SATYAM TRIPATHI
202151141

LAB 9

CODE

1. a.) Infix to Postfix

```
import java.util.Stack;

class InfixToPostfix
{



    static int Prec(char ch)
    {
        switch (ch)
        {
        case '+':
            return 1;
        case '-':
            return 1;

        case '*':
            return 2;
        case '/':

            return 2;

        case '^':
            return 3;
        }
        return -1;
    }

    static String infixToPostfix(String exp)
    {
        String ans = "";
        Stack<Character> stack = new Stack<>();
```

SATYAM TRIPATHI
202151141

```
        for (int i = 0; i<exp.length(); ++i)
        {
            char c = exp.charAt(i);
            if (Character.isLetterOrDigit(c))
                ans += c;
            else if (c == '(')
                stack.push(c);
            else if (c == ')')
            {
                while (!stack.isEmpty() &&
                        stack.peek() != '(')
                    ans += stack.pop();

                    stack.pop();
            }
            else
            {
                while (!stack.isEmpty() && Prec(c)
                        <= Prec(stack.peek())){

                    ans += stack.pop();
            }
                stack.push(c);
            }

        }

        while (!stack.isEmpty()){
            if(stack.peek() == '(')
                return "Invalid Expression";
            ans += stack.pop();
        }
        return ans;
    }

    public static void main(String[] args)
    {
        String exp = "c/(b*a+d)";
```

SATYAM TRIPATHI
202151141

```
        System.out.println(infixToPostfix(exp));
    }
}
```

OUTPUT

```
cba*d+/
```

b.) Infix to Prefix

```
import java.util.*;
public class InfixToPrefix {
    static int precedence(char c){
        switch (c){
            case '+':
            return 1;
            case '-':
                return 1;
            case '*':
            return 2;
            case '/':
                return 2;
            case '^':
                return 3;
        }
        return -1;
    }

    static String infixToPreFix(String s){

        StringBuffer ans = new StringBuffer();
        StringBuffer sb = new StringBuffer(s);
        sb.reverse();
        Stack<Character> stack = new Stack<Character>();

        char [] arr = new String(sb).toCharArray();
        for (int i = 0; i < arr.length; i++) {

            if (arr[i] == '(') {
                arr[i] = ')';
```

SATYAM TRIPATHI
202151141

```
                i++;
            }
            else if (arr[i] == ')') {
                arr[i] = '(';
                i++;
            }
        }
        for (int i = 0; i <arr.length ; i++) {
            char c = arr[i];
            if(precedence(c)>0){
                while(stack.isEmpty()==false &&
precedence(stack.peek())>=precedence(c)){
                    ans.append(stack.pop());
                }
                stack.push(c);
            }else if(c==')'){
                char x = stack.pop();
                while(x!='('){
                    ans.append(x);
                    x = stack.pop();
                }
            }else if(c=='('){
                stack.push(c);
            }else{
                ans.append(c);
            }
        }

        for (int i = 0; i <=stack.size() ; i++) {
            ans.append(stack.pop());
        }
        ans.reverse();

        String temp=ans.toString();
        return temp;
    }

    public static void main(String[] args) {
        String exp = "A+B*(C^D-E)";
```

SATYAM TRIPATHI
202151141

```
        System.out.println("Infix => " + exp);
        System.out.println("Prefix => " + infixToPreFix(exp));
    }
}
```

OUTPUT

```
Infix => A+B*(C^D-E)
Prefix => +A*B-^CDE
```

2. a) Evaluate postfix expression

```
import java.util.*;
public class practice
{
    static int evaluatePostfix(String exp)
    {
        Stack<Integer> stack=new Stack<>();
        for(int i=0;i<exp.length();i++)
        {
            char ch=exp.charAt(i);
            if(Character.isDigit(ch))
            stack.push(ch - '0');
            else
            {
            int a = stack.pop();
            int b = stack.pop();
            switch(ch)
            {
                case '+':
                stack.push(b+a);
                break;
                case '-':
                stack.push(b- a);
                break;
                case '/':
                stack.push(b/a);
                break;
                case '*':
                stack.push(b*a);
```

SATYAM TRIPATHI
202151141

```
                break;
            }
            }
        }
        return stack.pop();
    }
    public static void main(String[] args)
    {
        String exp="46+2/5*7+";
        System.out.println("Infix Expression: " + exp);
        System.out.println("Postfix: " + evaluatePostfix(exp));
    }
}
```

OUTPUT

```
Infix Expression: 46+2/5*7+
Postfix: 32
```

b) Evaluate prefix expression

```
import java.util.Stack;
public class practice
{
    static int evaluatePrefix(String exp)
    {
        Stack<Integer> stack=new Stack<>();
        for (int i = exp.length() - 1; i >= 0; i--)
        {
            char ch=exp.charAt(i);
            if(Character.isDigit(ch))
                stack.push(ch - '0');
            else
            {
                int a = stack.pop();
                int b = stack.pop();
                switch(ch)
                {
                    case '+':
```

SATYAM TRIPATHI
202151141

```
                    stack.push(a+b);
                    break;
                    case '-':
                    stack.push(a- b);
                    break;
                    case '/':
                    stack.push(a/b);
                    break;
                    case '*':
                    stack.push(a*b);
                    break;
                }
            }
        }
        return stack.pop();
    }
    public static void main(String[] args)
    {
        String exp="-+7*45+20";
        System.out.println("Infix Expression: " + exp);
        System.out.println("Prefix Evaluation: " + evaluatePrefix(exp));
    }
}
```

OUTPUT

```
Infix Expression: -+7*45+20
Prefix Evaluation: 25
```

3. Implement Heap data structure

a. Min heap

```
class Main
{
    private int[] Heap;
    private int size;
    private int maxsize;
    private static final int FRONT = 1;
    public Main(int maxsize)
```

SATYAM TRIPATHI
202151141

```
    {
        this.maxsize = maxsize;
        this.size = 0;
        Heap = new int[this.maxsize + 1];
        Heap[0] = Integer.MIN_VALUE;
    }
    private int parent(int pos)
    {
        return pos / 2;
    }
    private int leftChild(int pos)
    {
        return (2 * pos);
    }
    private int rightChild(int pos)
    {
        return (2 * pos) + 1;
    }
    private boolean isLeaf(int pos)
    {
        if (pos > (size / 2))
        {
            return true;
        }
        return false;
    }
    private void swap(int fpos, int spos)
    {
        int tmp;
        tmp = Heap[fpos];
        Heap[fpos] = Heap[spos];
        Heap[spos] = tmp;
    }
    private void minHeapify(int pos)
    {
        if(!isLeaf(pos))
        {
            int swapPos =
```

SATYAM TRIPATHI
202151141

```
Heap[leftChild(pos)]<Heap[rightChild(pos)]?leftChild(pos):rightChild(pos);
            if(Heap[pos]>Heap[leftChild(pos)] || Heap[pos]>
Heap[rightChild(pos)]){
            swap(pos,swapPos);
            minHeapify(swapPos);
            }
        }
    }
    public void insert(int element)
    {
    if (size >= maxsize)
    {
        return;
    }
    Heap[++size] = element;
    int current = size;
    while (Heap[current] < Heap[parent(current)])
    {
        swap(current, parent(current));
        current = parent(current);
    }
    }
    public void print()
    {
        for (int i = 1; i <= size / 2; i++)
        {
        System.out.print(" Parent : " + Heap[i]+ " Left child : " + Heap[2 * i]+
" Right child :" + Heap[2 * i + 1]);
        System.out.println();
        }
    }
    public int remove()
    {
        int popped = Heap[FRONT];
        Heap[FRONT] = Heap[size--];
        minHeapify(FRONT);
        return popped;
    }
```

SATYAM TRIPATHI
202151141

```
    public static void main(String[] arg)
    {
        System.out.println("The Min Heap is ");
        Main minHeap = new Main(15);
        minHeap.insert(5);
        minHeap.insert(3);
        minHeap.insert(17);
        minHeap.insert(10);
        minHeap.insert(84);
        minHeap.insert(19);
        minHeap.insert(6);
        minHeap.insert(22);
        minHeap.insert(9);
        minHeap.print();
        System.out.println("The Min val is " + minHeap.remove());
    }
}
```

OUTPUT

```
The Min Heap is
 Parent : 3 Left child : 5 Right child :6
 Parent : 5 Left child : 9 Right child :84
 Parent : 6 Left child : 19 Right child :17
 Parent : 9 Left child : 22 Right child :10
The Min val is 3
```

b. Max heap

```
public class Main
{
    private int[] Heap;
    private int size;
    private int maxsize;
    public Main(int maxsize)
    {
        this.maxsize = maxsize;
        this.size = 0;
        Heap = new int[this.maxsize];
```

SATYAM TRIPATHI
202151141

```
    }
    private int parent(int pos)
    {
        return (pos - 1) / 2;
    }
    private int leftChild(int pos)
    {
        return (2 * pos) + 1;
    }
    private int rightChild(int pos)
    {
        return (2 * pos) + 2;
    }
    private boolean isLeaf(int pos)
    {
        if (pos > (size / 2) && pos <= size)
        {
            return true;
        }
        return false;
    }
    private void swap(int fpos, int spos)
    {
        int tmp;
        tmp = Heap[fpos];
        Heap[fpos] = Heap[spos];
        Heap[spos] = tmp;
    }
    private void maxHeapify(int pos)
    {
    if (isLeaf(pos))
    return;
    if (Heap[pos] < Heap[leftChild(pos)]
    || Heap[pos] < Heap[rightChild(pos)])
    {
    if (Heap[leftChild(pos)]
    > Heap[rightChild(pos)])
    {
    swap(pos, leftChild(pos));
```

SATYAM TRIPATHI
202151141

```
    maxHeapify(leftChild(pos));
    }
    else
    {
    swap(pos, rightChild(pos));
    maxHeapify(rightChild(pos));
    }
    }
    }
    public void insert(int element)
    {
    Heap[size] = element;
    int current = size;
    while (Heap[current] > Heap[parent(current)])
    {
    swap(current, parent(current));
    current = parent(current);
    }
    size++;
    }
    public void print()
    {
        for(int i=0;i<size/2;i++)
        {
            System.out.print("Parent Node : " + Heap[i] );
            if(leftChild(i)<size)
                System.out.print( " Left Child Node: " + Heap[leftChild(i)]);
            if(rightChild(i)<size)
                System.out.print(" Right Child Node: "+ Heap[rightChild(i)]);
            System.out.println();
        }
    }
    public int extractMax()
    {
        int popped = Heap[0];
        Heap[0] = Heap[--size];
        maxHeapify(0);
        return popped;
    }
```

SATYAM TRIPATHI
202151141

```
    public static void main(String[] arg)
    {
        System.out.println("The Max Heap is ");
        Main maxHeap = new Main(15);
        maxHeap.insert(5);
        maxHeap.insert(3);
        maxHeap.insert(17);
        maxHeap.insert(10);
        maxHeap.insert(84);
        maxHeap.insert(19);
        maxHeap.insert(6);
        maxHeap.insert(22);
        maxHeap.insert(9);
        maxHeap.print();
        System.out.println("The max val is " + maxHeap.extractMax());
    }
}
```

OUTPUT

```
The Max Heap is
Parent Node : 84 Left Child Node: 22 Right Child Node: 19
Parent Node : 22 Left Child Node: 17 Right Child Node: 10
Parent Node : 19 Left Child Node: 5 Right Child Node: 6
Parent Node : 17 Left Child Node: 3 Right Child Node: 9
The max val is 84
```

4. Implement Heap Sort

```
public class practice
{
    public void sort(int arr[])
    {
        int n = arr.length;
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);
        for (int i = n - 1; i > 0; i--)
        {
            int temp = arr[0];
            arr[0] = arr[i];
```

SATYAM TRIPATHI
202151141

```
            arr[i] = temp;
            heapify(arr, i, 0);
        }
    }
    void heapify(int arr[], int n, int i)
    {
        int largest = i;
        int l = 2 * i + 1;
        int r = 2 * i + 2;
        if (l < n && arr[l] > arr[largest])
            largest = l;
        if (r < n && arr[r] > arr[largest])
            largest = r;
        if (largest != i)
        {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;
            heapify(arr, n, largest);
        }
    }
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }
    public static void main(String args[])
    {
        int arr[] = { 112, 101, 13, 55, 46, 7 ,48,74,69,3};
        int n = arr.length;
        Main ob = new Main();
        ob.sort(arr);
        System.out.println("Sorted array is");
        printArray(arr);
    }
    }
    public class Main
```

SATYAM TRIPATHI
202151141

```
{
public void sort(int arr[])
{
    int n = arr.length;
    for (int i = n / 2 - 1; i >= 0; i--)
    heapify(arr, n, i);
    for (int i = n - 1; i > 0; i--)
    {
    int temp = arr[0];
    arr[0] = arr[i];
    arr[i] = temp;
    heapify(arr, i, 0);
}
}
void heapify(int arr[], int n, int i)
{
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < n && arr[l] > arr[largest])
    largest = l;
    if (r < n && arr[r] > arr[largest])
    largest = r;
    if (largest != i)
    {
    int swap = arr[i];
    arr[i] = arr[largest];
    arr[largest] = swap;
    heapify(arr, n, largest);
    }
}
static void printArray(int arr[])
{
    int n = arr.length;
    for (int i = 0; i < n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
}
public static void main(String args[])
```

SATYAM TRIPATHI
202151141

```
    {
    int arr[] = { 112, 101, 13, 55, 46, 7 ,48,74,69,3};
    int n = arr.length;
    Main ob = new Main();
    ob.sort(arr);
    System.out.println("Sorted array is");
    printArray(arr);
    }
}
```

OUTPUT

```
Sorted array is
3 7 13 46 48 55 69 74 101 112
```