.right, val)
        else:
            if root.left is None:
                return root.right
            elif root.right is None:
                return root.left

            temp = self._min_value_node(root.right)
            root.val = temp.val
            root.right = self._delete_helper(root.right, temp.val)

        return root

    def _min_value_node(self, node):
        current = node
        while current.left is not None:
            current = current.left
        return current

    def inorder_traversal(self):
        result = []
        self._inorder_traversal_helper(self.root, result)
        return result

    def _inorder_traversal_helper(self, node, result):
        if node:
            self._inorder_traversal_helper(node.left, result)
            result.append(node.val)
            self._inorder_traversal_helper(node.right, result)

# Example usage:
bst = BST()
bst.insert(5)
bst.insert(3)
bst.insert(7)
bst.insert(2)
```

```
bst.insert(4)
bst.insert(6)
bst.insert(8)

print("Inorder Traversal:", bst.inorder_traversal())

bst.delete(3)
print("After deleting 3, Inorder Traversal:", bst.inorder_traversal())
```

Output :

```
(base) aadarsh@Aadarshs-Maccy Lab_9 % python3 bst.py
Inorder Traversal: [2, 3, 4, 5, 6, 7, 8]
After deleting 3, Inorder Traversal: [2, 4, 5, 6, 7, 8]
(base) aadarsh@Aadarshs-Maccy Lab_9 %
```

Practical : 09

Implement AVL tree insertion and deletion.

Code:

```
class AVLNode:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
        self.height = 1


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

    def height(self, node):
        if node is None:
            return 0
        return node.height

    def update_height(self, node):
        node.height = 1 + max(self.height(node.left), self.height(node.right))

    def balance(self, node):
        if node is None:
            return 0
        return self.height(node.left) - self.height(node.right)

    def rotate_right(self, y):
        x = y.left
        T2 = x.right

        x.right = y
        y.left = T2

        self.update_height(y)
        self.update_height(x)
```

```
        return x

    def rotate_left(self, x):
        y = x.right
        T2 = y.left

        y.left = x
        x.right = T2

        self.update_height(x)
        self.update_height(y)

        return y

    def insert(self, node, key):
        if node is None:
            return AVLNode(key)
        if key < node.key:
            node.left = self.insert(node.left, key)
        else:
            node.right = self.insert(node.right, key)

        self.update_height(node)

        balance = self.balance(node)

        if balance > 1:
            if key < node.left.key:
                return self.rotate_right(node)
            else:
                node.left = self.rotate_left(node.left)
                return self.rotate_right(node)

        if balance < -1:
            if key > node.right.key:
                return self.rotate_left(node)
            else:
                node.right = self.rotate_right(node.right)
                return self.rotate_left(node)
```

```
        return node

    def min_value_node(self, node):
        current = node
        while current.left is not None:
            current = current.left
        return current

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

        if key < root.key:
            root.left = self.delete(root.left, key)
        elif key > root.key:
            root.right = self.delete(root.right, key)
        else:
            if root.left is None:
                temp = root.right
                root = None
                return temp
            elif root.right is None:
                temp = root.left
                root = None
                return temp

            temp = self.min_value_node(root.right)
            root.key = temp.key
            root.right = self.delete(root.right, temp.key)

        if root is None:
            return root

        self.update_height(root)

        balance = self.balance(root)

        if balance > 1:
            if self.balance(root.left) >= 0:
                return self.rotate_right(root)
```

```
            else:
                root.left = self.rotate_left(root.left)
                return self.rotate_right(root)

        if balance < -1:
            if self.balance(root.right) <= 0:
                return self.rotate_left(root)
            else:
                root.right = self.rotate_right(root.right)
                return self.rotate_left(root)

        return root

    def preorder_traversal(self, root):
        if root is not None:
            print(root.key, end=" ")
            self.preorder_traversal(root.left)
            self.preorder_traversal(root.right)


# Example usage:

avl_tree = AVLTree()
avl_tree.root = avl_tree.insert(avl_tree.root, 10)
avl_tree.root = avl_tree.insert(avl_tree.root, 20)
avl_tree.root = avl_tree.insert(avl_tree.root, 30)
avl_tree.root = avl_tree.insert(avl_tree.root, 40)
avl_tree.root = avl_tree.insert(avl_tree.root, 50)
avl_tree.root = avl_tree.insert(avl_tree.root, 25)

print("Preorder traversal of the constructed AVL tree is:")
avl_tree.preorder_traversal(avl_tree.root)

print("\nDelete 20")
avl_tree.root = avl_tree.delete(avl_tree.root, 20)

print("Preorder traversal of the AVL tree after deletion of 20:")
avl_tree.preorder_traversal(avl_tree.root)
```

```
(base) aadarsh@Aadarshs-Maccy Lab_10 % python3 avl.py
Preorder traversal of the constructed AVL tree is:
30 20 10 25 40 50
Delete 20
Preorder traversal of the AVL tree after deletion of 20:
30 25 10 40 50 %
(base) aadarsh@Aadarshs-Maccy Lab_10 %
```

Practical : 10

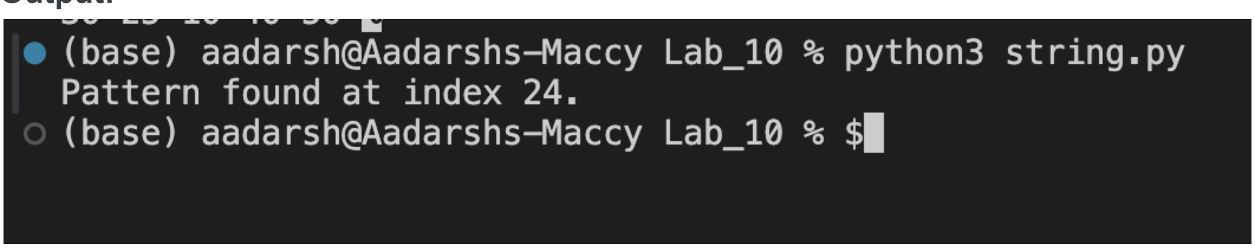
Implement following programs:
1. String matching with a pattern using brute force approach.
2. Max heap insertion, Deletion, and sorting. Print the numbers present in the heap in sorted order.

String Matching

Code:

```
def brute_force_string_matching(text, pattern):
    n = len(text)
    m = len(pattern)

    for i in range(n - m + 1):
        j = 0
        while j < m and text[i + j] == pattern[j]:
            j += 1
        if j == m:
            return i  # pattern found at index i
    return -1  # pattern not found


# Example usage:

text = "ADARSH KUMAR GUPTA FROM NIT JALANDHAR"
pattern = "NIT"
index = brute_force_string_matching(text, pattern)

if index != -1:
    print(f"Pattern found at index {index}.")
else:
    print("Pattern not found.")
```

Output:



```
(base) aadarsh@Aadarshs-Maccy Lab_10 % python3 string.py
Pattern found at index 24.
(base) aadarsh@Aadarshs-Maccy Lab_10 % $
```

Max Heap Implementation

Code:

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

    def parent(self, i):
        return (i - 1) // 2

    def insert(self, key):
        self.heap.append(key)
        i = len(self.heap) - 1
        while i != 0 and self.heap[self.parent(i)] < self.heap[i]:
            self.heap[i], self.heap[self.parent(i)] = self.heap[self.parent(i)], self.heap[i]
            i = self.parent(i)

    def delete(self, key):
        i = self.heap.index(key)
        self.increase_key(i, float('inf'))
        self.extract_max()

    def extract_max(self):
        if len(self.heap) == 0:
            return None
        if len(self.heap) == 1:
            return self.heap.pop()

        root = self.heap[0]
        self.heap[0] = self.heap.pop()
        self.max_heapify(0)
        return root

    def max_heapify(self, i):
        left = 2 * i + 1
        right = 2 * i + 2
        largest = i

        if left < len(self.heap) and self.heap[left] > self.heap[largest]:
```

```
            largest = left

        if right < len(self.heap) and self.heap[right] > self.heap[largest]:
            largest = right

        if largest != i:
            self.heap[i], self.heap[largest] = self.heap[largest], self.heap[i]
            self.max_heapify(largest)

    def increase_key(self, i, new_key):
        if new_key < self.heap[i]:
            return
        self.heap[i] = new_key
        while i != 0 and self.heap[self.parent(i)] < self.heap[i]:
            self.heap[i], self.heap[self.parent(i)] = self.heap[self.parent(i)], self.heap[i]
            i = self.parent(i)

    def heap_sort(self):
        n = len(self.heap)
        for i in range(n - 1, 0, -1):
            self.heap[i], self.heap[0] = self.heap[0], self.heap[i]
            self.max_heapify_subtree(0, i)
        return self.heap

    def max_heapify_subtree(self, i, n):
        left = 2 * i + 1
        right = 2 * i + 2
        largest = i

        if left < n and self.heap[left] > self.heap[largest]:
            largest = left

        if right < n and self.heap[right] > self.heap[largest]:
            largest = right

        if largest != i:
            self.heap[i], self.heap[largest] = self.heap[largest], self.heap[i]
            self.max_heapify_subtree(largest, n)
```

```
# Example usage:

heap = MaxHeap()
heap.insert(10)
heap.insert(20)
heap.insert(15)
heap.insert(40)
heap.insert(50)
heap.insert(100)

print("Heap elements after insertion:", heap.heap)

heap.delete(40)
print("Heap elements after deleting 40:", heap.heap)

sorted_elements = heap.heap_sort()
print("Sorted elements from the heap:", sorted_elements)
```

Output:

```
(base) aadarsh@Aadarshs-Maccy Lab_10 % python3 heap.py
Heap elements after insertion: [100, 40, 50, 10, 20, 15]
Heap elements after deleting 40: [100, 20, 50, 10, 15]
Sorted elements from the heap: [10, 15, 20, 50, 100]
(base) aadarsh@Aadarshs-Maccy Lab_10 %
```