

Sorting Techniques

Bubble sort

```
import java.util.Scanner;

public class bubbleSort {
    static int[] arr;
    int size;

    void insert() {

        System.out.println("Enter the size of the array : ");
        Scanner sc = new Scanner(System.in);
        size = sc.nextInt();
        arr = new int[size];
        for(int i = 0; i < size; i++) {
            System.out.println("Enter the element : ");
            int element = sc.nextInt();
            arr[i] = element;
        }
    }

    void bubblesort() {
        int i,j,temp;
        for(i = 0; i < size; i++) {
            for(j = 0; j < size - 1; j++) {
                if(arr[j] > arr[j+1]) {
                    temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }

        System.out.println("Elements sorted");
    }

    void display() {
        for(int i = 0; i < arr.length; i++) {
            System.out.println(arr[i] + " ");
        }
    }

    public static void main(String[] args) {
        System.out.println("Pranali B36 ");
        bubbleSort b1 = new bubbleSort();
        Scanner sc = new Scanner(System.in);
        int option;
```

```

char ch = 'y';

while(ch == 'y') {
    System.out.println("1. Insert Element");
    System.out.println("2. Sort array");
    System.out.println("3. Display array");

    System.out.println("Enter option : ");
    option = sc.nextInt();

    switch(option) {
        case 1 : {
            b1.insert();
            System.out.println("Do you want to continue ? (y / n)");
            break;
        }
        case 2 : {
            b1.bubblesort();
            System.out.println("Do you want to continue ? (y / n)");
            break;
        }
        case 3 : {
            b1.display();
            System.out.println("Do you want to continue ? (y / n)");
            break;
        }
    }
    ch = sc.next().charAt(0);
}
}
}

```

OUTPUT :

Pranali B36

1. Insert Element
2. Sort array
3. Display array

Enter option :

1

Enter the size of the array :

4

Enter the element :

9

Enter the element :

6

Enter the element :

7

Enter the element :

8

Do you want to continue ? (y / n)

y

1. Insert Element
2. Sort array
3. Display array

Enter option :

3

9

6

7

8

Do you want to continue ? (y / n)

y

1. Insert Element
2. Sort array
3. Display array

Enter option :

2

Elements sorted

Do you want to continue ? (y / n)

y

1. Insert Element
2. Sort array
3. Display array

Enter option :

3

6

7

8

9

Do you want to continue ? (y / n)

n

Shell sort

```
import java.util.Scanner;
public class shellSort {
    static int[] arr;
    int size;

    void insert() {

        System.out.println("Enter the size of the array : ");
        Scanner sc = new Scanner(System.in);
        size = sc.nextInt();
        arr = new int[size];
        for(int i = 0; i < size; i++) {
            System.out.println("Enter the element : ");
            int element = sc.nextInt();
            arr[i] = element;
        }
    }

    void shellsort() {
        int temp;
        for(int gap = size / 2; gap > 0; gap /=2) {
            for(int i = gap; i < size; i++) {
                temp = arr[i];
                int j;
                for(j = i; j >= gap && arr[j-gap] > temp; j-= gap) {
                    arr[j] = arr[j- gap];
                }

                arr[j] = temp;
            }
        }

        System.out.println("Elements sorted");
    }

    void display() {
        for(int i = 0; i < arr.length; i++) {
            System.out.println(arr[i] + " ");
        }
    }

    public static void main(String[] args) {
        System.out.println("Pranali B36 ");
        shellSort b1 = new shellSort();
        Scanner sc = new Scanner(System.in);
    }
}
```

```

int option;
char ch = 'y';

while(ch == 'y') {
    System.out.println("1. Insert Element");
    System.out.println("2. Sort array");
    System.out.println("3. Display array");

    System.out.println("Enter option : ");
    option = sc.nextInt();

    switch(option) {
        case 1 : {
            b1.insert();
            System.out.println("Do you want to continue ? (y / n)");
            break;
        }
        case 2 :{
            b1.shellsort();
            System.out.println("Do you want to continue ? (y / n)");
            break;
        }
        case 3 : {
            b1.display();
            System.out.println("Do you want to continue ? (y / n)");
            break;
        }
    }
    ch = sc.next().charAt(0);
}
}
}

```

OUTPUT :

```
Problems @ Javadoc Declaration Console ✕
shellSort [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (13-Dec-2024, 11:21:17 pm)
Pranali B36
1. Insert Element
2. Sort array
3. Display array
Enter option :
1
Enter the size of the array :
4
Enter the element :
8
Enter the element :
6
Enter the element :
7
Enter the element :
5
Do you want to continue ? (y / n)
y
1. Insert Element
2. Sort array
3. Display array
Enter option :
3
8
6
7
5
Do you want to continue ? (y / n)
y
1. Insert Element
2. Sort array
3. Display array
Enter option :
2
Elements sorted
Do you want to continue ? (y / n)
y
```

```

1. Insert Element
2. Sort array
3. Display array
Enter option :
3
5
6
7
8
Do you want to continue ? (y / n)

```

Selection Sort

```

import java.util.Scanner;
public class selectionSort {
    static int[] arr;
    int size;
    void insert() {
        System.out.println("Enter the size of the array: ");
        Scanner sc = new Scanner(System.in);
        size = sc.nextInt();
        arr = new int[size];
        for(int i = 0; i < size; i++) {
            System.out.println("Enter the element at index " + i + ": ");
            int element = sc.nextInt();
            arr[i] = element;
        }
    }
    void display() {
        System.out.println("Array elements are: ");
        for(int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
    void selectionsort() {
        int i, j, temp;
        for(i = 0; i < size - 1; i++) { // Run till size-1 for optimization
            for(j = i + 1; j < size; j++) {
                if(arr[j] < arr[i]) {
                    temp = arr[i];
                    arr[i] = arr[j];

```



```

        arr[j] = temp;
    }
}
}
System.out.println("Elements sorted successfully.");
}
public static void main(String[] args) {
    System.out.println("Pranali B36 ");

    selectionSort b1 = new selectionSort();
    Scanner sc = new Scanner(System.in);
    int option;
    char ch = 'y';

    while(ch == 'y' || ch == 'Y') { // Added support for both lowercase and uppercase 'Y'
        System.out.println("\nMenu:");
        System.out.println("1. Insert Elements");
        System.out.println("2. Sort Array");
        System.out.println("3. Display Array");
        System.out.println("Enter option: ");

        option = sc.nextInt();

        switch(option) {
            case 1:
                b1.insert();
                break;
            case 2:
                if(arr != null) {
                    b1.selectionsort();
                } else {
                    System.out.println("Array is empty. Please insert elements first.");
                }
                break;
            case 3:
                if(arr != null) {
                    b1.display();
                } else {
                    System.out.println("Array is empty. Please insert elements first.");
                }
                break;
            default:
                System.out.println("Invalid option. Please select a valid option.");
                break;
        }
    }
}

```

```
    }

    System.out.println("Do you want to continue? (y / n)");
    ch = sc.next().charAt(0);
}

sc.close(); // Close the scanner to prevent resource leaks
}
```

OUTPUT :

Menu:

1. Insert Elements
2. Sort Array
3. Display Array

Enter option:

1

Enter the size of the array:

4

Enter the element at index 0:

7

Enter the element at index 1:

8

Enter the element at index 2:

5

Enter the element at index 3:

4

Do you want to continue? (y / n)

y

Menu:

1. Insert Elements
2. Sort Array
3. Display Array

Enter option:

3

Array elements are:

7 8 5 4

Do you want to continue? (y / n)

y

Menu:

1. Insert Elements
2. Sort Array
3. Display Array

Enter option:

2

Elements sorted successfully.

Do you want to continue? (y / n)

y

Menu:

1. Insert Elements
2. Sort Array
3. Display Array

Enter option:

3

Array elements are:

4 5 7 8

Insertion Sort

```
package searching_technic;
import java.util.Scanner;
public class InsertionSort {
    static int[] arr;
    int size;
    void insert() {
        System.out.println("Enter the size of the array : ");
        Scanner sc = new Scanner(System.in);
        size = sc.nextInt();
        arr = new int[size];
        for(int i = 0; i < size; i++) {
            System.out.println("Enter the element : ");
            int element = sc.nextInt();
            arr[i] = element;
        }
    }
    void isort() {
        for(int i = 1; i < size; i++) {
            int key = arr[i];
            int j;
            for(j = i - 1; j >= 0 && arr[j] > key; j--) {
                arr[j + 1] = arr[j];
            }
            arr[j + 1] = key;
        }
        System.out.println("Elements sorted");
    }
    void display() {
        for(int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        System.out.println("Pranali B36 ");

        InsertionSort b1 = new InsertionSort();
        Scanner sc = new Scanner(System.in);
        int option;
        char ch = 'y';
```

```

while(ch == 'y' || ch == 'Y') {
    System.out.println("1. Insert Element");
    System.out.println("2. Sort array");
    System.out.println("3. Display array");
    System.out.println("Enter option : ");
    option = sc.nextInt();

    switch(option) {
        case 1 : {
            b1.insert();
            System.out.println("Do you want to continue ? (y / n)");
            break;
        }
        case 2 :{
            b1.isort();
            System.out.println("Do you want to continue ? (y / n)");
            break;
        }
        case 3 : {
            b1.display();
            System.out.println("Do you want to continue ? (y / n)");
            break;
        }
        default: {
            System.out.println("Invalid option. Please try again.");
        }
    }
    ch = sc.next().charAt(0);
}
sc.close();
}
}

```

OUTPUT :

Pranali B36

1. Insert Element

2. Sort array

3. Display array

Enter option :

1

Enter the size of the array :

4

Enter the element :

5

Enter the element :

8

Enter the element :

9

Enter the element :

7

Do you want to continue ? (y / n)

y

1. Insert Element

2. Sort array

3. Display array

Enter option :

3

5 8 9 7

Do you want to continue ? (y / n)

y

1. Insert Element

2. Sort array

3. Display array

Enter option :

2

Elements sorted

Do you want to continue ? (y / n)

y

1. Insert Element

2. Sort array

3. Display array

Enter option :

3

5 7 8 9

Searching Technique

Linear search

```
import java.util.Scanner;

public class LinearSearch {
    public static int linearSearch(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {

            if (arr[i] == target) {
                return i;
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] arr = { 64, 34, 25, 12, 22, 11, 90 };

        System.out.println("Pranali B36");

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number to search: ");
        int target = scanner.nextInt();

        int result = linearSearch(arr, target);

        if (result == -1) {
            System.out.println("Element not found in the array.");
        } else {
            System.out.println("Element found at index: " + result);
        }
        scanner.close();
    }
}
```

OUTPUT :



```
Console X
<terminated> LinearSearch [Java Application] C:\Users\Admin\p2\prof\plugins\org.eclipse.j
Pranali B36
Enter the number to search: 11
Element found at index: 5
```


Binary search

```
import java.util.Scanner;

public class BinarySearch {
    public static int binarySearch(int[] arr, int target) {

        int left = 0;
        int right = arr.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (arr[mid] == target) {
                return mid;
            }

            if (arr[mid] > target) {
                right = mid - 1;
            }
            else {
                left = mid + 1;
            }
        }

        return -1;
    }

    public static void main(String[] args) {
        int[] arr = { 11, 22, 34, 45, 64, 90, 100 };
        System.out.println("Pranali B36 ");

        Scanner scanner = new Scanner(System.in);

        System.out.println("Original array: ");
        for (int i : arr) {
            System.out.print(i + " ");
        }
        System.out.println();

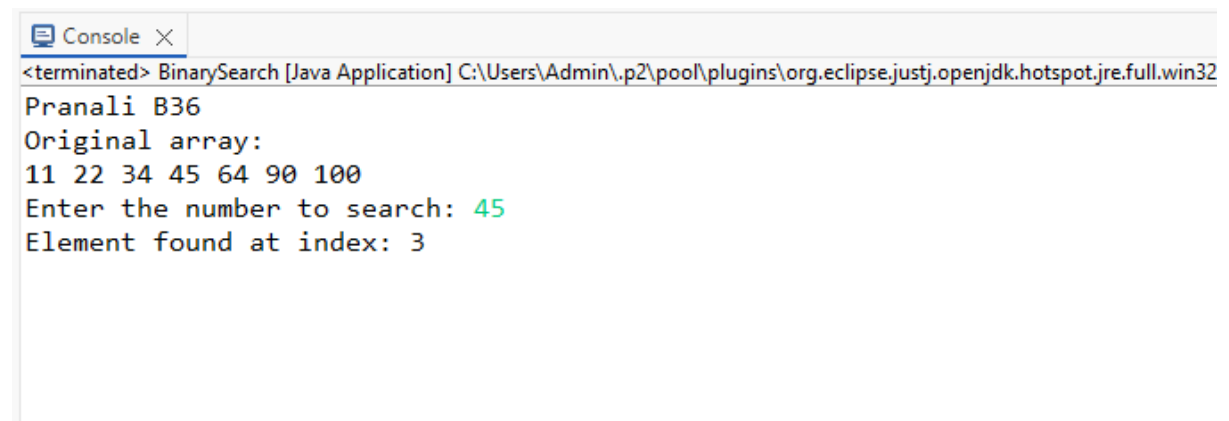
        System.out.print("Enter the number to search: ");
```

```
int target = scanner.nextInt();

int result = binarySearch(arr, target);

if (result == -1) {
    System.out.println("Element not found in the array.");
} else {
    System.out.println("Element found at index: " + result);
}
scanner.close();
}
}
```

OUTPUT :



```
Console ×
<terminated> BinarySearch [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32
Pranali B36
Original array:
11 22 34 45 64 90 100
Enter the number to search: 45
Element found at index: 3
```

PRACTICAL 2

1. Write a program for Modulo division.

CODE :

```
import java.util.Scanner;

public class ModuloDivision {

    public static void main(String[] args) {        Scanner
scanner = new Scanner(System.in);

        System.out.println("Pranali B36 ");

        System.out.print("Enter the dividend: ");        int
dividend = scanner.nextInt();

        System.out.print("Enter the divisor: ");        int
divisor = scanner.nextInt();        int result = dividend
% divisor;

        System.out.println("The remainder when " + dividend + " is divided by " + divisor + "
is: " + result);    scanner.close();
    }
}
```

OUTPUT :

```
<terminated> ModuloDivision [Java Application] C:\java\New folder\bin\javaw.exe (04-Dec-2024, 2:31:35 pm)
Pranali B36
Enter the dividend: 17
Enter the divisor: 5
The remainder when 17 is divided by 5 is: 2
```

2. Write a program for digit extraction

CODE :

```
import java.util.Scanner; public
class DigitExtraction {

    public static void main(String[] args) {        Scanner
scanner = new Scanner(System.in);
        System.out.println("Pranali B36 ");

        System.out.print("Enter a string with numbers: ");
        String input = scanner.nextLine();

        System.out.println("Digits present in the string are : ");        for (int i
= 0; i < input.length(); i++) {            char ch = input.charAt(i);
if (Character.isDigit(ch)) {
                System.out.print(ch);
            }
        }
        scanner.close();
    }
```

```
}
```

OUTPUT

:

```
<terminated> DigitExtraction [Java Application] C:\java\New folder\bin\javaw.exe (04-Dec-2024, 2:35:16 pm)
Pranali B36
Enter a string with numbers: 12345pranali
Digits present in the string are :
12345
```

3. Write a program for Linear probe for collision resolution.

```
package ads;
```

```
import java.util.Scanner;
```

```
public class LinearProbing {
```

```
    static class HashTable {
        int[] table;    int size;
```

```

        // Constructor to initialize hash table        public
HashTable(int size) {        this.size = size;        table
= new int[size];        for (int i = 0; i < size; i++) {
table[i] = -1; // Using -1 to indicate empty slots
        }
    }

    // Hash function
    private int hashFunction(int key) {
return key % size;
    }

    // Insert a key into the hash table
public void insert(int key) {        int
index = hashFunction(key);        int
startIndex = index;

        // Linear probing for collision resolution        while
(table[index] != -1) {        index = (index + 1) % size; //
Move to the next slot        if (index == startIndex) { // If
we return to the start, table is full
System.out.println("Hash table is full. Cannot insert " + key);
return;
        }
    }
    table[index] = key;
    System.out.println("Inserted " + key + " at index " + index);
}

    // Search for a key in the hash table
public boolean search(int key) {
int index = hashFunction(key);        int
startIndex = index;

        // Linear probing to find the key
while (table[index] != -1) {        if
(table[index] == key) {
return true; // Key found
        }
        index = (index + 1) % size;        if (index == startIndex) { // If we
return to the start, key is not in the table        break;

```

```

        }
    }
    return false; // Key not found
}

// Display the hash table
public void display() {
    System.out.println("Hash Table:");
    for (int i = 0; i < size; i++) {
        System.out.println("Index " + i + ": " + (table[i] == -1 ? "Empty" : table[i]));
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Pranali B36 ");

    System.out.println("Enter the size of the hash table:");
    int size = scanner.nextInt();
    HashTable hashTable = new HashTable(size);

    while (true) {
        System.out.println("\nChoose an operation:");
        System.out.println("1. Insert");
        System.out.println("2. Search");
        System.out.println("3. Display");
        System.out.println("4. Exit");
        int choice = scanner.nextInt();

        switch (choice) {
            case 1:
                System.out.println("Enter the key to insert:");
                int key = scanner.nextInt();
                hashTable.insert(key);
                break;
            case 2:
                System.out.println("Enter the key to search:");
                key = scanner.nextInt();
                if (hashTable.search(key)) {
                    System.out.println("Key " + key + " found in the hash table.");
                } else {

```

```
                System.out.println("Key " + key + " not found in the hash table.");
            }
            break;
case 3:
    hashTable.display();
    break;
case 4:
    System.out.println("Exiting...");
    scanner.close();
    return;
default:
    System.out.println("Invalid choice. Try again.");
}
}
}
```


OUTPUT :

Pranali B36

Enter the size of the hash table:

5

Choose an operation:

1. Insert
2. Search
3. Display
4. Exit

1

Enter the key to insert:

8

Inserted 8 at index 3

Choose an operation:

1. Insert
2. Search
3. Display
4. Exit

2

Enter the key to search:

8

Key 8 found in the hash table.

Choose an operation:

1. Insert
2. Search
3. Display
4. Exit

3

Hash Table:

Index 0: Empty

Index 1: Empty

Index 2: Empty

Index 3: 8

Index 4: Empty

Choose an operation:

1. Insert
2. Search
3. Display
4. Exit

4

Exiting...

PRACTICAL 3 :

1) Write a program for array implementation of stack.

```
public class StackUsingArray {
    private int[] stack;    private int
    top;
    private int capacity;

    public StackUsingArray(int capacity) {
        this.capacity = capacity;    stack
    = new int[capacity];
        top = -1;
    }

    public boolean isFull() {
        return top == capacity - 1;
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public void push(int item) {
        if (isFull()) {
            System.out.println("Stack Overflow! Cannot add more items.");
        } else {
            stack[++top] = item;
            System.out.println(item + " pushed onto stack.");
        }
    }

    public int pop() {
        if (isEmpty()) {
```

```

        System.out.println("Stack Underflow! No items to pop.");
return -1;
    } else {
        int poppedItem = stack[top--];
        System.out.println(poppedItem + " popped from stack.");    return
poppedItem;
    }
}

```

```

    public int peek() {
if (isEmpty()) {
    System.out.println("Stack is empty.");
return -1;    } else {
    return stack[top];
}
}

```

```

    public void display() {
if (isEmpty()) {
    System.out.println("Stack is empty.");
} else {
    System.out.print("Stack elements: ");
for (int i = 0; i <= top; i++) {
    System.out.print(stack[i] + " ");
}
    System.out.println();
}
}

```

```

    public static void main(String[] args) {
        System.out.println("Pranali B36");
        StackUsingArray stack = new StackUsingArray(5);

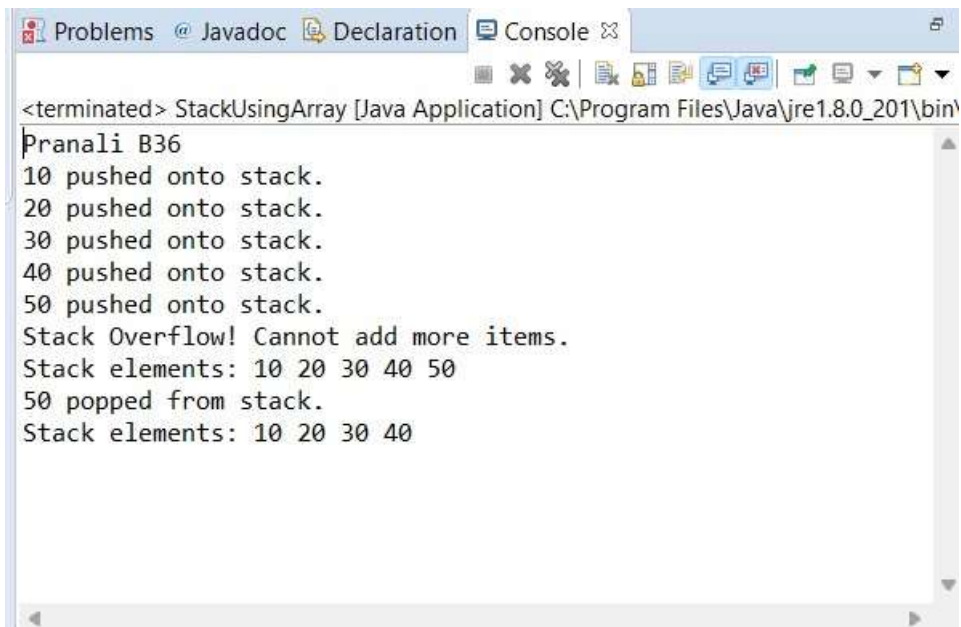
        stack.push(10);    stack.push(20);
stack.push(30);    stack.push(40);
stack.push(50);    stack.push(60);

        stack.display();
    }
}

```

```
        stack.pop();    stack.peak();  
stack.display();  
    }  
}
```

OUTPUT :



```
<terminated> StackUsingArray [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\  
Pranali B36  
10 pushed onto stack.  
20 pushed onto stack.  
30 pushed onto stack.  
40 pushed onto stack.  
50 pushed onto stack.  
Stack Overflow! Cannot add more items.  
Stack elements: 10 20 30 40 50  
50 popped from stack.  
Stack elements: 10 20 30 40
```

2) Write a program for array implementation of ordinary and circular queue

i) CIRCULAR QUEUE :

```
public class CircularQueue {  
    private int[] queue;  
    private int front;  
    private int rear;  
    private int capacity;  
    private int size;  
  
    public CircularQueue(int capacity) {  
        this.capacity = capacity;  
        queue = new int[capacity]  
        front = -1;  
        rear = -1;  
        size = 0;  
    }  
  
    public boolean isFull() {  
  
        return size == capacity;  
    }  
  
    public boolean isEmpty() {  
        return size == 0;  
    }  
  
    public void enqueue(int item) {  
        if (isFull()) {  
            System.out.println("Queue Overflow! Cannot add more items.");  
        } else {
```

```

    if (isEmpty()) {
front = 0;
        }
        rear = (rear + 1) % capacity;
queue[rear] = item
size++;
        System.out.println(item + " added to the circular queue.");
    }
}

```

```

public int dequeue() {
if (isEmpty()) {
        System.out.println("Queue Underflow! No items to remove.");
        return -1;

    } else {

        int removedItem = queue[front];
front = (front + 1) % capacity;
        size--;
        return removedItem;
    }
}

```

```

public void display() {
if (isEmpty()) {
        System.out.println("Circular queue is empty.");

    } else {

        System.out.print("Circular queue elements: ");

for (int i = 0; i < size; i++) {
            System.out.print(queue[(front + i) % capacity] + " ");
        }
        System.out.println();
    }
}

```

```

    }

    public static void main(String[] args) {
System.out.println("Pranali B36");
    CircularQueue circularQueue = new CircularQueue(5);

        circularQueue.enqueue(10);
circularQueue.enqueue(20);
circularQueue.enqueue(30);
circularQueue.enqueue(40);
circularQueue.enqueue(50);
circularQueue.enqueue(60); // This will show Queue Overflow

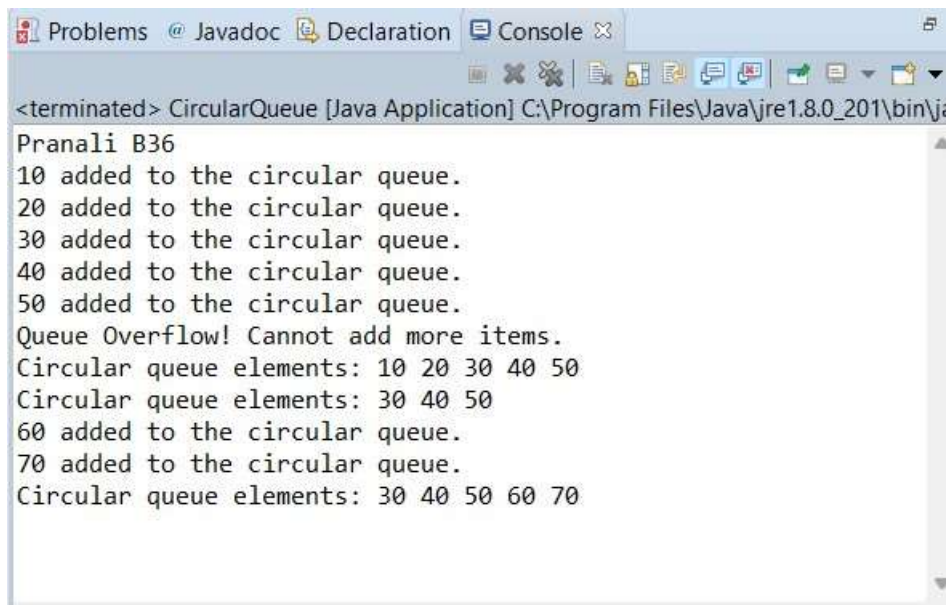
        circularQueue.display();

        circularQueue.dequeue();
circularQueue.dequeue();
circularQueue.display();

        circularQueue.enqueue(60);
circularQueue.enqueue(70);
circularQueue.display();
    }
}

```

OUTPUT :



```
<terminated> CircularQueue [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\java.exe
Pranali B36
10 added to the circular queue.
20 added to the circular queue.
30 added to the circular queue.
40 added to the circular queue.
50 added to the circular queue.
Queue Overflow! Cannot add more items.
Circular queue elements: 10 20 30 40 50
Circular queue elements: 30 40 50
60 added to the circular queue.
70 added to the circular queue.
Circular queue elements: 30 40 50 60 70
```

ii) ORDINARY QUEUE :

```
public class OrdinaryQueue {
    private int[] queue;
        private int front;
    private int rear;
    private int capacity;

    public OrdinaryQueue(int capacity) {
        this.capacity = capacity;
        queue = new int[capacity];
        front = 0;
        rear = -1;
    }

    public boolean isFull() {
```

```
return rear == capacity - 1;
}
```

```
public boolean isEmpty() {
    return front > rear;
}
```

```
public void enqueue(int item) {
    if (isFull()) {
        System.out.println("Queue Overflow! Cannot add more items.");
    } else {
        queue[++rear] = item;
        System.out.println(item + " added to the queue.");
    }
}
```

```
public int dequeue() {    if
(isEmpty()) {
    System.out.println("Queue Underflow! No items to remove.");
    return -1;
    } else {
return queue[front++];
    }
}
```

```
public void display() {
    if (isEmpty()) {
        System.out.println("Queue is empty.");
    } else {
        System.out.print("Queue elements: ");
        for (int i = front; i <= rear; i++) {
            System.out.print(queue[i] + " ");
        }
    }
}
```

```

        System.out.println();
    }
}

public static void main(String[] args) {
    System.out.println("Pranali B36");
    OrdinaryQueue queue = new OrdinaryQueue(5);

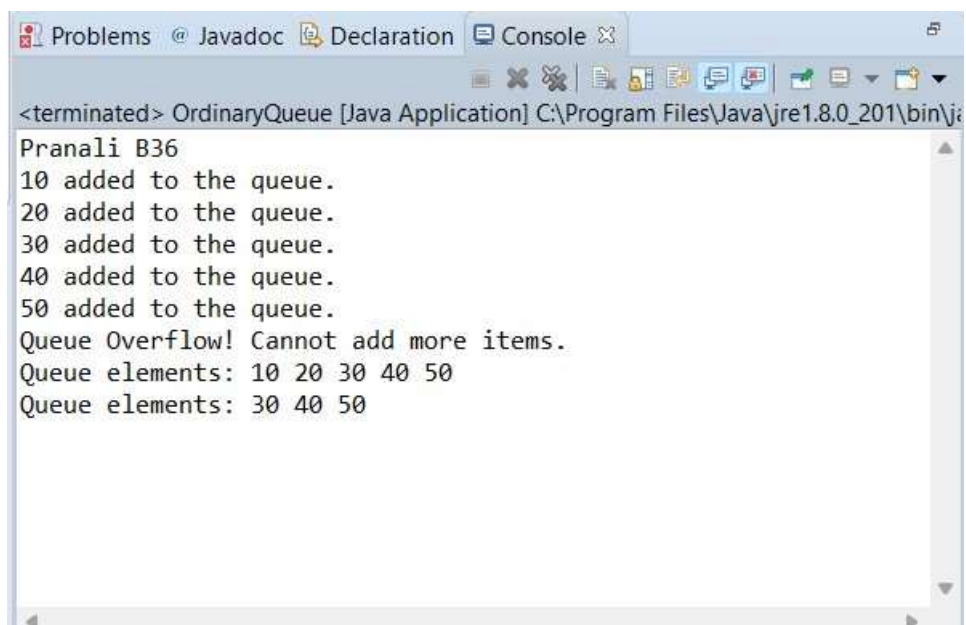
    queue.enqueue(10);
    queue.enqueue(20);
    queue.enqueue(30);
    queue.enqueue(40
    queue.enqueue(50);    queue.enqueue(60);

    queue.display();

    queue.dequeue();
    queue.dequeue();
    queue.display();
}
}

```

OUTPUT :



```

<terminated> OrdinaryQueue [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\ja
Pranali B36
10 added to the queue.
20 added to the queue.
30 added to the queue.
40 added to the queue.
50 added to the queue.
Queue Overflow! Cannot add more items.
Queue elements: 10 20 30 40 50
Queue elements: 30 40 50

```

3) Write a program for conversion of infix notation to postfix notation.

```
import java.util.Stack;
public class infixtopostfix {

    private static boolean isOperator(char ch) {
        return ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^'; }

    private static int precedence(char operator) {

        switch (operator) {
            case '+': case '-': return
```

```

1; case '*': case '/':
return 2; case '^':
return 3; default:
return -1;
}
}

```

```

public static String convertToPostfix(String infix) { Stack<Character> stack = new
Stack<>(); StringBuilder postfix = new StringBuilder();
for (int i = 0; i < infix.length(); i++) {
char ch = infix.charAt(i);

```

```

if (Character.isLetterOrDigit(ch)) { postfix.append(ch);
}

```

```

else if (ch == '(') { stack.push(ch);
}

```

```

else if (ch == ')') {

```

```

while (!stack.isEmpty() && stack.peek() != '(') {

```

```

postfix.append(stack.pop());
}

```

```

if (!stack.isEmpty() && stack.peek() == '(') { stack.pop();
}
}

```

```

else if (isOperator(ch)) { while (!stack.isEmpty() &&
precedence(stack.peek()) >= precedence(ch)) {
postfix.append(stack.pop());
}
}

```

```

stack.push(ch);
}
}

```

```

while (!stack.isEmpty()) {

```

```

postfix.append(stack.pop());
}
return postfix.toString();
}

```

```

public static void main(String[] args) {

```

```
System.out.println("Pranali B36");
```

```
String infixExpression = "a+b*(c^d-e)^(f+g*h)";
```

```
System.out.println("Infix
```

```
Expression: " + infixExpression);
```

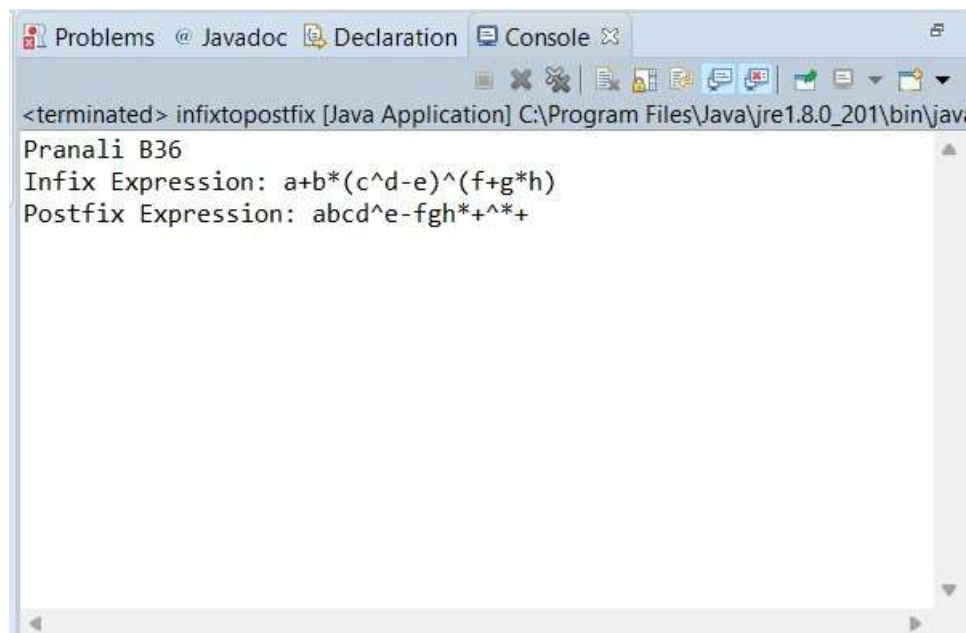
```
String postfixExpression = convertToPostfix(infixExpression);
```

```
System.out.println("Postfix Expression: " + postfixExpression);
```

```
}
```

```
}
```

OUTPUT :



4) Write a program for evaluation of postfix expression

```
import java.util.Stack;
public class postfixexpression {

    public static int evaluatePostfix(String postfix) {
        Stack<Integer> stack = new Stack<>();
        for (int i = 0; i < postfix.length(); i++) {
            char ch = postfix.charAt(i);

            if (Character.isDigit(ch)) {
                stack.push(ch - '0');
            }
            else {
                int operand2 =
                stack.pop(); int operand1 =
                stack.pop(); switch (ch) {case
                '+':
                stack.push(operand1 + operand2);
                break;

                case '-':
                stack.push(operand1 - operand2); break;
                case '*': stack.push(operand1 * operand2);
                break; case '/':
                if (operand2 == 0) {
                    throw new ArithmeticException("Division by zero");
                }
                stack.push(operand1 / operand2); break; case '^':
                stack.push((int) Math.pow(operand1, operand2));
                break; default:
                throw new IllegalArgumentException("Invalid operator: " + ch);
                }
                }
                }

            return stack.pop();
        }

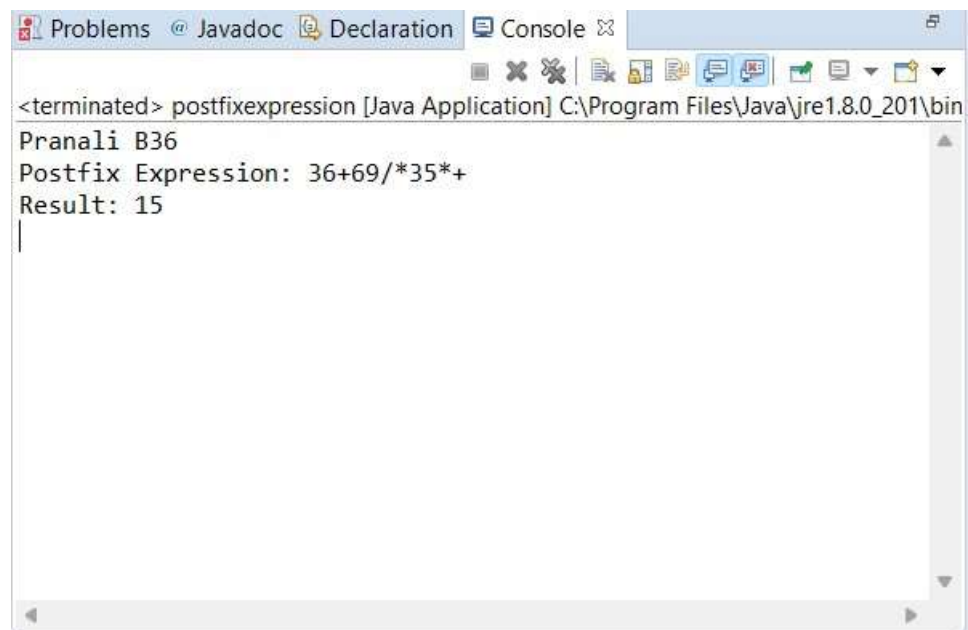
        public static void main(String[] args) {
            System.out.println("Pranali B36");
        }
    }
}
```

```
String postfixExpression = "36+69/*35*+";
```

```
System.out.println("Postfix Expression: " +postfixExpression);
```

```
try {  
int result = evaluatePostfix(postfixExpression);  
    System.out.println("Result: " + result);  
} catch (Exception e) {  
System.out.println("Error: " + e.getMessage());  
}  
}  
}
```

OUTPUT :



5) Write a program for balancing the parenthesis

```
import java.util.Stack;
public class Parentheses {

    public static boolean isBalanced(String expression) {
        Stack<Character> stack = new Stack<>();
        for (char ch : expression.toCharArray()) {

            if (ch == '(' || ch == '{' || ch == '[') {
                stack.push(ch);
            }

            else if (ch == ')' || ch == '}' || ch == ']') {

                if (stack.isEmpty() || !isMatchingPair(stack.pop(), ch)) { return
                false;
                }
                }

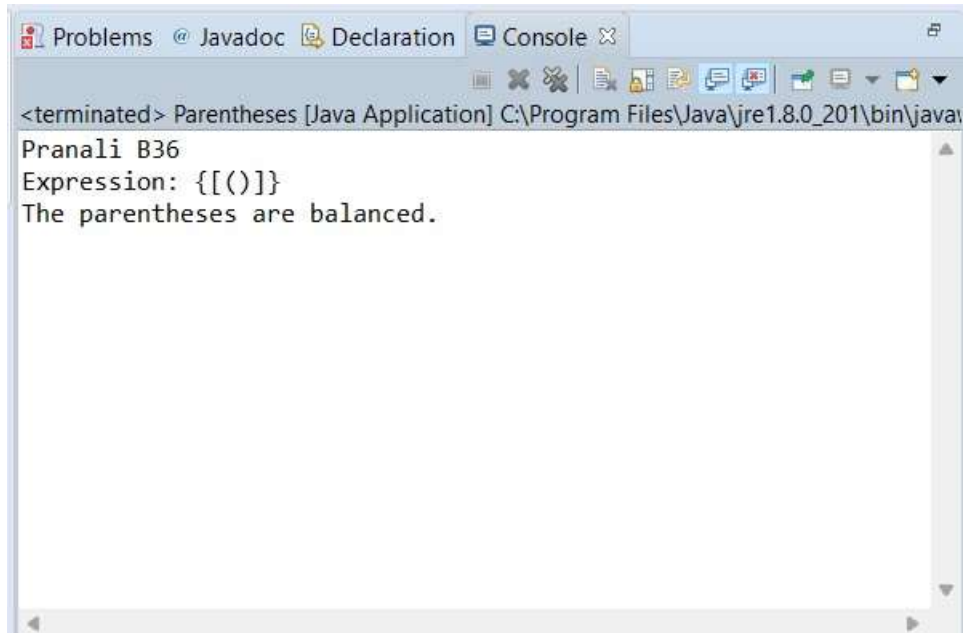
            }

            return stack.isEmpty();
        }

        private static boolean isMatchingPair(char open, char close) {
            return (open == '(' && close == ')') ||
                (open == '{' && close == '}') ||
                (open == '[' && close == ']');
        }

        public static void main(String[] args) {
            System.out.println("Pranali B36");
            String expression = "{[()]}" ;
            System.out.println("Expression: " + expression); if
            (isBalanced(expression)) {
                System.out.println("The parentheses are balanced.");
            } else {
                System.out.println("The parentheses are not balanced.");
            }
        }
    }
}
```

OUTPUT :



The screenshot shows an IDE's console window with the following content:

```
<terminated> Parentheses [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\java
Pranali B36
Expression: {[()]}
The parentheses are balanced.
```

The window has tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The 'Console' tab is active, displaying the output of a Java application named 'Parentheses'. The output includes the user's name 'Pranali B36', the input expression '[{()]}' (note: the image shows '[{()]}' but the text description says '{[()]}'), and the result 'The parentheses are balanced.'.

PRACTICAL NO 4

4.1 singly linked list (Insert, display, delete, search, count, reverse)

```
import java.util.Scanner;
// Node Class
class Node { int data; Node
next; public Node(int data) {
    this.data = data; this.next =
    null; }
}
// Singly Linked List Class
class SinglyLinkedList {
    private Node head;
    // Insert a new node at the end
    public void insert(int data) {
        Node newNode = new Node(data); if (head ==
        null) {
            head = newNode; }
        else {
            Node temp = head; while
            (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
        }
    }
    // Display all nodes
    public void display() {
        if (head == null) {
            System.out.println("List is empty."); return;
        }
        Node temp = head;
        System.out.print("Linked List: "); while (temp
        != null) {
            System.out.print(temp.data + " -> "); temp =
            temp.next;
        }
        System.out.println("null");
    }
    // Delete a node by value
```

```

public void delete(int value) {
    if (head == null) {
        System.out.println("List is empty. Nothing to delete."); return;
    }
    if (head.data == value) {
        head = head.next; // Remove head
        System.out.println("Deleted " + value); return;
    }
    Node temp = head; while (temp.next != null && temp.next.data !=
value) {
        temp = temp.next;
    }
    if (temp.next == null) {
        System.out.println(value + " not found in the list.");
    } else { temp.next = temp.next.next; // Skip the node to delete it
        System.out.println("Deleted " + value);
    }
}

// Search for a node by value
public boolean search(int value) { Node
temp = head; while (temp != null) {
    if (temp.data == value) {
        return true;
    }
    temp = temp.next;
}
return false;
}

// Count the number of nodes
public int count() { int count = 0;
Node temp = head; while (temp !=
null) { count++; temp = temp.next;
    }
    return count;
}

// Reverse the linked list
public void reverse() {
    Node prev = null;

```

```

Node current = head; Node
next; while (current != null)
{
    next = current.next; // Store next node
    current.next = prev; // Reverse current node's pointer
    prev = current;      // Move prev to current
    current = next;      // Move current to next
}
head = prev; // Update head to the new first node
System.out.println("List reversed.");
}
// Main Class public static void main(String[] args)
{
    Scanner scanner = new Scanner(System.in);
    SinglyLinkedList list = new SinglyLinkedList();
    while (true) {
        System.out.println("\n--- Menu ---");
        System.out.println("1. Insert");
        System.out.println("2. Display");
        System.out.println("3. Delete");
        System.out.println("4. Search");
        System.out.println("5. Count");
        System.out.println("6. Reverse");
        System.out.println("7. Exit");
        System.out.print("Enter your choice: "); int choice
        = scanner.nextInt(); switch (choice) {
            case 1: // Insert
                System.out.print("Enter the value to insert: ");
                int value = scanner.nextInt(); list.insert(value);
                break;
            case 2: // Display
                list.display();
                break;
            case 3: // Delete
                System.out.print("Enter the value to delete: ");
                int deleteValue = scanner.nextInt();
                list.delete(deleteValue);
                break;
            case 4: // Search

```

```

        System.out.print("Enter the value to search: "); int
        searchValue = scanner.nextInt(); if
        (list.search(searchValue)) {
            System.out.println(searchValue + " is present in the list.");
        } else {
            System.out.println(searchValue + " is not present in the
list."); }

        break;
    case 5: // Count
        System.out.println("Number of nodes in the list: " +
list.count()); break;
    case 6: // Reverse
        list.reverse(); break;
    case 7: // Exit
        System.out.println("Exiting...");
        scanner.close();
        System.exit(0); default:
        System.out.println("Invalid choice! Please try again.");
    }
}
}
}

```

OUTPUT :-

--- Menu ---

1. Insert
2. Display
3. Delete
4. Search
5. Count
6. Reverse
7. Exit

Enter your choice: 1

Enter the value to insert: 4

--- Menu ---

1. Insert
2. Display
3. Delete
4. Search
5. Count
6. Reverse
7. Exit

Enter your choice: 1

Enter the value to insert: 5

```
--- Menu ---
1. Insert
2. Display
3. Delete
4. Search
5. Count
6. Reverse
7. Exit
Enter your choice: 2
Linked List: 4 -> 5 -> null
```

```
--- Menu ---
1. Insert
2. Display
3. Delete
4. Search
5. Count
6. Reverse
7. Exit
Enter your choice: 3
Enter the value to delete: 5
Deleted 5
```

```
--- Menu ---
1. Insert
2. Display
3. Delete
4. Search
5. Count
6. Reverse
7. Exit
Enter your choice: 7
Exiting...
```


4.2 doubly linked list (Insert, display, delete, search, count, reverse)

```
import java.util.Scanner;
// Node class for Doubly Linked List
class DoublyNode {
    int data;
    DoublyNode prev; DoublyNode next;
    public DoublyNode(int data) {
        this.data = data; this.prev =
        null; this.next = null;
    }
}
// Doubly Linked List class
class DoublyLinkedList { private
DoublyNode head; private DoublyNode
tail; // Insert a new node at the end
public void insert(int data) {
    DoublyNode newNode = new DoublyNode(data); if (head ==
    null) { head = newNode; tail = newNode;
    } else { tail.next = newNode;
        newNode.prev = tail; tail
        = newNode;
    }
}
// Display all nodes
public void display() {
    if (head == null) {
        System.out.println("List is empty."); return;
    }
    DoublyNode temp = head;
    System.out.print("Doubly Linked List: "); while
    (temp != null) {
        System.out.print(temp.data + " <-> "); temp =
        temp.next;
    }
    System.out.println("null");
}
// Delete a node by value
public void delete(int value) {
    if (head == null) {
```

```

        System.out.println("List is empty. Nothing to delete."); return;
    }
    if (head.data == value) { // Delete the head node
        head = head.next; if
        (head != null) {
            head.prev = null;
        } else { tail = null; // List becomes empty
        }
        System.out.println("Deleted " + value); return;
    }
    DoublyNode temp = head; while (temp != null &&
    temp.data != value) {
        temp = temp.next;
    }
    if (temp == null) { // Value not found
        System.out.println(value + " not found in the list.");
    } else { if (temp.next != null) { // Deleting a middle node
        temp.next.prev = temp.prev;
    } else { // Deleting the tail node
        tail = temp.prev;
    }
        temp.prev.next = temp.next;
        System.out.println("Deleted " + value);
    }
}

// Search for a node by value
public boolean search(int value) {
    DoublyNode temp = head;
    while (temp != null) {
        if (temp.data == value) {
            return true;
        }
        temp = temp.next;
    }
    return false;
}

// Count the number of nodes
public int count() {

```

```

    int count = 0; DoublyNode temp
    = head; while (temp != null) {
        count++; temp = temp.next;
    }
    return count;
}

// Reverse the doubly linked list
public void reverse() {
    if (head == null) {
        System.out.println("List is empty. Cannot reverse."); return;
    }
    DoublyNode current = head;
    DoublyNode temp = null; while
    (current != null) {
        temp = current.prev; current.prev
        = current.next; current.next =
        temp; current = current.prev;
    }
    if (temp != null) {
        head = temp.prev; // Update the head to the last node
    }
    System.out.println("List reversed.");
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in); DoublyLinkedList list
    = new DoublyLinkedList(); while (true) {
        System.out.println("\n--- Menu ---");
        System.out.println("1. Insert");
        System.out.println("2. Display");
        System.out.println("3. Delete");
        System.out.println("4. Search");
        System.out.println("5. Count");
        System.out.println("6. Reverse");
        System.out.println("7. Exit");
        System.out.print("Enter your choice: "); int choice
        = scanner.nextInt(); switch (choice) {
            case 1: // Insert
                System.out.print("Enter the value to insert: "); int value =
                scanner.nextInt(); list.insert(value); break;

```

```

    case 2: // Display
        list.display(); break;
    case 3: // Delete
        System.out.print("Enter the value to delete: "); int
        deleteValue = scanner.nextInt(); list.delete(deleteValue);
        break;
    case 4: // Search
        System.out.print("Enter the value to search: "); int
        searchValue = scanner.nextInt(); if
        (list.search(searchValue)) {
            System.out.println(searchValue + " is present in the list.");
        } else {
            System.out.println(searchValue + " is not present in the
list.");
        }
        break;
    case 5: // Count
        System.out.println("Number of nodes in the list: " +
list.count()); break;
    case 6: // Reverse
        list.reverse(); break;
    case 7: // Exit
        System.out.println("Exiting...");
        scanner.close();
        System.exit(0); default:
        System.out.println("Invalid choice! Please try again.");
    }
}
}
}

```

OUTPUT :-

--- Menu ---

1. Insert
2. Display
3. Delete
4. Search
5. Count
6. Reverse
7. Exit

Enter your choice: 1

Enter the value to insert: 3

--- Menu ---

1. Insert
2. Display
3. Delete
4. Search
5. Count
6. Reverse
7. Exit

Enter your choice: 1

Enter the value to insert: 8

--- Menu ---

1. Insert
2. Display
3. Delete
4. Search
5. Count
6. Reverse
7. Exit

Enter your choice: 2

Doubly Linked List: 3 <-> 8 <-> null

--- Menu ---

1. Insert
2. Display
3. Delete
4. Search
5. Count
6. Reverse
7. Exit

Enter your choice: 5

Number of nodes in the list: 2

--- Menu ---|

1. Insert
2. Display
3. Delete
4. Search
5. Count
6. Reverse
7. Exit

Enter your choice: 7

Exiting...

4.3 circular linked list (Insert, display, delete, search,count, reverse)

```
import java.util.Scanner;
// Node class for Circular Linked List
class CircularNode {
    int data;
    CircularNode next; public
    CircularNode(int data) {
        this.data = data; this.next =
        null;
    }
}
// Circular Linked List class
class CircularLinkedList {
    private CircularNode last;
    // Insert a new node at the end
    public void insert(int data) {
        CircularNode newNode = new CircularNode(data); if (last ==
        null) {
            last = newNode;
            last.next = last; // Circular link
        } else { newNode.next = last.next;
            last.next = newNode; last =
            newNode;
        }
    }
    // Display all nodes
    public void display() {
        if (last == null) {
            System.out.println("List is empty."); return;
        }
        CircularNode temp = last.next;
        System.out.print("Circular Linked List: "); do {
            System.out.print(temp.data + " -> "); temp =
            temp.next;
        } while (temp != last.next);
        System.out.println("(back to start)");
    }
    // Delete a node by value
```

```

public void delete(int value) { if (last
== null) {
    System.out.println("List is empty. Nothing to delete."); return;
}
CircularNode current = last.next, prev = last;
// Deleting the head node
if (current.data == value) {
    if (current == last) {
        // Only one node
        last = null;
    } else { last.next = current.next;
    }
    System.out.println("Deleted " + value); return;
}
// Traverse to find the node to delete
while (current != last && current.data != value) {
    prev = current; current =
    current.next;
}
if (current.data == value) {
    prev.next = current.next; if (current == last) {
        // Deleting the last node
        last = prev;
    }
    System.out.println("Deleted " + value);
} else {
    System.out.println(value + " not found in the list.");
}
}

// Search for a node by value
public boolean search(int value) {
    if (last == null) {
        return false;
    }
    CircularNode temp = last.next;
    do {
        if (temp.data == value) {
            return true;

```



```

    }
    temp = temp.next; } while
    (temp != last.next); return false;
}
// Count the number of nodes
public int count() {
    if (last == null) {
        return 0;
    }
    int count = 0;
    CircularNode temp = last.next; do {
        count++; temp = temp.next; }
    while (temp != last.next); return
    count;
}
// Reverse the circular linked list
public void reverse() {
    if (last == null || last.next == last) {
        System.out.println("List reversed."); return;
    }
    CircularNode prev = last, current = last.next, next = null; do {
        next = current.next; current.next
    = prev; prev = current; current =
    next; } while (current != last.next);
    last.next = prev;
    last = current; // Update last to the new last node
    System.out.println("List reversed.");
}
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    CircularLinkedList list = new CircularLinkedList(); while (true) {
        System.out.println("\n--- Menu ---");
        System.out.println("1. Insert");
        System.out.println("2. Display");
        System.out.println("3. Delete");
        System.out.println("4. Search");
        System.out.println("5. Count");
        System.out.println("6. Reverse");
        System.out.println("7. Exit");
    }
}

```

```

System.out.print("Enter your choice: "); int
choice = scanner.nextInt(); switch (choice) {
    case 1: // Insert
        System.out.print("Enter the value to insert: "); int value =
            scanner.nextInt(); list.insert(value); break;
    case 2: // Display
        list.display(); break;
    case 3: // Delete
        System.out.print("Enter the value to delete: "); int
        deleteValue = scanner.nextInt(); list.delete(deleteValue);
        break;
    case 4: // Search
        System.out.print("Enter the value to search: "); int
        searchValue = scanner.nextInt(); if
        (list.search(searchValue)) {
            System.out.println(searchValue + " is present in the list.");
        } else {
            System.out.println(searchValue + " is not present in the
list.");
        }
        break;
    case 5: // Count
        System.out.println("Number of nodes in the list: " +
list.count()); break;
    case 6: // Reverse
        list.reverse(); break;
    case 7: // Exit
        System.out.println("Exiting...");
        scanner.close();
        System.exit(0); default:
        System.out.println("Invalid choice! Please try again.");
    }
}
}
}

```

OUTPUT :-

--- Menu ---

1. Insert
2. Display
3. Delete
4. Search
5. Count
6. Reverse
7. Exit

Enter your choice: 1

Enter the value to insert: 7

--- Menu ---

1. Insert
2. Display
3. Delete
4. Search
5. Count
6. Reverse
7. Exit

Enter your choice: 1

Enter the value to insert: 2

--- Menu ---

1. Insert
2. Display
3. Delete
4. Search
5. Count
6. Reverse
7. Exit

Enter your choice: 2

Circular Linked List: 7 -> 2 -> (back to start)

--- Menu ---

1. Insert
2. Display
3. Delete
4. Search
5. Count
6. Reverse
7. Exit

Enter your choice: 6

List reversed.

--- Menu ---

1. Insert
2. Display
3. Delete
4. Search
5. Count
6. Reverse
7. Exit

Enter your choice: 7

Exiting...

4.4 polynomial addition using linked list

```
import java.util.Scanner;
// Node class for polynomial representation
class PolyNode {
    int coefficient;
    int exponent; PolyNode next
    public PolyNode(int coefficient, int exponent) {
        this.coefficient = coefficient;
        this.exponent = exponent;
        this.next = null;
    }
}
// Polynomial class
class Polynomial {
    private PolyNode head;
    // Insert a term into the polynomial (in sorted order by exponent)
    public void insertTerm(int coefficient, int exponent) {
        PolyNode newNode = new PolyNode(coefficient, exponent);
        if (head == null || head.exponent < exponent) {
            // Insert at the head
            newNode.next = head; head =
            newNode;
        } else {
            // Insert in proper order
            PolyNode current = head;
            while (current.next != null && current.next.exponent > exponent)
            {
                current = current.next;
            }
            if (current.next != null && current.next.exponent == exponent) { // If the exponent
                already exists, add the coefficients
                current.next.coefficient += coefficient;
            } else {
                // Insert the new node
                newNode.next = current.next; current.next =
                newNode;
            }
        }
    }
}
```

```

    }
    // Display the polynomial
    public void display() {
        if (head == null) {
            System.out.println("Polynomial is empty.");
            return;
        }
        PolyNode current = head;
        StringBuilder result = new StringBuilder(); while
        (current != null) {

            result.append(current.coefficient).append("x^").append(current.exponent
            );

            if (current.next != null) { result.append(" + ");
                }
            current = current.next;
        }
        System.out.println(result);
    }
    // Add two polynomials
    public static Polynomial addPolynomials(Polynomial poly1,
    Polynomial poly2) {
        Polynomial result = new Polynomial();
        PolyNode p1 = poly1.head; PolyNode p2
        = poly2.head; while (p1 != null && p2 !=
        null) {
            if (p1.exponent == p2.exponent) { result.insertTerm(p1.coefficient +
                p2.coefficient, p1.exponent); p1 = p1.next; p2 = p2.next;
            } else if (p1.exponent > p2.exponent) {
                result.insertTerm(p1.coefficient, p1.exponent); p1 =
                p1.next;
            } else { result.insertTerm(p2.coefficient, p2.exponent); p2 =
                p2.next;
            }
        }
        // Add remaining terms from the first polynomial
        while (p1 != null) { result.insertTerm(p1.coefficient,
            p1.exponent); p1 = p1.next;
        }
        // Add remaining terms from the second polynomial
    }

```

```

        while (p2 != null) {
            result.insertTerm(p2.coefficient, p2.exponent); p2 = p2.next;
        }
        return result;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Input first polynomial
        System.out.println("Enter the first polynomial:");
        Polynomial poly1 = new Polynomial();
        System.out.print("Enter the number of terms: "); int n1 =
        scanner.nextInt(); for (int i = 0; i < n1; i++) {
            System.out.print("Enter coefficient and exponent: "); int
            coefficient = scanner.nextInt(); int exponent = scanner.nextInt();
            poly1.insertTerm(coefficient, exponent);
        }
        // Input second polynomial
        System.out.println("\nEnter the second polynomial:");
        Polynomial poly2 = new Polynomial();
        System.out.print("Enter the number of terms: "); int n2 =
        scanner.nextInt(); for (int i = 0; i < n2; i++) {
            System.out.print("Enter coefficient and exponent: "); int
            coefficient = scanner.nextInt(); int exponent = scanner.nextInt();
            poly2.insertTerm(coefficient, exponent);
        }
        // Display the input polynomials
        System.out.println("\nFirst Polynomial:");
        poly1.display(); System.out.println("Second
        Polynomial:"); poly2.display();
        // Perform polynomial addition
        Polynomial result = Polynomial.addPolynomials(poly1, poly2);
        // Display the result
        System.out.println("\nResultant Polynomial (Sum:");
        result.display(); scanner.close();
    }
}

```

OUTPUT :-

Enter the first polynomial:
Enter the number of terms: 3
Enter coefficient and exponent: 5 3
Enter coefficient and exponent: 4 2
Enter coefficient and exponent: 4 0

Enter the second polynomial:
Enter the number of terms: 3
Enter coefficient and exponent: 2 3
Enter coefficient and exponent: 3 2
Enter coefficient and exponent: 7 2

First Polynomial:
 $5x^3 + 4x^2 + 4x^0$
Second Polynomial:
 $2x^3 + 10x^2$

Resultant Polynomial (Sum):
 $7x^3 + 14x^2 + 4x^0$

4.5 Linked List implementation of stack, ordinary queue

```
import java.util.Scanner;
// Queue implementation using linked list
class LinkedListQueue {
    private Node front; private Node
    rear;
    // Enqueue an element to the queue
    public void enqueue(int data) { Node newNode
    = new Node(data); if (rear == null) {
        front = rear = newNode;
    } else { rear.next = newNode;
        rear = newNode;
    }
    System.out.println(data + " enqueued to the queue.");
}
// Dequeue an element from the queue
public void dequeue() {
    if (front == null) {
        System.out.println("Queue is empty. Cannot dequeue.");
    } else {
        System.out.println(front.data + " dequeued from the queue."); front =
        front.next; if (front == null) {
            rear = null;
        }
    }
}
// Display all elements in the queue
public void display() {
    if (front == null) {
        System.out.println("Queue is empty.");
    } else {
        Node temp = front;
        System.out.print("Queue: "); while (temp
        != null) { System.out.print(temp.data + " -
        > "); temp = temp.next;
        }
        System.out.println("null");
    }
}
```

```

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in); LinkedListQueue queue =
    new LinkedListQueue(); while (true) {
        System.out.println("\n--- Queue Menu ---");
        System.out.println("1. Enqueue");
        System.out.println("2. Dequeue");
        System.out.println("3. Display");
        System.out.println("4. Exit");
        System.out.print("Enter your choice: "); int
        choice = scanner.nextInt(); switch (choice) {
            case 1:
                System.out.print("Enter the value to enqueue: "); int value =
                scanner.nextInt(); queue.enqueue(value); break;
            case 2:
                queue.dequeue(); break;
            case 3:
                queue.display(); break;
            case 4:
                System.out.println("Exiting...");
                scanner.close();
                System.exit(0); default:
                System.out.println("Invalid choice! Please try again.");
            }
        }
    }
}

```

OUTPUT :-

--- Queue Menu ---

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 1

Enter the value to enqueue: 10

10 enqueued to the queue.

--- Queue Menu ---

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 1

Enter the value to enqueue: 20

20 enqueued to the queue.

--- Queue Menu ---

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice:

--- Queue Menu ---

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 1

Enter the value to enqueue: 10

10 enqueued to the queue.

--- Queue Menu ---

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 1

Enter the value to enqueue: 20

20 enqueued to the queue.

--- Queue Menu ---

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice:

--- Queue Menu ---

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 4

Exiting...

4.6 linked list implementation of priority queue, deque

```
import java.util.Scanner; // Node
class for the linked list
class PriorityNode {
    int data; int
    priority;
    PriorityNode next;
    public PriorityNode(int data, int priority) {
        this.data = data; this.priority =
        priority; this.next = null;
    }
}
// Priority Queue using Linked List
class LinkedListPriorityQueue {
    private PriorityNode head;
    // Insert an element based on priority
    public void enqueue(int data, int priority) {
        PriorityNode newNode = new PriorityNode(data, priority);
        if (head == null || head.priority < priority) {
            newNode.next = head; head =
            newNode;
        } else {
            PriorityNode temp = head;
            while (temp.next != null && temp.next.priority >= priority) {
                temp = temp.next;
            }
            newNode.next = temp.next;
            temp.next = newNode;
        }
        System.out.println(data + " with priority " + priority + " enqueued.");
    }
    // Dequeue the highest priority element
    public void dequeue() {
        if (head == null) {
            System.out.println("Priority Queue is empty.");
        } else {
            System.out.println(head.data + " with priority " + head.priority + " dequeued.");
            head = head.next;
        }
    }
    // Display the priority queue
    public void display() {
```

```

    if (head == null) {
        System.out.println("Priority Queue is empty.");
    } else {
        PriorityNode temp = head;
        System.out.print("Priority Queue: "); while (temp
            != null) {
                System.out.print("(" + temp.data + ", " + temp.priority + ") -> "); temp =
                temp.next;
            }
        System.out.println("null");
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    LinkedListPriorityQueue pq = new LinkedListPriorityQueue(); while (true) {
        System.out.println("\n--- Priority Queue Menu ---");
        System.out.println("1. Enqueue");
        System.out.println("2. Dequeue");
        System.out.println("3. Display");
        System.out.println("4. Exit");
        System.out.print("Enter your choice: "); int
        choice = scanner.nextInt(); switch (choice) {
            case 1:
                System.out.print("Enter the value to enqueue: "); int value =
                scanner.nextInt(); System.out.print("Enter the priority: "); int
                priority = scanner.nextInt(); pq.enqueue(value, priority);
                break;
            case 2:
                pq.dequeue(); break;
            case 3:
                pq.display(); break;
            case 4:
                System.out.println("Exiting...");
                scanner.close();
                System.exit(0); default:
                System.out.println("Invalid choice! Please try again.");
            }
        }
    }
}

```

OUTPUT :-

```
--- Priority Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 20
Enter the priority: 1
20 with priority 1 enqueued.
```

```
--- Priority Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Priority Queue: (20, 1) -> null
```

```
--- Priority Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting...
```

PRACTICAL NO : 5

1) Write a program to insert a node in Binary Search Tree.

Codepackage tree;

//Node class representing a single node in the Binary Search Tree

class Node {

int key;

Node left, right;

public Node(int item) {

key = item;

left = right = null;

}

}

//Binary Search Tree class

class BinarySearchTree {

Node root;

// Constructor to initialize the BST

public BinarySearchTree() {

root = null;

}

// Method to insert a new node

void insert(int key) {

root = insertRec(root, key);

}

// A recursive function to insert a new key in the BST

Node insertRec(Node root, int key) {

// If the tree is empty, return a new node

if (root == null) {


```

        root = new Node(key);
        return root;
    }

    // Otherwise, recursively traverse down the tree
    if (key < root.key)
        root.left = insertRec(root.left, key);
    else if (key > root.key)
        root.right = insertRec(root.right, key);

    // Return the (unchanged) node pointer
    return root;
}

// Method to perform an inorder traversal
void inorder() {
    inorderRec(root);
}

// A utility function to perform inorder traversal of BST
void inorderRec(Node root) {
    if (root != null) {
        inorderRec(root.left);
        System.out.print(root.key + " ");
        inorderRec(root.right);
    }
}

// Main method to test the BST insertion
public static void main(String[] args) {
    BinarySearchTree bst = new BinarySearchTree();

```

```

/* Example usage:

* Insert the following nodes: 50, 30, 20, 40, 70, 60, 80

*/

bst.insert(50);
bst.insert(30);
bst.insert(20);
bst.insert(40);
bst.insert(70);
bst.insert(60);
bst.insert(80);

// Print inorder traversal of the BST
System.out.println("Inorder traversal of the BST:");
bst.inorder();
}
}

```

Output:

```

Inorder traversal of the BST:
20 30 40 50 60 70 80

```

2) Write a program to traverse BST in perorder, postorder and inorder.

Code:

```

package tree;

//Node class representing a single node in the Binary Search Tree
class Node {
    int key;
    Node left, right;

    public Node(int item) {

```

```
    key = item;
    left = right = null;
}
}
```

//Binary Search Tree class

```
class BinarySearchTree {
```

```
Node root;
```

// Constructor to initialize the BST

```
public BinarySearchTree() {
```

```
    root = null;
```

```
}
```

// Method to insert a new node

```
void insert(int key) {
```

```
    root = insertRec(root, key);
```

```
}
```

// A recursive function to insert a new key in the BST

```
Node insertRec(Node root, int key) {
```

```
    // If the tree is empty, return a new node if
```

```
    (root == null) {
```

```
        root = new Node(key);
```

```
        return root;
```

```
    }
```

```
    // Otherwise, recursively traverse down the tree if
```

```
    (key < root.key)
```

```
        root.left = insertRec(root.left, key);
```

```
    else if (key > root.key)
```

```
        root.right = insertRec(root.right, key);

        // Return the (unchanged) node pointer
        return root;
    }
}
```

```
// Method for inorder traversal
```

```
void inorder() {
    System.out.print("Inorder traversal: ");
    inorderRec(root);
    System.out.println();
}
```

```
void inorderRec(Node root)
```

```
{ if (root != null) {
    inorderRec(root.left);
    System.out.print(root.key + " ");
    inorderRec(root.right);
}
}
```

```
// Method for preorder traversal
```

```
void preorder() {
    System.out.print("Preorder traversal: ");
    preorderRec(root);
    System.out.println();
}
```

```
void preorderRec(Node root) {
```

```
    if (root != null) {
        System.out.print(root.key + " ");
    }
}
```

```
        preorderRec(root.left);
        preorderRec(root.right);
    }
}
```

// Method for postorder traversal

```
void postorder() {
    System.out.print("Postorder traversal: ");
    postorderRec(root); System.out.println();
}
```

```
void postorderRec(Node root) {
    if (root != null) {
        postorderRec(root.left);
        postorderRec(root.right);
        System.out.print(root.key + " ");
    }
}
```

// Main method to test the traversals

```
public static void main(String[] args)
{
    BinarySearchTree bst = new BinarySearchTree();

    /* Example usage:
    * Insert the following nodes: 50, 30, 20, 40, 70, 60, 80
    */

    bst.insert(50);
    bst.insert(30);
    bst.insert(20);
    bst.insert(40);
```

```

bst.insert(70);
bst.insert(60);
bst.insert(80);

// Perform all three traversals
bst.inorder(); // Inorder traversal
bst.preorder(); // Preorder traversal
bst.postorder(); // Postorder traversal
}
}

```

Output:

```

<terminated> parenthesis [Java Application] C:\Users\Rapid Solutions\.p2\pool\plug
Inorder traversal: 20 30 40 50 60 70 80
Preorder traversal: 50 30 20 40 70 60 80
Postorder traversal: 20 40 30 60 80 70 50

```

3) Write a program to find the largest and smallest node in BST.

Code:

```

package tree;

// Node class representing a single node in the Binary Search Tree
class Node {
    int key;
    Node left, right;

    public Node(int item) {
        key = item;
        left = right = null;
    }
}

```

```
// Binary Search Tree class
class BinarySearchTree {
    Node root;

    // Constructor to initialize the BST
    public BinarySearchTree() {
        root = null;
    }

    // Method to insert a new node
    void insert(int key) {
        root = insertRec(root, key);
    }

    // A recursive function to insert a new key in the BST
    Node insertRec(Node root, int key) {
        // If the tree is empty, return a new node
        if (root == null) {
            root = new Node(key);
            return root;
        }

        // Otherwise, recursively traverse down the tree
        if (key < root.key)
            root.left = insertRec(root.left, key);
        else if (key > root.key)
            root.right = insertRec(root.right, key);

        // Return the (unchanged) node pointer
    }
}
```

```
    return root;
}
```

// Method to find the smallest node

```
int findMin() {
    return findMinRec(root).key;
}
```

```
Node findMinRec(Node root) {
    // The smallest node is the leftmost node
    if (root.left == null) {
        return root;
    }
    return findMinRec(root.left);
}
```

// Method to find the largest node

```
int findMax() {
    return findMaxRec(root).key;
}
```

```
Node findMaxRec(Node root) {
    // The largest node is the rightmost node
    if (root.right == null) {
        return root;
    }
    return findMaxRec(root.right);
}
```



```

// Main method to test the functionality
public static void main(String[] args) {
    BinarySearchTree bst = new BinarySearchTree();

    /* Example usage:
    * Insert the following nodes: 50, 30, 20, 40, 70, 60, 80
    */

    bst.insert(50);
    bst.insert(30);
    bst.insert(20);
    bst.insert(40);
    bst.insert(70);
    bst.insert(60);
    bst.insert(80);

    // Find the smallest and largest nodes
    System.out.println("Smallest node in BST: " + bst.findMin());
    System.out.println("Largest node in BST: " + bst.findMax());
}
}

```

Output:

```

<terminated> parenthesis [Java Application] C:\Users\Rapid Solutions\p2\pool\plug
Smallest node in BST: 20
Largest node in BST: 80

```

4) Write a program to insert and delete a node in Max-Heap.

Code:

```
package tree;
```

```
import java.util.ArrayList;

class MaxHeap {
    private ArrayList<Integer> heap;

    // Constructor to initialize the heap
    public MaxHeap() {
        heap = new ArrayList<>();
    }

    // Method to insert a node into the Max-Heap
    public void insert(int value) {
        // Add the new value at the end of the heap
        heap.add(value);
        // Move the new value up to maintain the heap property
        heapifyUp(heap.size() - 1);
    }

    // Helper method to restore the heap property from bottom to top
    private void heapifyUp(int index) {
        int parentIndex = (index - 1) / 2;

        // While we haven't reached the root and the parent is smaller than the
        // child
        while (index > 0 && heap.get(parentIndex) < heap.get(index)) {
            // Swap the parent and the child
            swap(parentIndex, index);

            // Move up the tree
        }
    }
}
```

```
        index = parentIndex;
        parentIndex = (index - 1) / 2;
    }
}
```

// Method to delete the root node (maximum value) from the Max-Heap

```
public int deleteMax() {
    if (heap.size() == 0) {
        throw new IllegalStateException("Heap is empty");
    }
}
```

// Replace the root with the last element

```
int maxVal = heap.get(0);
int lastVal = heap.remove(heap.size() - 1);
```

```
if (heap.size() > 0) {
    heap.set(0, lastVal);
    // Restore the heap property from top to bottom
    heapifyDown(0);
}
```

```
return maxVal;
}
```

// Helper method to restore the heap property from top to bottom

```
private void heapifyDown(int index) {
    int leftChildIndex = 2 * index + 1;
    int rightChildIndex = 2 * index + 2;
    int largestIndex = index;
```

```
        // Find the largest value among the current node and its children
        if (leftChildIndex < heap.size() && heap.get(leftChildIndex) >
heap.get(largestIndex)) {
            largestIndex = leftChildIndex;
        }
```

```
        if (rightChildIndex < heap.size() && heap.get(rightChildIndex) >
heap.get(largestIndex)) {
            largestIndex = rightChildIndex;
        }
```

```
        // If the largest value is not the current node, swap and continue
        if (largestIndex != index) {
            swap(index, largestIndex);
            heapifyDown(largestIndex);
        }
    }
```

```
// Method to swap two elements in the heap
private void swap(int i, int j) {
    int temp = heap.get(i);
    heap.set(i, heap.get(j));
    heap.set(j, temp);
}
```

```
// Method to print the heap
public void printHeap() {
    System.out.println("Heap: " + heap);
}
```

```

    }

    // Main method to test the Max-Heap operations
    public static void main(String[] args) {
        MaxHeap maxHeap = new MaxHeap();

        // Insert elements into the heap
        maxHeap.insert(50);
        maxHeap.insert(30);
        maxHeap.insert(20);
        maxHeap.insert(40);
        maxHeap.insert(70);
        maxHeap.insert(60);
        maxHeap.insert(80);

        // Print the heap
        System.out.println("After inserting elements:");
        maxHeap.printHeap();

        // Delete the maximum element
        System.out.println("Deleted max element: " + maxHeap.deleteMax());

        // Print the heap after deletion
        System.out.println("After deleting the max element:");
        maxHeap.printHeap();
    }
}

```

Output:

<terminated> parenthesis [Java Application] C:\Users\Rapid Solutions\p2\

After inserting elements:

Heap: [80, 50, 70, 30, 40, 20, 60]

Deleted max element: 80

After deleting the max element:

Heap: [70, 50, 60, 30, 40, 20]

PRACTICAL NO 6 :

1) write a program to represent graph using adjacency matrix and adjacency list Code:

```
import

java.util.ArrayList;

import java.util.List;

public class Main {

    // Method to add edge to an adjacency list

    public static void addEdgeList(List<List<Integer>> adj, int

        i, int j) { adj.get(i).add(j);

        adj.get(j).add(i);

    }

    // Method to display adjacency list representation

    public static void

    displayAdjList(List<List<Integer>> adj) {

        System.out.println("Adjacency List

        Representation:");for (int i = 0; i < adj.size(); i++)

        {

            System.out.print(i + ": ");

            for (int j : adj.get(i)) {

                System.out.print(j + " ");

            }

            System.out.println();

        }

    }

    // Method to add edge to an adjacency matrix

    public static void addEdgeMatrix(int[][] mat, int i,

        int j) { mat[i][j] = 1;

        mat[j][i] = 1;

    }

}
```

```
// Method to display adjacency matrix representation
```

```
public static void displayAdjMatrix(int[][] mat) {
```

```
    System.out.println("Adjacency Matrix
```

```
Representation:");for (int[] row : mat) {
```

```
    for (int val : row) {
```

```
        System.out.print(val + "
```

```
        ");
```

```
    }
```

```
    System.out.println();
```

```
}
```

```
}
```

```
public static void main(String[] args) {
```

```
    int vertices = 4; // Number of vertices in the graph
```

```
    // Create adjacency list representation
```

```
    List<List<Integer>> adjList = new
```

```
    ArrayList<>();for (int i = 0; i < vertices;
```

```
    i++) {
```

```
        adjList.add(new ArrayList<>());
```

```
    }
```

```
    // Create adjacency matrix representation
```

```
    int[][] adjMatrix = new
```

```
    int[vertices][vertices];
```

```
    // Add edges to the
```

```
    graph
```

```
    addEdgeList(adjList, 0,
```

```
    1);
```

```
    addEdgeList(adjList, 0, 3);
```

```
    addEdgeList(adjList, 1, 3);
```



```

addEdgeList(adjList, 2, 3);
addEdgeMatrix(adjMatrix, 0, 1);
addEdgeMatrix(adjMatrix, 0, 2);
addEdgeMatrix(adjMatrix, 1, 2);
addEdgeMatrix(adjMatrix, 2, 3);

// Create a loop to allow repeated user interaction
java.util.Scanner scanner = new
java.util.Scanner(System.in);int choice;
do {
    // Display options to user
    System.out.println("\nChoose the type of graph representation to display:");
    System.out.println("1 - Adjacency List");
    System.out.println("2 - Adjacency
Matrix");System.out.println("3 - Exit");
    choice = scanner.nextInt();

    switch
    (choice) {
        case 1:
            displayAdjList(adjL
            ist);break;
        case 2:
            displayAdjMatrix(adjMatrix
            );break;
        case 3:
            System.out.println("Exiting
            program.");break;
        default:
            System.out.println("Invalid choice. Please try again.");
    }
} while (choice != 3);

```

```
// Display the exit message with the name

scanner.close();

}

}
```

Output:

```
Choose the type of graph representation to display:
1 - Adjacency List
2 - Adjacency Matrix
3 - Exit
1
Adjacency List Representation:
0: 1 3
1: 0 3
2: 3
3: 0 1 2

Choose the type of graph representation to display:
1 - Adjacency List
2 - Adjacency Matrix
3 - Exit
2
Adjacency Matrix Representation:
0 1 1 0
1 0 1 0
1 1 0 1
0 0 1 0

Choose the type of graph representation to display:
1 - Adjacency List
2 - Adjacency Matrix
3 - Exit
3
Exiting program.
```

2) Write a program to traverse a graph using BFS and DFS

method.CODE:

```
import java.util.*;

class Main {

    // Breadth-First Search (BFS)
    static void bfs(List<List<Integer>> adj, int
        s) { Queue<Integer> q = new
        LinkedList<>(); boolean[] visited = new
        boolean[adj.size()];visited[s] = true;
        q.add(s);
        while
            (!q.isEmpty()) {
                int curr =
                q.poll();
                System.out.print(curr + "
                ");for (int x : adj.get(curr))
                {
                    if (!visited[x]) {
                        visited[x] =
                        true;q.add(x);
                    }
                }
            }
        System.out.println();
    }

    // Depth-First Search (DFS) - Recursive
    static void DFSRec(List<List<Integer>> adj, boolean[] visited, int s)
        { visited[s] = true;
        System.out.print(s + " ");
        for (int i : adj.get(s)) {
```

```

        if (!visited[i]) {
            DFSRec(adj,
                visited, i);
        }
    }
}

static void DFS(List<List<Integer>> adj,
    int s) { boolean[] visited = new
        boolean[adj.size()];DFSRec(adj,
        visited, s); System.out.println();
}

// Function to add edge in the graph
static void addEdge(List<List<Integer>> adj, int u, int v) {
    adj.get(u).add(v);
    adj.get(v).add(u);
}

public static void main(String[]
    args) { int V = 5;
    List<List<Integer>> adj = new
    ArrayList<>(V);for (int i = 0; i < V; i++) {
        adj.add(new ArrayList<>());
    }

    // Add edges to the
    graphaddEdge(adj, 0,
    1);
    addEdge(adj, 0, 2);
    addEdge(adj, 1, 3);
    addEdge(adj, 1, 4);
    addEdge(adj, 2, 4);

```

```
// BFS and DFS traversals

System.out.println("BFS starting from 0:");
bfs(adj, 0);

System.out.println("DFS starting from 0:");
DFS(adj, 0);
}
}
```

Output:

```
-
BFS starting from 0:
0 1 2 3 4
DFS starting from 0:
0 1 3 4 2
```

```
...Program finished with exit code 0
```

3) Write a program to find the MST using Prim's and Kruskal's algorithm.

Code:

```
import java.util.*;

// MST class with Prim's
Implementationclass MST {
    private static final int V = 5; // Number of vertices in the graph

    // Method to find minimum key
    value int minKey(int key[], boolean
mstSet[]) {
        int min =
        Integer.MAX_VALUE;int
        min_index = -1;
        for (int v = 0; v < V; v++) {
            if (!mstSet[v] && key[v] < min)
                { min = key[v];
                min_index = v;
                }
        }
        return min_index;
    }

    // Print the MST edges
    int printMst(int parent[], int
graph[][]) { for (int i = 1; i < V;
i++) {
        System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
    }
    return 0;
}

// Implementation of Prim's
MSTvoid primMST(int
graph[][]) {
```

```

int parent[] = new
int[V];int key[] = new
int[V];
boolean mstSet[] = new boolean[V];

Arrays.fill(key, Integer.MAX_VALUE);
Arrays.fill(mstSet, false);
key[0] = 0;
parent[0] = -1;

for (int count = 0; count < V - 1;
    count++) { int u = minKey(key,
    mstSet);
    mstSet[u] = true;

    for (int v = 0; v < V; v++) {
        if (graph[u][v] != 0 && !mstSet[v] && graph[u][v] < key[v]) {
            parent[v] = u;
            key[v] = graph[u][v];
        }
    }
}

System.out.println("Edges in the MST using Prim's:");
printMst(parent, graph);
}

}

// Kruskal's MST logic with union-
findclass KruskalMST {
    static class Edge {
        int src, dest, weight;

```

```

public Edge(int src, int dest, int weight) {
    this.src = src;
    this.dest = dest;
    this.weight =
    weight;
}

}

static class Subset
{ int parent,
rank;

public Subset(int parent, int rank)
{ this.parent = parent;
this.rank = rank;
}
}

public static void kruskalMST(int V, List<Edge>
edges) { int noOfEdges = 0;
Subset subsets[] = new Subset[V];
Edge results[] = new Edge[V]

for (int i = 0; i < V; i++) {
    subsets[i] = new Subset(i,
    0);
}

edges.sort(Comparator.comparingInt(e -> e.weight));
int j = 0;
while (noOfEdges < V - 1) {
    Edge nextEdge =
edges.get(j);
int x = findRoot(subsets, nextEdge.src);

```



```

    int y = findRoot(subsets, nextEdge.dest);

    if (x != y) {
        results[noOfEdges] =
            nextEdge; union(subsets, x,
                y); noOfEdges++;
    }
    j++;
    ;
}

System.out.println("Edges in the MST using
Kruskal's:"); int minCost = 0;
for (int i = 0; i < noOfEdges; i++) {
    System.out.println(results[i].src + " -- " + results[i].dest + " == " +
        results[i].weight); minCost += results[i].weight;
}

System.out.println("Total cost of MST: " + minCost);

}

private static void union(Subset[] subsets, int x, int
y) { int rootX = findRoot(subsets, x);
    int rootY = findRoot(subsets, y);

    if (subsets[rootY].rank <
        subsets[rootX].rank) {
        subsets[rootY].parent = rootX;
    } else if (subsets[rootX].rank < subsets[rootY].rank) {
        subsets[rootX].parent = rootY;
    } else {
        subsets[rootY].parent =
            rootX;
        subsets[rootX].rank++;
    }
}

```

```
}
```

```
private static int findRoot(Subset[] subsets, int  
    i) { if (subsets[i].parent != i) {  
        subsets[i].parent = findRoot(subsets, subsets[i].parent);  
    }  
    return subsets[i].parent;  
}  
}
```

```
public class Main {  
    public static void main(String[] args)  
    { Scanner sc = new  
Scanner(System.in);boolean  
running = true;  
  
while (running) {  
    System.out.println("\nSelect an  
option:");System.out.println("1.  
Prim's MST"); System.out.println("2.  
Kruskal's MST");  
System.out.println("3. Exit");  
System.out.print("Your choice: ");  
int choice = sc.nextInt();  
  
switch  
    (choice) {  
    case 1:  
        MST prim = new  
MST();int graph[][]  
        = {  
            {0, 2, 0, 6, 0},
```

```

        {2, 0, 3, 8, 5},
        {0, 3, 0, 0, 7},
        {6, 8, 0, 0, 9},
        {0, 5, 7, 9, 0}

    };

    prim.primMST(grap
    h);break;
case 2:
    int V = 6;
    List<KruskalMST.Edge> edges = new
        ArrayList<>(List.of(new KruskalMST.Edge(0,
            1, 10),
            new KruskalMST.Edge(0, 2, 6),
            new KruskalMST.Edge(0, 3, 5),
            new KruskalMST.Edge(1, 3, 15),
            new KruskalMST.Edge(2, 3, 4),
            new KruskalMST.Edge(3, 4, 8),
            new KruskalMST.Edge(4, 5, 2)
        ));
    KruskalMST.kruskalMST(V, edges);
    break;
case 3:
    running = false;
    System.out.println("Exiting...");
    break;
default:
    System.out.println("Invalid choice! Please select again.");

    }

    }
    sc.close();
}

}

```

Output:

```
Select an option:
1. Prim's MST
2. Kruskal's MST
3. Exit
Your choice: 1
Edges in the MST using Prim's:
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5

Select an option:
1. Prim's MST
2. Kruskal's MST
3. Exit
Your choice: 2
Edges in the MST using Kruskal's:
4 -- 5 == 2
2 -- 3 == 4
0 -- 3 == 5
3 -- 4 == 8
0 -- 1 == 10
Total cost of MST: 29

Select an option:
1. Prim's MST
2. Kruskal's MST
3. Exit
Your choice: 3
Exiting...

...Program finished with exit code 0
```
