Subject: Algorithm and Data Structure
Assignment 3

1. Implement a Stack using an array.

- **Test Case 1**:
  Input: Push 5, 3, 7, Pop
  Output: Stack = [5, 3], Popped element = 7
- **Test Case 2**:
  Input: Push 10, Push 20, Pop, Push 15
  Output: Stack = [10, 15], Popped element = 20

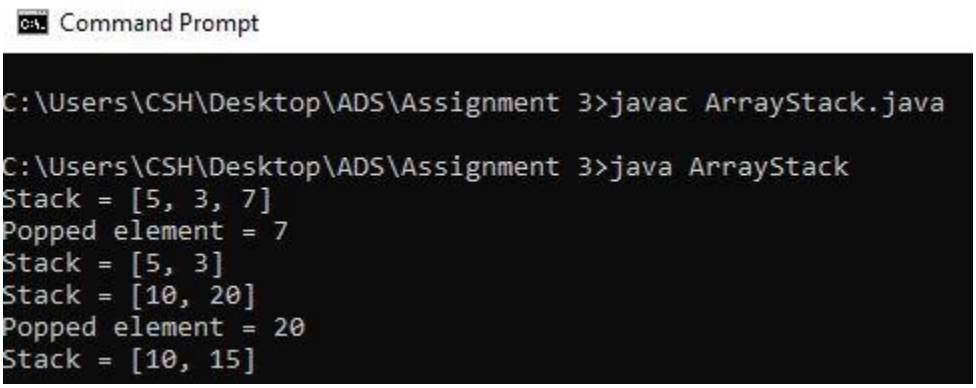
Code-

```
import java.util.Arrays;


public class ArrayStack {
   private int[] stack;
   private int top;
   private int capacity;


   public ArrayStack(int size) {
      stack = new int[size];
      top = -1;
      capacity = size;
   }


   public void push(int element) {
      if (top == capacity - 1) {
         System.out.println("Stack Overflow ");
         resizeStack();
      }
      stack[++top] = element;
   }
```

```
public int pop() {
    if (top == -1) {
        System.out.println("Stack Underflow");
        return -1;
    }
    return stack[top--];
}

private void resizeStack() {
    capacity = capacity * 2;
    stack = Arrays.copyOf(stack, capacity);
}

public void displayStack() {
    if (top == -1) {
        System.out.println("Stack is empty.");
    } else {
        System.out.print("Stack = [");
        for (int i = 0; i <= top; i++) {
            System.out.print(stack[i]);
            if (i < top) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }
}

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

```
        ArrayStack stack1 = new ArrayStack(3);
        stack1.push(5);
        stack1.push(3);
        stack1.push(7);
        stack1.displayStack();
        System.out.println("Popped element = " + stack1.pop());
        stack1.displayStack();

        ArrayStack stack2 = new ArrayStack(2);
        stack2.push(10);
        stack2.push(20);
        stack2.displayStack();
        System.out.println("Popped element = " + stack2.pop());
        stack2.push(15);
        stack2.displayStack();
    }
}
```

Output -



2. Check for balanced parentheses using a stack.

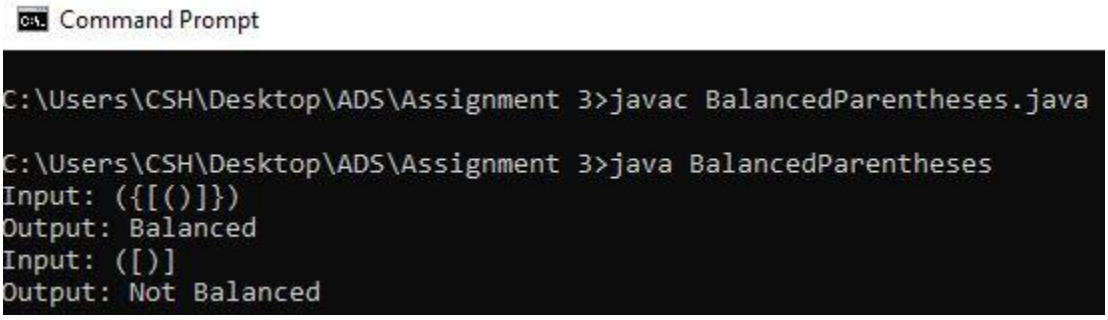
- **Test Case 1**:
  Input: "({[()]})"
  Output: Balanced
- **Test Case 2**:
  Input: "([)]"
  Output: Not Balanced

Code –

```
import java.util.Stack;


public class BalancedParentheses {


   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()) {
               return false;
            }
            char top = stack.pop();
            if (!isMatchingPair(top, 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) {

        String expression1 = "({[()]})";
        System.out.println("Input: " + expression1);
        if (isBalanced(expression1)) {
            System.out.println("Output: Balanced");
        } else {
            System.out.println("Output: Not Balanced");
        }

        String expression2 = "([)]";
        System.out.println("Input: " + expression2);
        if (isBalanced(expression2)) {
            System.out.println("Output: Balanced");
        } else {
            System.out.println("Output: Not Balanced");
        }
    }
}
```

Output –



3. Reverse a string using a stack.

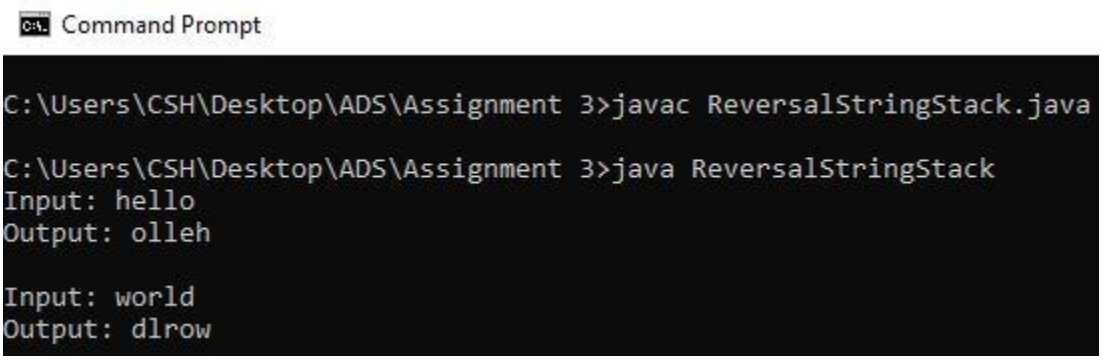
- **Test Case 1**:
  Input: "hello"
  Output: "olleh"
- **Test Case 2**:
  Input: "world"
  Output: "dlrow"

Code –

```
import java.util.Stack;


public class ReversalStringStack {


   public static String reverseString(String input) {
      Stack<Character> stack = new Stack<>();


      // Push each character of the string onto the stack
      for (char c : input.toCharArray()) {
         stack.push(c);
      }


      // Pop each character and build the reversed string
      StringBuilder reversed = new StringBuilder();
      while (!stack.isEmpty()) {
```

```
            reversed.append(stack.pop());
        }

        return reversed.toString();
    }

    public static void main(String[] args) {
        // Test Case 1
        String input1 = "hello";
        System.out.println("Input: " + input1);
        System.out.println("Output: " + reverseString(input1) + "\n");  // Output: "olleh"

        // Test Case 2
        String input2 = "world";
        System.out.println("Input: " + input2);
        System.out.println("Output: " + reverseString(input2));  // Output: "dlrow"
    }
}
```

Output –



4. Evaluate a postfix expression using a stack.

- **Test Case 1**:
  Input: "5 3 + 2 *"
  Output: 16
- **Test Case 2**:
  Input: "4 5 * 6 /"
  Output: 3

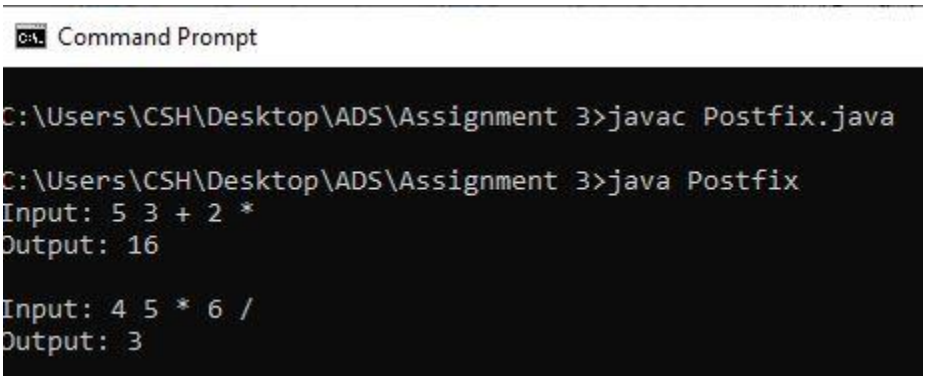
Code –

```
import java.util.Stack;


public class Postfix {


   public static int evaluatePostfix(String expression) {
      Stack<Integer> stack = new Stack<>();


      String[] tokens = expression.split(" ");


      for (String token : tokens) {


         if (isNumeric(token)) {
            stack.push(Integer.parseInt(token));
         } else {


            int operand2 = stack.pop();
            int operand1 = stack.pop();


            switch (token) {
               case "+":
                  stack.push(operand1 + operand2);
                  break;
               case "-":
                  stack.push(operand1 - operand2);
```

```
                break;
            case "*":
                stack.push(operand1 * operand2);
                break;
            case "/":
                stack.push(operand1 / operand2);
                break;
          }
        }
    }

    return stack.pop();
}

public static boolean isNumeric(String str) {
    try {
        Integer.parseInt(str);
        return true;
    } catch (NumberFormatException e) {
        return false;
    }
}

public static void main(String[] args) {

    String expression1 = "5 3 + 2 *";
    System.out.println("Input: " + expression1);
    System.out.println("Output: " + evaluatePostfix(expression1) + "\n");
```

```
        String expression2 = "4 5 * 6 /";
        System.out.println("Input: " + expression2);
        System.out.println("Output: " + evaluatePostfix(expression2));
    }
}
```

Output –



5. Convert an infix expression to postfix using a stack.

- **Test Case 1**:
  Input: "A + B * C"
  Output: "A B C * +"
- **Test Case 2**:
  Input: "A * B + C / D"
  Output: "A B * C D / +"

Code –

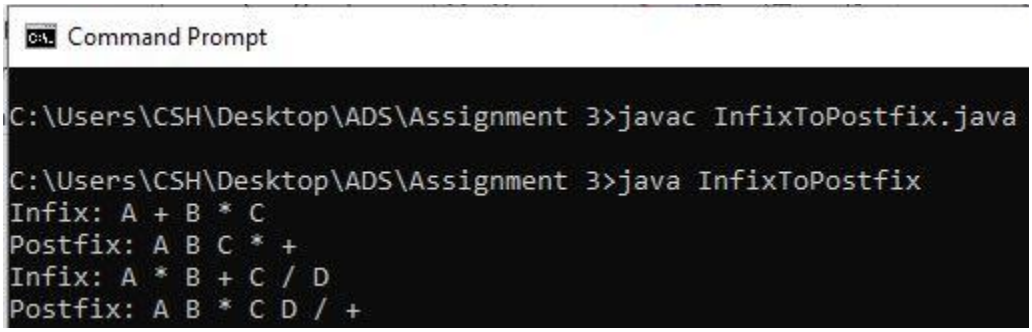
```
import java.util.Stack;


public class InfixToPostfix {


    private static int precedence(char ch) {
        switch (ch) {
            case '+':
            case '-':
                return 1;
            case '*':
```

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

public static String infixToPostfix(String expression) {

    Stack<Character> stack = new Stack<>();
    StringBuilder result = new StringBuilder();

    for (int i = 0; i < expression.length(); i++) {
        char c = expression.charAt(i);

        if (Character.isWhitespace(c)) {
            continue;
        }

        if (Character.isLetterOrDigit(c)) {
            result.append(c).append(" ");
        }

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

        else if (c == ')') {
```

```
            while (!stack.isEmpty() && stack.peek() != '(') {
                result.append(stack.pop()).append(" ");
            }
            stack.pop();
        }

        else {
            while (!stack.isEmpty() && precedence(stack.peek()) >= precedence(c)) {
                result.append(stack.pop()).append(" ");
            }
            stack.push(c);
        }
    }

    while (!stack.isEmpty()) {
        result.append(stack.pop()).append(" ");
    }

    return result.toString().trim();
}

public static void main(String[] args) {

    String exp1 = "A + B * C";
    System.out.println("Infix: " + exp1);
    System.out.println("Postfix: " + infixToPostfix(exp1));

    String exp2 = "A * B + C / D";
    System.out.println("Infix: " + exp2);
```

```
        System.out.println("Postfix: " + infixToPostfix(exp2));
    }
}
```

Output –



6. Implement a Queue using an array.

- **Test Case 1**:
  Input: Enqueue 5, Enqueue 10, Dequeue
  Output: Queue = [10], Dequeued element = 5
- **Test Case 2**:
  Input: Enqueue 1, 2, 3, Dequeue, Dequeue
  Output: Queue = [3], Dequeued elements = 1, 2

Code –

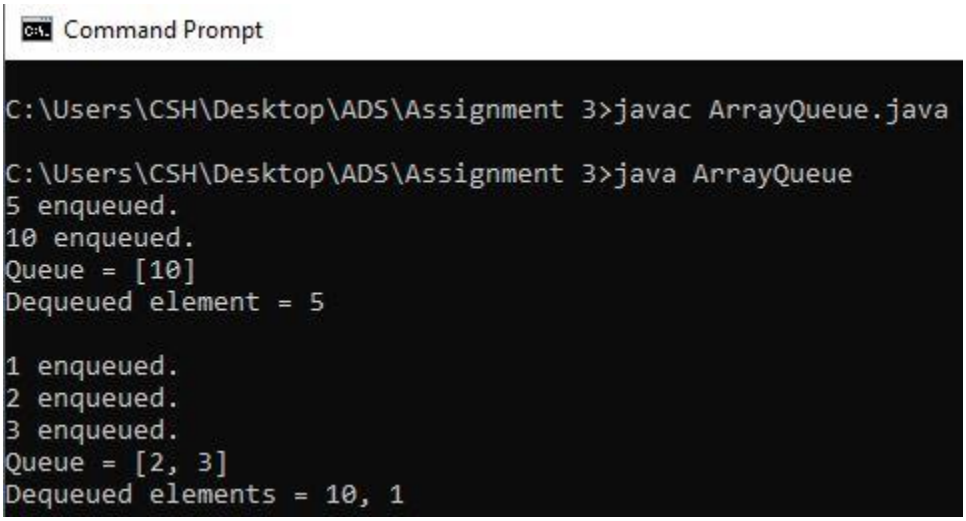
```
class Queue {
    private int[] queue;
    private int front;
    private int rear;
    private int size;
    private int capacity;

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

```
public void enqueue(int value) {
    if (isFull()) {
        System.out.println("Queue is full! Cannot enqueue " + value);
        return;
    }
    rear = (rear + 1) % capacity;
    queue[rear] = value;
    size++;
}

public int dequeue() {
    if (isEmpty()) {
        System.out.println("Queue is empty! Cannot dequeue.");
        return -1;
    }
    int dequeuedValue = queue[front];
    front = (front + 1) % capacity;
    size--;
    return dequeuedValue;
}

public boolean isEmpty() {
    return size == 0;
}

public boolean isFull() {
    return size == capacity;
}
```

```
public void displayQueue() {
    if (isEmpty()) {
        System.out.println("Queue is empty");
        return;
    }
    System.out.print("Queue = [");
    for (int i = 0; i < size; i++) {
        System.out.print(queue[(front + i) % capacity]);
        if (i != size - 1) {
            System.out.print(", ");
        }
    }
    System.out.println("]");
}

public static void main(String[] args) {

    Queue q1 = new Queue(5);
    q1.enqueue(5);
    q1.enqueue(10);
    int dequeued1 = q1.dequeue();
    q1.displayQueue();
    System.out.println("Dequeued element = " + dequeued1);

    Queue q2 = new Queue(5);
    q2.enqueue(1);
    q2.enqueue(2);
    q2.enqueue(3);
```

```
        int dequeued2_1 = q2.dequeue();
        int dequeued2_2 = q2.dequeue();
        q2.displayQueue();
        System.out.println("Dequeued elements = " + dequeued2_1 + ", " + dequeued2_2);
    }
}
```

Output –



7. Implement a Circular Queue using an array.

- **Test Case 1**:
  Input: Enqueue 4, 5, 6, 7, Dequeue, Enqueue 8
  Output: Queue = [8, 5, 6, 7]
- **Test Case 2**:
  Input: Enqueue 1, 2, 3, 4, Dequeue, Dequeue, Enqueue 5
  Output: Queue = [5, 3, 4]

Code –

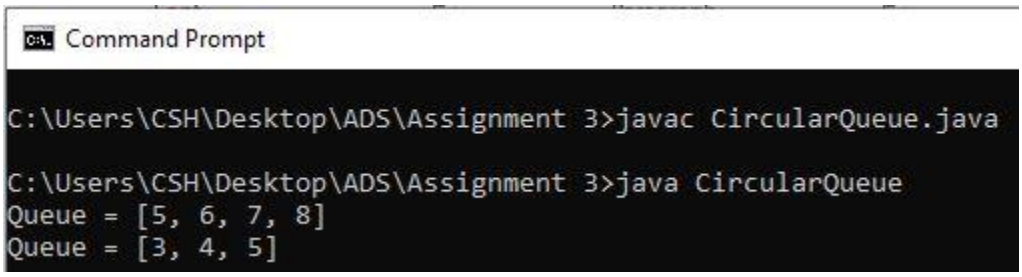
```
class CircularQueue {
    private int[] queue;
    private int front;
    private int rear;
    private int size;
    private int capacity;

    public CircularQueue(int capacity) {
```

```
    this.capacity = capacity;
    queue = new int[capacity];
    front = -1;
    rear = -1;
    size = 0;
}

public void enqueue(int value) {
    if (isFull()) {
        System.out.println("Queue is full! Cannot enqueue " + value);
        return;
    }
    if (isEmpty()) {
        front = 0;
    }
    rear = (rear + 1) % capacity;
    queue[rear] = value;
    size++;
}

public int dequeue() {
    if (isEmpty()) {
        System.out.println("Queue is empty! Cannot dequeue.");
        return -1;
    }
    int dequeuedValue = queue[front];
    front = (front + 1) % capacity;
    size--;
    return dequeuedValue;
```

```
}

public boolean isEmpty() {
   return size == 0;
}

public boolean isFull() {
   return size == capacity;
}

public void displayQueue() {
   if (isEmpty()) {
      System.out.println("Queue is empty");
      return;
   }
   System.out.print("Queue = [");
   for (int i = 0; i < size; i++) {
      System.out.print(queue[(front + i) % capacity]);
      if (i != size - 1) {
         System.out.print(", ");
      }
   }
   System.out.println("]");
}

public static void main(String[] args) {

   CircularQueue cq1 = new CircularQueue(5);
   cq1.enqueue(4);
```

```
        cq1.enqueue(5);
        cq1.enqueue(6);
        cq1.enqueue(7);
        cq1.dequeue();
        cq1.enqueue(8);
        cq1.displayQueue();

        CircularQueue cq2 = new CircularQueue(5);
        cq2.enqueue(1);
        cq2.enqueue(2);
        cq2.enqueue(3);
        cq2.enqueue(4);
        cq2.dequeue();
        cq2.dequeue();
        cq2.enqueue(5);
        cq2.displayQueue();
    }
}
```

Output –



8. Implement a Queue using two Stacks.

- **Test Case 1**:
  Input: Enqueue 3, Enqueue 7, Dequeue
  Output: Queue = [7], Dequeued element = 3
- **Test Case 2**:
  Input: Enqueue 10, 20, Dequeue, Dequeue
  Output: Queue = [], Dequeued elements = 10, 20

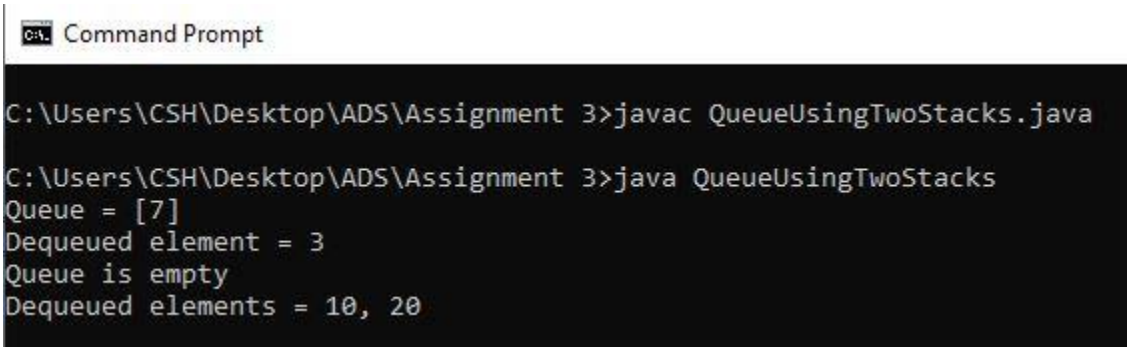
Code –

```
import java.util.Stack;

class QueueUsingTwoStacks {
    private Stack<Integer> stack1;
    private Stack<Integer> stack2;

    // Constructor to initialize the two stacks
    public QueueUsingTwoStacks() {
        stack1 = new Stack<>();
        stack2 = new Stack<>();
    }

    // Enqueue operation
    public void enqueue(int value) {
        stack1.push(value);
    }

    // Dequeue operation
    public int dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty! Cannot dequeue.");
            return -1;
        }
        if (stack2.isEmpty()) {
            while (!stack1.isEmpty()) {
                stack2.push(stack1.pop());
            }
        }
```

```
    return stack2.pop();
}

// Check if the queue is empty
public boolean isEmpty() {
    return stack1.isEmpty() && stack2.isEmpty();
}

// Display the queue
public void displayQueue() {
    if (isEmpty()) {
        System.out.println("Queue is empty");
        return;
    }
    Stack<Integer> tempStack = new Stack<>();

    // First, move elements from stack2 (front of the queue) to tempStack
    while (!stack2.isEmpty()) {
        tempStack.push(stack2.pop());
    }

    // Then, move elements from stack1 (back of the queue) to tempStack
    for (int i = stack1.size() - 1; i >= 0; i--) {
        tempStack.push(stack1.get(i));
    }

    System.out.print("Queue = [");
    while (!tempStack.isEmpty()) {
        int val = tempStack.pop();
```

```
            System.out.print(val);
            if (!tempStack.isEmpty()) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }

    public static void main(String[] args) {
        // Test Case 1
        QueueUsingTwoStacks q1 = new QueueUsingTwoStacks();
        q1.enqueue(3);
        q1.enqueue(7);
        int dequeued1 = q1.dequeue();
        q1.displayQueue(); // Expected Output: Queue = [7]
        System.out.println("Dequeued element = " + dequeued1); // Expected Output: Dequeued element =
3

        // Test Case 2
        QueueUsingTwoStacks q2 = new QueueUsingTwoStacks();
        q2.enqueue(10);
        q2.enqueue(20);
        int dequeued2_1 = q2.dequeue();
        int dequeued2_2 = q2.dequeue();
        q2.displayQueue(); // Expected Output: Queue = []
        System.out.println("Dequeued elements = " + dequeued2_1 + ", " + dequeued2_2); // Expected
Output: Dequeued elements = 10, 20
    }
}
```

Output –



9. Implement a Min-Heap.

- **Test Case 1**:
  Input: Insert 10, 15, 20, 17, Extract Min
  Output: Min-Heap = [15, 17, 20], Extracted Min = 10
- **Test Case 2**:
  Input: Insert 30, 40, 20, 50, Extract Min
  Output: Min-Heap = [30, 40, 50], Extracted Min = 20

Code –

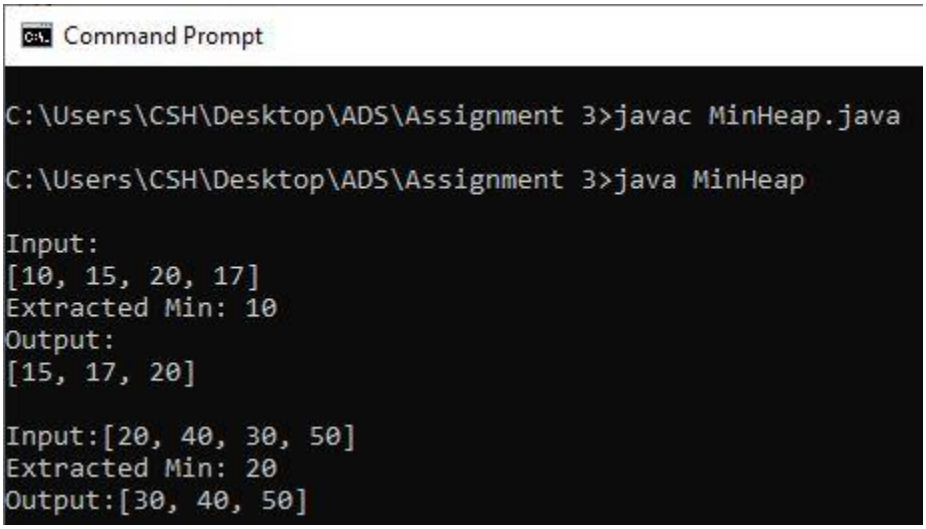
```
import java.util.ArrayList;


public class MinHeap {
   private ArrayList<Integer> heap;


   public MinHeap() {
      heap = new ArrayList<>();
   }
   public void insert(int value) {
      heap.add(value);
      int index = heap.size() - 1;


      while (index > 0 && heap.get(index) < heap.get(parent(index))) {
         swap(index, parent(index));
         index = parent(index);
      }
```

```
}

public int extractMin() {
    if (heap.isEmpty()) {
        throw new IllegalStateException("Heap is empty");
    }

    int min = heap.get(0);
    int last = heap.remove(heap.size() - 1);

    if (!heap.isEmpty()) {
        heap.set(0, last);
        heapify(0);
    }

    return min;
}

private void heapify(int i) {
    int left = leftChild(i);
    int right = rightChild(i);
    int smallest = i;

    if (left < heap.size() && heap.get(left) < heap.get(smallest)) {
        smallest = left;
    }

    if (right < heap.size() && heap.get(right) < heap.get(smallest)) {
        smallest = right;
```

```
    }

    if (smallest != i) {
        swap(i, smallest);
        heapify(smallest);
    }
}

private int parent(int i) {
    return (i - 1) / 2;
}

private int leftChild(int i) {
    return 2 * i + 1;
}

private int rightChild(int i) {
    return 2 * i + 2;
}

private void swap(int i, int j) {
    int temp = heap.get(i);
    heap.set(i, heap.get(j));
    heap.set(j, temp);
}
public void printHeap() {
    System.out.println(heap);
}
```

```
    public static void main(String[] args) {
        MinHeap minHeap = new MinHeap();

        System.out.println();
        minHeap.insert(10);
        minHeap.insert(15);
        minHeap.insert(20);
        minHeap.insert(17);
        System.out.println("Input:");
        minHeap.printHeap();
        System.out.println("Extracted Min: " + minHeap.extractMin());
        System.out.println("Output:");
        minHeap.printHeap();

        minHeap = new MinHeap();

        System.out.println();
        minHeap.insert(30);
        minHeap.insert(40);
        minHeap.insert(20);
        minHeap.insert(50);
        System.out.print("Input:");
        minHeap.printHeap();
        System.out.println("Extracted Min: " + minHeap.extractMin());
        System.out.print("Output:");
        minHeap.printHeap();
    }
}
```

Output –



10. Implement a Max-Heap.

- **Test Case 1**:
  Input: Insert 12, 7, 15, 5, Extract Max
  Output: Max-Heap = [12, 7, 5], Extracted Max = 15
- **Test Case 2**:
  Input: Insert 8, 20, 10, 3, Extract Max
  Output: Max-Heap = [10, 8, 3], Extracted Max = 20

Code –

```
import java.util.ArrayList;


public class MaxHeap {
   private ArrayList<Integer> heap;


   public MaxHeap() {
      heap = new ArrayList<>();
   }


   public void insert(int value) {
      heap.add(value);
      int index = heap.size() - 1;
```

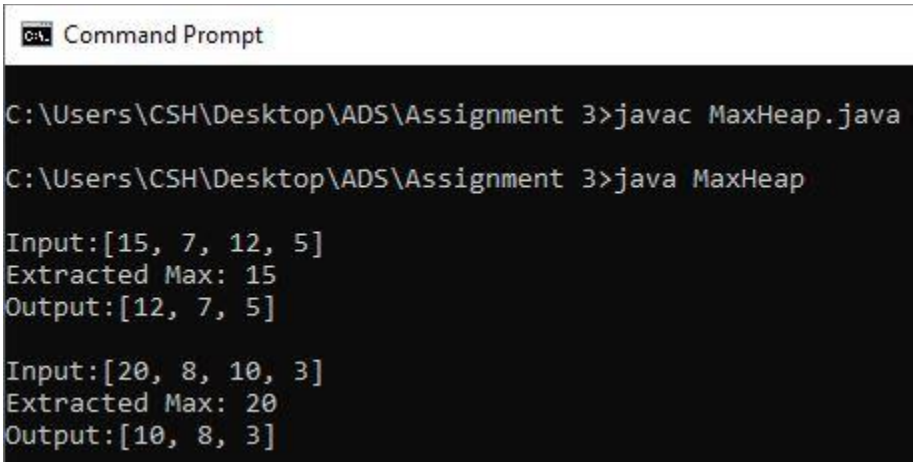
```
    while (index > 0 && heap.get(index) > heap.get(parent(index))) {
        swap(index, parent(index));
        index = parent(index);
    }
}

public int extractMax() {
    if (heap.isEmpty()) {
        throw new IllegalStateException("Heap is empty");
    }

    int max = heap.get(0);
    int last = heap.remove(heap.size() - 1);

    if (!heap.isEmpty()) {
        heap.set(0, last);
        heapify(0);
    }

    return max;
}

private void heapify(int i) {
    int left = leftChild(i);
    int right = rightChild(i);
    int largest = i;

    if (left < heap.size() && heap.get(left) > heap.get(largest)) {
        largest = left;
```

```
    }

    if (right < heap.size() && heap.get(right) > heap.get(largest)) {
        largest = right;
    }

    if (largest != i) {
        swap(i, largest);
        heapify(largest);
    }
}

private int parent(int i) {
    return (i - 1) / 2;
}

private int leftChild(int i) {
    return 2 * i + 1;
}

private int rightChild(int i) {
    return 2 * i + 2;
}

private void swap(int i, int j) {
    int temp = heap.get(i);
    heap.set(i, heap.get(j));
    heap.set(j, temp);
}
```

```
public void printHeap() {
    System.out.println(heap);
}

public static void main(String[] args) {
    MaxHeap maxHeap = new MaxHeap();

    System.out.println();
    maxHeap.insert(12);
    maxHeap.insert(7);
    maxHeap.insert(15);
    maxHeap.insert(5);
    System.out.print("Input:");
    maxHeap.printHeap();
    System.out.println("Extracted Max: " + maxHeap.extractMax());
    System.out.print("Output:");
    maxHeap.printHeap();

    maxHeap = new MaxHeap();

    System.out.println();
    maxHeap.insert(8);
    maxHeap.insert(20);
    maxHeap.insert(10);
    maxHeap.insert(3);
    System.out.print("Input:");
    maxHeap.printHeap();
    System.out.println("Extracted Max: " + maxHeap.extractMax());
    System.out.print("Output:");
```

```
        maxHeap.printHeap();
    }
}
```

Output –



11. Sort an arraycls using a heap (Heap Sort).

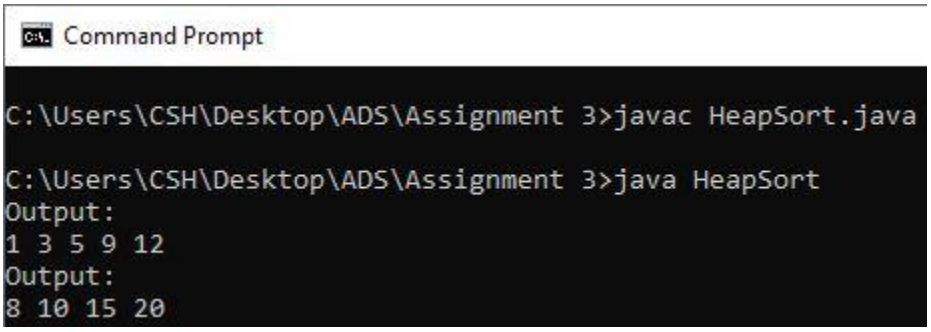
- **Test Case 1**:
  Input: [5, 1, 12, 3, 9]
  Output: [1, 3, 5, 9, 12]
- **Test Case 2**:
  Input: [20, 15, 8, 10]
  Output: [8, 10, 15, 20]

Code –

```
public class HeapSort {


    public void heapify(int arr[], int n, int i) {
        int largest = i;
        int left = 2 * i + 1;
        int right = 2 * i + 2;


        if (left < n && arr[left] > arr[largest])
            largest = left;


        if (right < n && arr[right] > arr[largest])
```

```
        largest = right;

    if (largest != i) {
        int swap = arr[i];
        arr[i] = arr[largest];
        arr[largest] = swap;

        heapify(arr, n, largest);
    }
}

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--) {
        // Move current root to end
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;

        heapify(arr, i, 0);
    }
}

public static void main(String args[]) {
    HeapSort heapSort = new HeapSort();
```

```
        int arr1[] = {5, 1, 12, 3, 9};
        heapSort.sort(arr1);
        System.out.println("Output: ");
        for (int i : arr1) {
            System.out.print(i + " ");
        }

        System.out.println();

        int arr2[] = {20, 15, 8, 10};
        heapSort.sort(arr2);
        System.out.println("Output: ");
        for (int i : arr2) {
            System.out.print(i + " ");
        }
    }
}
```



12. Find the kth largest element in a stream of numbers using a heap.

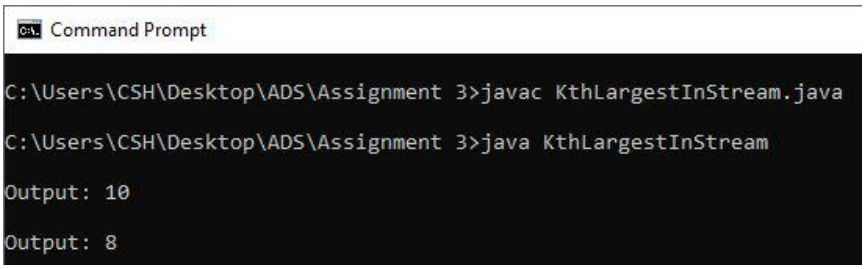
- **Test Case 1**:
  Input: Stream = [3, 10, 5, 20, 15], k = 3
  Output: 10
- **Test Case 2**:
  Input: Stream = [7, 4, 8, 2, 9], k = 2
  Output: 8

Code –

```
import java.util.PriorityQueue;


public class KthLargestInStream {


   private PriorityQueue<Integer> minHeap;

   private int k;


   public KthLargestInStream(int k) {

      this.k = k;

      this.minHeap = new PriorityQueue<>(k);

   }


   public void add(int num) {

      if (minHeap.size() < k) {

         minHeap.add(num);

      } else if (num > minHeap.peek()) {

         minHeap.poll();

         minHeap.add(num);

      }

   }


   public int getKthLargest() {

      if (minHeap.size() < k) {

         throw new IllegalStateException("Less than k elements in the stream");
```

```
        }
        return minHeap.peek();
    }

    public static void main(String[] args) {

        System.out.println("");
        KthLargestInStream kthLargest1 = new KthLargestInStream(3);
        int[] stream1 = {3, 10, 5, 20, 15};
        for (int num : stream1) {
            kthLargest1.add(num);
        }
        System.out.println("Output: " + kthLargest1.getKthLargest());

        System.out.println();
        KthLargestInStream kthLargest2 = new KthLargestInStream(2);
        int[] stream2 = {7, 4, 8, 2, 9};
        for (int num : stream2) {
            kthLargest2.add(num);
        }
        System.out.println("Output: " + kthLargest2.getKthLargest());
    }
}
```

Output –



13. Implement a Priority Queue using a heap.

- **Test Case 1**:
  Input: Enqueue with priorities: 3 (priority 1), 10 (priority 3), 5 (priority 2), Dequeue
  Output: Dequeued element = 10 (highest priority), Priority Queue = [5, 3]
- **Test Case 2**:
  Input: Enqueue with priorities: 7 (priority 4), 8 (priority 2), 6 (priority 3), Dequeue
  Output: Dequeued element = 7, Priority Queue = [6, 8]

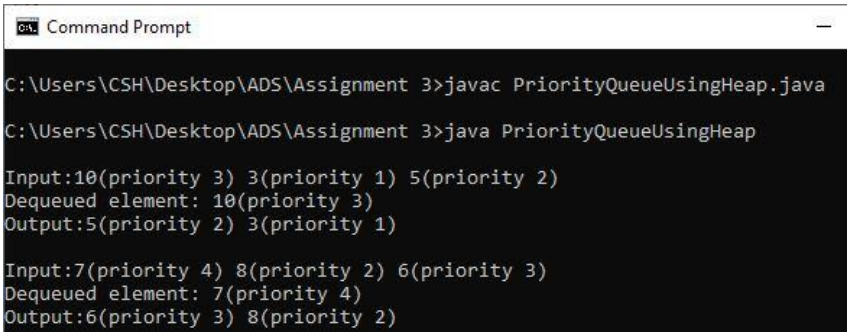
Code –

```
import java.util.PriorityQueue;

import java.util.Comparator;


class Element {

   int value;

   int priority;


   public Element(int value, int priority) {

      this.value = value;

      this.priority = priority;

   }


   public String toString() {

      return value + "(priority " + priority + ")";

   }

}


class PriorityQueueUsingHeap {


   private PriorityQueue<Element> maxHeap;


   public PriorityQueueUsingHeap() {


      maxHeap = new PriorityQueue<>(new Comparator<Element>() {
```

```
        public int compare(Element e1, Element e2) {

            return e2.priority - e1.priority;
        }
    });
}

public void enqueue(int value, int priority) {
    Element element = new Element(value, priority);
    maxHeap.add(element);
}

public Element dequeue() {
    if (maxHeap.isEmpty()) {
        throw new IllegalStateException("Priority Queue is empty");
    }
    return maxHeap.poll();
}

public void printQueue() {
    for (Element element : maxHeap) {
        System.out.print(element + " ");
    }
    System.out.println();
}

public static void main(String[] args) {
    PriorityQueueUsingHeap pq = new PriorityQueueUsingHeap();
```

```
        System.out.println();
        pq.enqueue(3, 1);
        pq.enqueue(10, 3);
        pq.enqueue(5, 2);
        System.out.print("Input:");
        pq.printQueue();
        System.out.println("Dequeued element: " + pq.dequeue());
        System.out.print("Output:");
        pq.printQueue();

        pq = new PriorityQueueUsingHeap();
        System.out.println();
        pq.enqueue(7, 4);
        pq.enqueue(8, 2);
        pq.enqueue(6, 3);
        System.out.print("Input:");
        pq.printQueue();
        System.out.println("Dequeued element: " + pq.dequeue());
        System.out.print("Output:");
        pq.printQueue();
    }
}
```

Output –



14. Design an algorithm to implement a stack with a getMin() function to return the minimum element in constant time.

- **Test Case 1**:
  Input: Push 5, Push 3, Push 7, Get Min
  Output: Min = 3
- **Test Case 2**:
  Input: Push 10, Push 8, Push 6, Push 12, Get Min
  Output: Min = 6

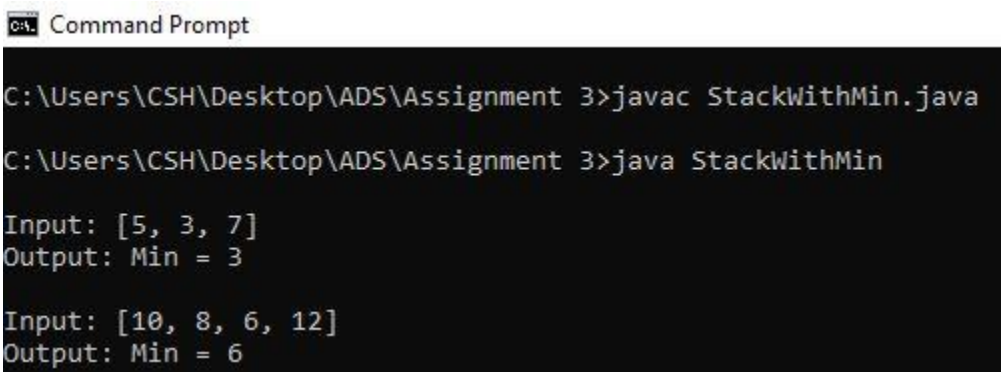
Code –

```
import java.util.Stack;


public class StackWithMin {
   private Stack<Integer> stack;
   private Stack<Integer> minStack;
   public StackWithMin() {
      stack = new Stack<>();
      minStack = new Stack<>();
   }



   public void push(int value) {
      stack.push(value);


      if (minStack.isEmpty() || value <= minStack.peek()) {
         minStack.push(value);
```

```
    }
}

public int pop() {
    if (stack.isEmpty()) {
        throw new IllegalStateException("Stack is empty");
    }
    int popped = stack.pop();

    if (popped == minStack.peek()) {
        minStack.pop();
    }
    return popped;
}

public int getMin() {
    if (minStack.isEmpty()) {
        throw new IllegalStateException("Stack is empty");
    }
    return minStack.peek();
}

public void printStack() {
    System.out.println("Input: " + stack);
}

public void printMinStack() {
    System.out.println("Min Stack: " + minStack);
}
```

```
    public static void main(String[] args) {
        StackWithMin stackWithMin = new StackWithMin();

        System.out.println();
        stackWithMin.push(5);
        stackWithMin.push(3);
        stackWithMin.push(7);
        stackWithMin.printStack();
        System.out.println("Output: " + "Min = " + stackWithMin.getMin());

        System.out.println();
        stackWithMin = new StackWithMin();
        stackWithMin.push(10);
        stackWithMin.push(8);
        stackWithMin.push(6);
        stackWithMin.push(12);
        stackWithMin.printStack();
        System.out.println("Output: " + "Min = " + stackWithMin.getMin());
    }
}
```

Output –



15. Design a Circular Queue with a fixed size, supporting enqueue, dequeue, and isFull/isEmpty operations.

- **Test Case 1**:
  Input: Size = 4, Enqueue 1, 2, 3, 4, isFull()
  Output: True
- **Test Case 2**:
  Input: Size = 3, Enqueue 5, 6, Dequeue, Enqueue 7, isEmpty()
  Output: False

Code –

```
public class CirQueueAlgo {

   private int[] queue;

   private int front, rear, size, count;


   public CirQueueAlgo(int size) {

      this.size = size;

      queue = new int[size];

      front = 0;

      rear = 0;

      count = 0;

   }


   public boolean enqueue(int value) {

      if (isFull()) {

         System.out.println("Queue is full. Cannot enqueue " + value);

         return false;

      }

      queue[rear] = value;

      rear = (rear + 1) % size;

      count++;

      return true;

   }
```

```
public boolean dequeue() {
    if (isEmpty()) {
        System.out.println("Queue is empty. Cannot dequeue.");
        return false;
    }
    int removedElement = queue[front];
    front = (front + 1) % size;
    count--;
    System.out.println("Dequeued element: " + removedElement);
    return true;
}

public boolean isFull() {
    return count == size;
}

public boolean isEmpty() {
    return count == 0;
}

public void printQueue() {
    System.out.print("Input: ");
    for (int i = 0; i < count; i++) {
        System.out.print(queue[(front + i) % size] + " ");
    }
    System.out.println();
}

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

```
        System.out.println();
        CirQueueAlgo cq1 = new CirQueueAlgo(4);
        cq1.enqueue(1);
        cq1.enqueue(2);
        cq1.enqueue(3);
        cq1.enqueue(4);
        cq1.printQueue();
        System.out.println("Output: " + cq1.isFull());
       System.out.println();
        CirQueueAlgo cq2 = new CirQueueAlgo(3);
        cq2.enqueue(5);
        cq2.enqueue(6);
        cq2.printQueue();
        cq2.dequeue();
        cq2.enqueue(7);
        cq2.printQueue();
        System.out.println("Output: " + cq2.isEmpty());
    }
}
```

Output –