**5 C-DAC Mumbai Date 29/09/2024**

**Subject: Algorithm and Data Structure**

**Assignment 3**

**Solve the assignment with following thing to be added in each question.**

-Program

-Flow chart

-Explanation

-Output

-Time and Space complexity

Submission Date: 01/10/2024

**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

**Program:**

class stack{

static final int MAX= 50;

int top;

int a[] = new int[MAX]; //max size

boolean isEmpty()

{

return (top<0);

}

stack()

{

top = -1;

}

boolean push(int x)

{

if(top>=(MAX -1)){

System.out.println("stack overflow");

return false;

}

else{

a[++top] = x;

System.out.println( x +" push into the stack");

return true;

}

}

int pop()

{

if(top<=0){

System.out.println("stack underflow");

return 0;

}

else{

int x = a[top--];

return x;

}

}

void print(){

for(int i = top; i>-1; i--){

System.out.println(" " + a[i]);

}

}

}

class Ass3prg1{

public static void main(String args[])

{

//testcase1

stack s = new stack();

s.push(5);

s.push(3);

s.push(7);

System.out.println(s.pop() + " pop from stack" );

s.print();

//testcase1

s.push(10);

s.push(20);

System.out.println(s.pop() + " pop from stack" );

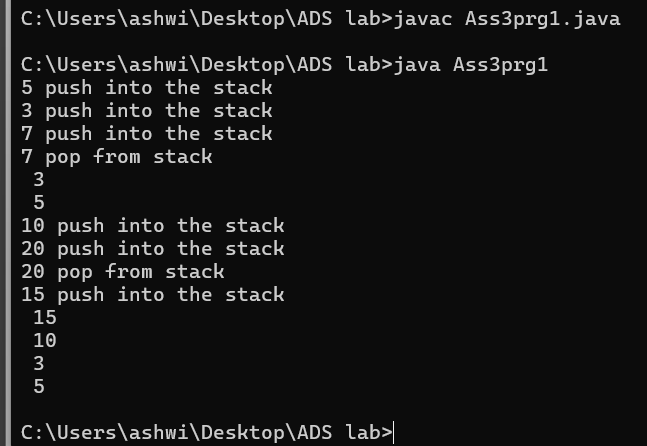
s.push(15);

s.print();

}

}

**Output:**



**2. Check for balanced parentheses using a stack.**

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

**Program**:

import java.util.Scanner;

import java.util.Stack;

public class Ass3prg2 {

public static boolean ispar(String s) {

Stack<Character> stk = new Stack<>();

for (int i = 0; i < s.length(); i++) {

char current = s.charAt(i);

if (current == '(' || current == '{' || current == '[') {

stk.push(current);

} else {

if (!stk.empty() &&

((stk.peek() == '(' && current == ')') ||

(stk.peek() == '{' && current == '}') ||

(stk.peek() == '[' && current == ']'))) {

stk.pop();

} else {

return false;

}

}

}

return stk.empty();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in); // Create Scanner object

System.out.println("Enter the string to check if it's balanced: ");

String s = scanner.nextLine(); // Take input from the user

if (ispar(s)) {

System.out.println("Balanced");

} else {

System.out.println("Not Balanced");

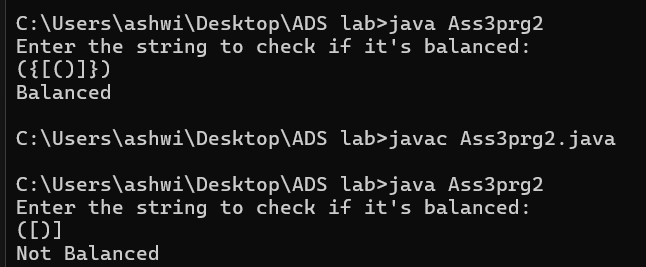
}

scanner.close(); // Close the scanner

}

}

**Output:**



**3. Reverse a string using a stack.**

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

**Program:**

import java.util.Stack;

import java.util.\*;

public class Ass3prg3{

public static void main(String[]args){

Scanner sc = new Scanner(System.in);

System.out.println("enter a string");

String str;

str = sc.nextLine();

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

for(int i = 0; i<str.length(); i++){

stack.push(str.charAt(i));

}

StringBuilder reversedString = new StringBuilder();

while(!stack.isEmpty()){

reversedString.append(stack.pop());

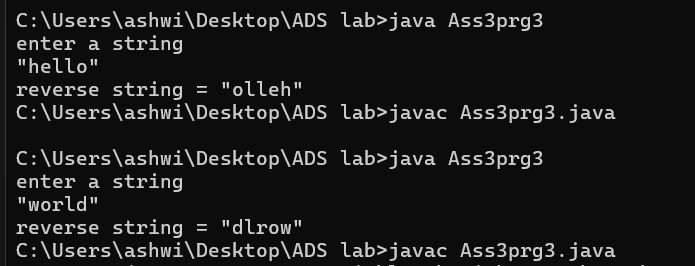
}

System.out.println("reversedString" + reversedString.toString());

}

}

**Output:**



**Complexity:**

Time Complexity: O(n)

Space Complexity: O(n)

**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

**Program:**

import java.util.Stack;

public class Ass3prg4 {

static int evaluatePostfix(String exp) {

Stack<Integer> stack = new Stack<>();

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

for (String token : tokens) {

if (Character.isDigit(token.charAt(0))) {

stack.push(Integer.parseInt(token));

} else {

int val1 = stack.pop();

int val2 = stack.pop();

switch (token) {

case "+":

stack.push(val2 + val1);

break;

case "-":

stack.push(val2 - val1);

break;

case "/":

stack.push(val2 / val1);

break;

case "\*":

stack.push(val2 \* val1);

break;

}

}

}

return stack.pop();

}

public static void main(String[] args) {

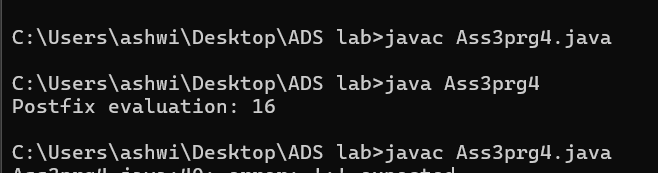
String exp = "5 3 + 2 \*";

System.out.println("Postfix evaluation: " + evaluatePostfix(exp));

}

}

**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 / +"

**Program:**

import java.util.Stack;

public class Ass3prg5{

public static int precedence(char ch) {

switch (ch) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

default:

return -1;

}

}

public static String infixToPostfix(String exp) {

StringBuilder result = new StringBuilder();

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

for (int i = 0; i < exp.length(); i++) {

char c = exp.charAt(i);

if (Character.isLetterOrDigit(c)) {

result.append(c);

result.append(' ');

}

else if (c == '(') {

stack.push(c);

}

else if (c == ')') {

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

result.append(stack.pop()).append(' ');

}

stack.pop(); // Remove '(' from the stack

}

else if (c == '+' || c == '-' || c == '\*' || c == '/') {

while (!stack.isEmpty() && precedence(c) <= precedence(stack.peek())) {

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) {

// Test Case 1

String infixExp1 = "A + B \* C";

System.out.println("Infix: " + infixExp1);

System.out.println("Postfix: " + infixToPostfix(infixExp1));

// Test Case 2

String infixExp2 = "A \* B + C / D";

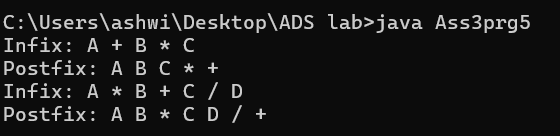
System.out.println("Infix: " + infixExp2);

System.out.println("Postfix: " + infixToPostfix(infixExp2));

}

}

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

**Program**:

class Ass3prg6 {

int MAX = 5;

int front = 0;

int rear = -1;

int size = 0;

int[] a = new int[MAX];

private boolean isFULL() {

return size == MAX;

}

public void insert(int val) {

if (isFULL()) {

System.out.println("Queue is full");

return;

}

rear = (rear + 1) % MAX;

a[rear] = val;

size++;

}

public boolean ifEmpty() {

return size == 0;

}

public int remove() {

if (ifEmpty()) {

System.out.println("Queue is empty");

return Integer.MIN\_VALUE;

}

int dequeuedValue = a[front];

front = (front + 1) % MAX;

size--;

return dequeuedValue;

}

public int getSize() {

return size;

}

public int getFront() {

if (ifEmpty()) {

System.out.println("Queue is empty");

return Integer.MIN\_VALUE;

}

return a[front];

}

public int getRear() {

if (ifEmpty()) {

System.out.println("Queue is empty");

return Integer.MIN\_VALUE;

}

return a[rear];

}

public void displayQueue() {

if (ifEmpty()) {

System.out.println("Queue is empty");

return;

}

System.out.print("Queue = [");

for (int i = 0; i < size; i++) {

System.out.print(a[(front + i) % MAX]);

if (i < size - 1) {

System.out.print(", ");

}

}

System.out.println("]");

}

public static void main(String[] args) {

Ass3prg6 queue = new Ass3prg6();

// Test Case 1

System.out.println("Test Case 1:");

queue.insert(5);

queue.insert(10);

queue.displayQueue();

int dequeued1 = queue.remove();

System.out.println("Dequeued element = " + dequeued1);

queue.displayQueue();

// Test Case 2

System.out.println("\nTest Case 2:");

queue.insert(1);

queue.insert(2);

queue.insert(3);

queue.displayQueue();

int dequeued2 = queue.remove();

System.out.println("Dequeued element = " + dequeued2);

int dequeued3 = queue.remove();

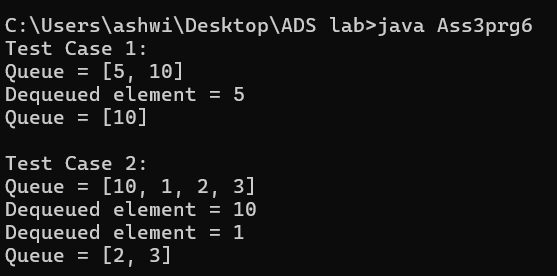
System.out.println("Dequeued element = " + dequeued3);

queue.displayQueue();

}

}

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]

**Program:**

public class Ass3prg7 {

int SIZE;

int front, rear;

int items[];

Ass3prg7(int size) {

front = -1;

rear = -1;

this.SIZE = size;

this.items = new int[size];

}

boolean isFull() {

if (front == 0 && rear == SIZE - 1) {

return true;

}

if (front == rear + 1) {

return true;

}

return false;

}

boolean isEmpty() {

return front == -1;

}

void enQueue(int element) {

if (isFull()) {

System.out.println("Queue is full");

} else {

if (front == -1) {

front = 0;

}

rear = (rear + 1) % SIZE;

items[rear] = element;

System.out.println("Enqueued: " + element);

}

}

int deQueue() {

if (isEmpty()) {

System.out.println("Queue is empty");

return -1;

} else {

int element = items[front];

if (front == rear) {

front = -1;

rear = -1;

} else {

front = (front + 1) % SIZE;

}

System.out.println("Dequeued: " + element);

return element;

}

}

void display() {

if (isEmpty()) {

System.out.println("Queue is empty");

} else {

System.out.print("Queue: ");

int i;

for (i = front; i != rear; i = (i + 1) % SIZE) {

System.out.print(items[i] + " ");

}

System.out.println(items[i]);

}

}

public static void main(String[] args) {

Ass3prg7 queue = new Ass3prg7(5);

// Test Case 1

System.out.println("Test Case 1:");

queue.enQueue(4);

queue.enQueue(5);

queue.enQueue(6);

queue.enQueue(7);

queue.deQueue();

queue.enQueue(8);

queue.display();

// Test Case 2

System.out.println("\nTest Case 2:");

queue = new Ass3prg7(5);

queue.enQueue(1);

queue.enQueue(2);

queue.enQueue(3);

queue.enQueue(4);

queue.deQueue();

queue.deQueue();

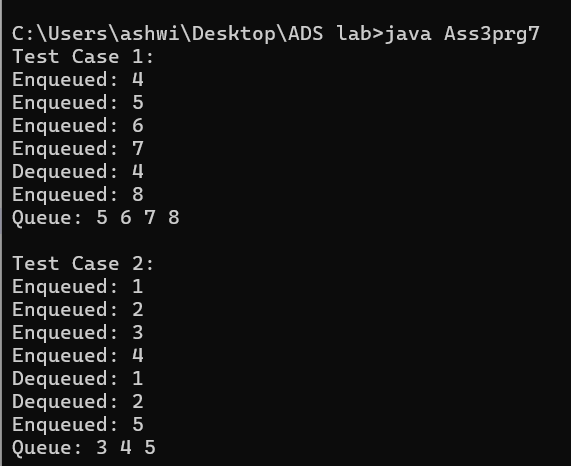
queue.enQueue(5);

queue.display();

}

}

Output:



**8. Implement a Queue using two Stacks.**

* 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

Program:

import java.util.Stack;

class Ass3prg8 {

Stack<Integer> stack1;

Stack<Integer> stack2;

public Ass3prg8() {

stack1 = new Stack<>();

stack2 = new Stack<>();

}

public void enqueue(int data) {

stack1.push(data);

System.out.println("Enqueued: " + data);

}

public int dequeue() {

if (stack1.isEmpty() && stack2.isEmpty()) {

System.out.println("Queue is empty.");

return -1;

}

if (stack2.isEmpty()) {

while (!stack1.isEmpty()) {

stack2.push(stack1.pop());

}

}

int dequeued = stack2.pop();

System.out.println("Dequeued: " + dequeued);

return dequeued;

}

public void displayQueue() {

if (stack1.isEmpty() && stack2.isEmpty()) {

System.out.println("Queue is empty.");

} else {

System.out.print("Queue = [");

for (int i = stack2.size() - 1; i >= 0; i--) {

System.out.print(stack2.get(i));

if (i != 0 || !stack1.isEmpty()) {

System.out.print(", ");

}

}

for (int i = 0; i < stack1.size(); i++) {

System.out.print(stack1.get(i));

if (i != stack1.size() - 1) {

System.out.print(", ");

}

}

System.out.println("]");

}

}

public static void main(String[] args) {

Ass3prg8 queue = new Ass3prg8();

// Test Case 1

queue.enqueue(3);

queue.enqueue(7);

queue.displayQueue();

queue.dequeue();

queue.displayQueue();

// Test Case 2

queue.enqueue(10);

queue.enqueue(20);

queue.displayQueue();

queue.dequeue();

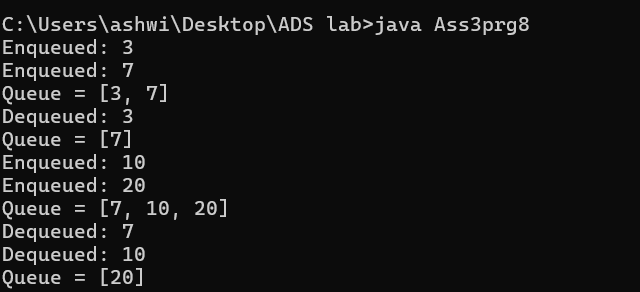
queue.dequeue();

queue.displayQueue();

}

}

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

**Program:**

import java.util.ArrayList;

class Ass3prg9 {

private ArrayList<Integer> heap;

public Ass3prg9() {

heap = new ArrayList<>();

}

public void insert(int value) {

heap.add(value);

int currentIndex = heap.size() - 1;

bubbleUp(currentIndex);

System.out.println("Inserted: " + value);

}

public int extractMin() {

if (heap.size() == 0) {

System.out.println("Heap is empty.");

return -1;

}

int minValue = heap.get(0);

heap.set(0, heap.remove(heap.size() - 1));

bubbleDown(0);

System.out.println("Extracted Min: " + minValue);

return minValue;

}

private void bubbleUp(int index) {

while (index > 0) {

int parentIndex = (index - 1) / 2;

if (heap.get(index) < heap.get(parentIndex)) {

swap(index, parentIndex);

index = parentIndex;

} else {

break;

}

}

}

private void bubbleDown(int index) {

int leftChild, rightChild, smallest;

while (index < heap.size()) {

leftChild = 2 \* index + 1;

rightChild = 2 \* index + 2;

smallest = index;

if (leftChild < heap.size() && heap.get(leftChild) < heap.get(smallest)) {

smallest = leftChild;

}

if (rightChild < heap.size() && heap.get(rightChild) < heap.get(smallest)) {

smallest = rightChild;

}

if (smallest != index) {

swap(index, smallest);

index = smallest;

} else {

break;

}

}

}

private void swap(int i, int j) {

int temp = heap.get(i);

heap.set(i, heap.get(j));

heap.set(j, temp);

}

public void displayHeap() {

System.out.println("Min-Heap = " + heap);

}

public static void main(String[] args) {

Ass3prg9 minHeap = new Ass3prg9();

// Test Case 1

System.out.println("Test Case 1:");

minHeap.insert(10);

minHeap.insert(15);

minHeap.insert(20);

minHeap.insert(17);

minHeap.displayHeap();

minHeap.extractMin();

minHeap.displayHeap();

// Test Case 2

System.out.println("\nTest Case 2:");

minHeap.insert(30);

minHeap.insert(40);

minHeap.insert(20);

minHeap.insert(50);

minHeap.displayHeap();

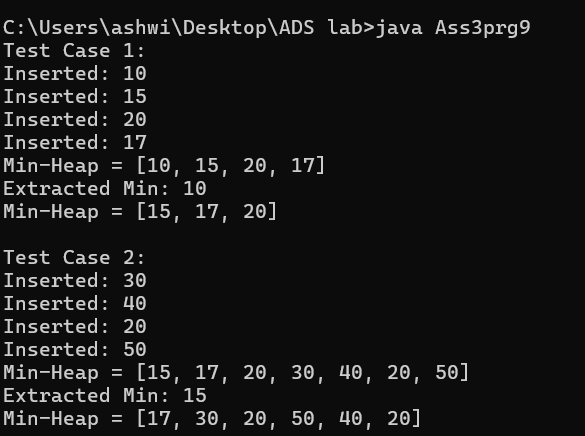
minHeap.extractMin();

minHeap.displayHeap();

}

}

**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

**Program:**

import java.util.ArrayList;

class Ass3prg10 {

private ArrayList<Integer> heap;

heap = new ArrayList<>();

}

public void insert(int value) {

heap.add(value);

int currentIndex = heap.size() - 1;

bubbleUp(currentIndex);

System.out.println("Inserted: " + value);

}

public int extractMax() {

if (heap.size() == 0) {

System.out.println("Heap is empty.");

return -1;

}

int maxValue = heap.get(0);

heap.set(0, heap.remove(heap.size() - 1));

bubbleDown(0);

System.out.println("Extracted Max: " + maxValue);

return maxValue;

}

private void bubbleUp(int index) {

while (index > 0) {

int parentIndex = (index - 1) / 2;

if (heap.get(index) > heap.get(parentIndex)) {

swap(index, parentIndex);

index = parentIndex;

} else {

break;

}

}

}

private void bubbleDown(int index) {

int leftChild, rightChild, largest;

while (index < heap.size()) {

leftChild = 2 \* index + 1;

rightChild = 2 \* index + 2;

largest = index;

if (leftChild < heap.size() && heap.get(leftChild) > heap.get(largest)) {

largest = leftChild;

}

if (rightChild < heap.size() && heap.get(rightChild) > heap.get(largest)) {

largest = rightChild;

}

if (largest != index) {

swap(index, largest);

index = largest;

} else {

break;

}

}

}

private void swap(int i, int j) {

int temp = heap.get(i);

heap.set(i, heap.get(j));

heap.set(j, temp);

}

public void displayHeap() {

System.out.println("Max-Heap = " + heap);

}

public static void main(String[] args) {

Ass3prg10 maxHeap = new Ass3prg10();

// Test Case 1

System.out.println("Test Case 1:");

maxHeap.insert(12);

maxHeap.insert(7);

maxHeap.insert(15);

maxHeap.insert(5);

maxHeap.displayHeap();

maxHeap.extractMax();

maxHeap.displayHeap();

// Test Case 2

System.out.println("\nTest Case 2:");

maxHeap.insert(8);

maxHeap.insert(20);

maxHeap.insert(10);

maxHeap.insert(3);

maxHeap.displayHeap();

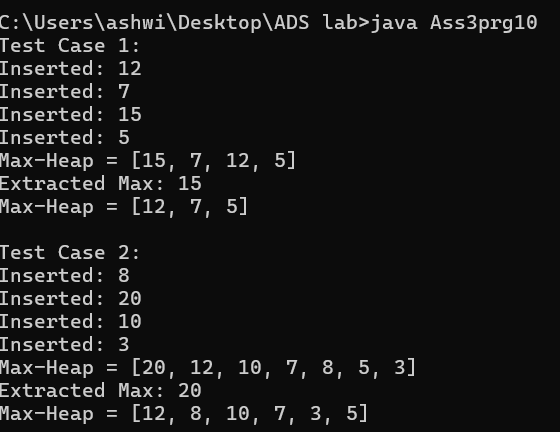
maxHeap.extractMax();

maxHeap.displayHeap();

}

}

Output:



**11. Sort an array 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]

Program:

class Ass3prg11 {

public void heapSort(int[] array) {

int n = array.length;

for (int i = n / 2 - 1; i >= 0; i--) {

heapify(array, n, i);

}

for (int i = n - 1; i > 0; i--) {

swap(array, 0, i);

heapify(array, i, 0);

}

}

private void heapify(int[] array, int n, int i) {

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && array[left] > array[largest]) {

largest = left;

}

if (right < n && array[right] > array[largest]) {

largest = right;

}

if (largest != i) {

swap(array, i, largest);

heapify(array, n, largest);

}

}

private void swap(int[] array, int i, int j) {

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

public void printArray(int[] array) {

for (int i : array) {

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

}

System.out.println();

}

public static void main(String[] args) {

Ass3prg11 heapSort = new Ass3prg11();

int[] array1 = {5, 1, 12, 3, 9};

System.out.println("Test Case 1:");

System.out.println("Input: [5, 1, 12, 3, 9]");

heapSort.heapSort(array1);

System.out.print("Sorted Output: ");

heapSort.printArray(array1);

// Test Case 2

int[] array2 = {20, 15, 8, 10};

System.out.println("\nTest Case 2:");

System.out.println("Input: [20, 15, 8, 10]");

heapSort.heapSort(array2);

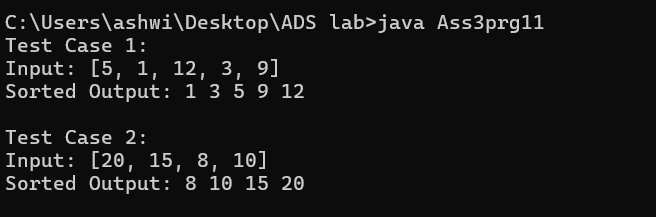
System.out.print("Sorted Output: ");

heapSort.printArray(array2);

}

}

Output:



**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

Program:

import java.util.PriorityQueue;

class Ass3prg12 {

public int findKthLargest(int[] stream, int k) {

PriorityQueue<Integer> minHeap = new PriorityQueue<>();

for (int num : stream) {

minHeap.offer(num);

if (minHeap.size() > k) {

minHeap.poll();

}

}

return minHeap.peek();

}

public static void main(String[] args) {

Ass3prg12 kthLargestFinder = new Ass3prg12();

// Test Case 1

int[] stream1 = {3, 10, 5, 20, 15};

int k1 = 3;

System.out.println("Test Case 1:");

System.out.println("Stream: [3, 10, 5, 20, 15], k = " + k1);

System.out.println("kth Largest Element: " + kthLargestFinder.findKthLargest(stream1, k1));

// Test Case 2

int[] stream2 = {7, 4, 8, 2, 9};

int k2 = 2;

System.out.println("\nTest Case 2:");

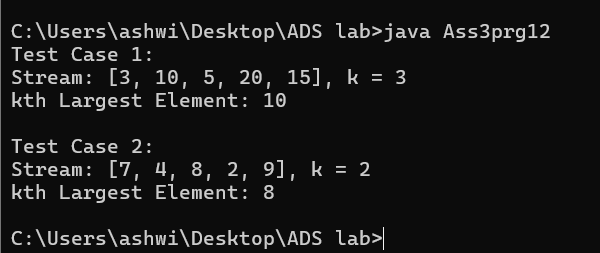
System.out.println("Stream: [7, 4, 8, 2, 9], k = " + k2);

System.out.println("kth Largest Element: " + kthLargestFinder.findKthLargest(stream2, k2));

}

}

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]

**Program:**

import java.util.PriorityQueue;

class Ass3prg13 {

static class Node implements Comparable<Node> {

int value;

int priority;

Node(int value, int priority) {

this.value = value;

this.priority = priority;

}

@Override

public int compareTo(Node other) {

return Integer.compare(other.priority, this.priority);

}

}

private PriorityQueue<Node> minHeap;

public Ass3prg13() {

minHeap = new PriorityQueue<>();

}

public void enqueue(int value, int priority) {

Node newNode = new Node(value, priority);

minHeap.offer(newNode);

System.out.println("Enqueued: " + value + " with priority " + priority);

}

public int dequeue() {

if (minHeap.isEmpty()) {

System.out.println("Priority Queue is empty.");

return -1;

}

Node dequeuedNode = minHeap.poll();

System.out.println("Dequeued element: " + dequeuedNode.value + " (priority " + dequeuedNode.priority + ")");

return dequeuedNode.value;

}

public void displayQueue() {

System.out.print("Priority Queue = [");

for (Node node : minHeap) {

System.out.print(node.value + " (priority " + node.priority + "), ");

}

System.out.println("]");

}

public static void main(String[] args) {

Ass3prg13 priorityQueue = new Ass3prg13();

// Test Case 1

System.out.println("Test Case 1:");

priorityQueue.enqueue(3, 1);

priorityQueue.enqueue(10, 3);

priorityQueue.enqueue(5, 2);

priorityQueue.displayQueue();

priorityQueue.dequeue();

priorityQueue.displayQueue();

// Test Case 2

System.out.println("\nTest Case 2:");

priorityQueue.enqueue(7, 4);

priorityQueue.enqueue(8, 2);

priorityQueue.enqueue(6, 3);

priorityQueue.displayQueue();

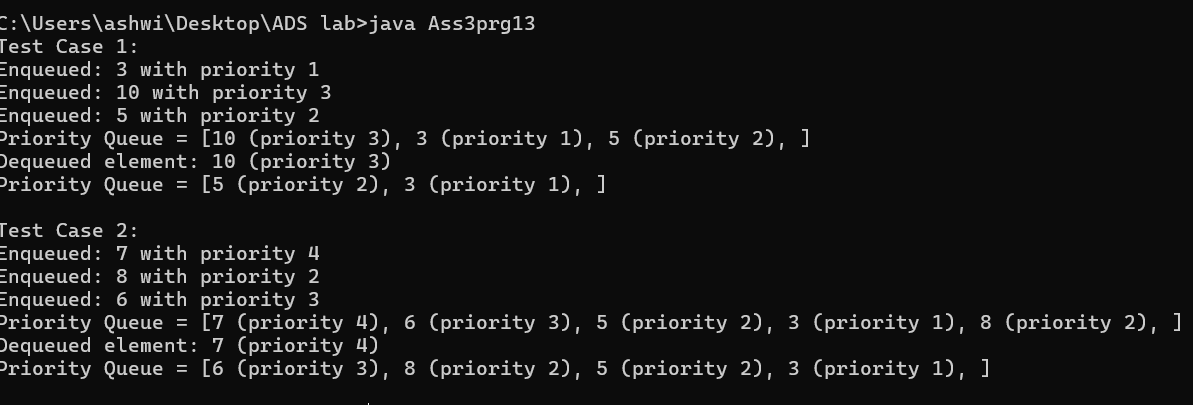
priorityQueue.dequeue();

priorityQueue.displayQueue();

}

}

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

**Program:**

import java.util.Stack;

class Ass3prg14 {

private Stack<Integer> mainStack;

private Stack<Integer> minStack;

public Ass3prg14() {

mainStack = new Stack<>();

minStack = new Stack<>();

}

public void push(int value) {

mainStack.push(value);

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

minStack.push(value);

}

System.out.println("Pushed: " + value);

}

public int pop() {

if (mainStack.isEmpty()) {

System.out.println("Stack is empty.");

return -1; // or throw an exception

}

int poppedValue = mainStack.pop();

if (poppedValue == minStack.peek()) {

minStack.pop();

}

System.out.println("Popped: " + poppedValue);

return poppedValue;

}

public int getMin() {

if (minStack.isEmpty()) {

System.out.println("Stack is empty.");

return -1;

}

return minStack.peek();

}

public static void main(String[] args) {

Ass3prg14 stack = new Ass3prg14();

// Test Case 1

System.out.println("Test Case 1:");

stack.push(5);

stack.push(3);

stack.push(7);

System.out.println("Min = " + stack.getMin());

// Test Case 2

System.out.println("\nTest Case 2:");

stack.push(10);

stack.push(8);

stack.push(6);

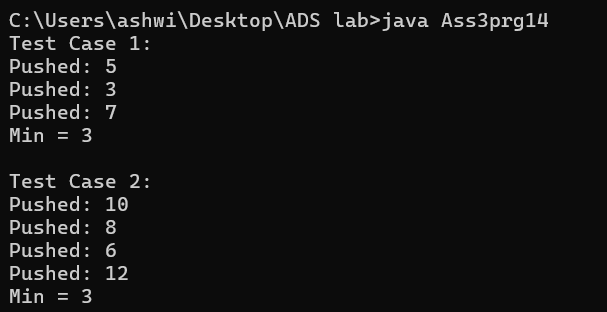
stack.push(12);

System.out.println("Min = " + stack.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

**Program:**

class Ass3prg15 {

private int[] queue;

private int front, rear, size, capacity;

public Ass3prg15(int capacity) {

this.capacity = capacity;

this.size = 0;

this.front = 0;

this.rear = -1;

this.queue = new int[capacity];

}

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++;

System.out.println("Enqueued: " + value);

}

public int dequeue() {

if (isEmpty()) {

System.out.println("Queue is empty. Cannot dequeue.");

return -1;

}

int dequeuedValue = queue[front];

front = (front + 1) % capacity;

size--;

System.out.println("Dequeued: " + dequeuedValue);

return dequeuedValue;

}

public boolean isFull() {

return size == capacity;

}

public boolean isEmpty() {

return size == 0;

}

public static void main(String[] args) {

// Test Case 1

System.out.println("Test Case 1:");

Ass3prg15 queue1 = new Ass3prg15(4);

queue1.enqueue(1);

queue1.enqueue(2);

queue1.enqueue(3);

queue1.enqueue(4);

System.out.println("isFull() = " + queue1.isFull());

// Test Case 2

System.out.println("\nTest Case 2:");

Ass3prg15 queue2 = new Ass3prg15(3);

queue2.enqueue(5);

queue2.enqueue(6);

queue2.dequeue();

queue2.enqueue(7);

System.out.println("isEmpty() = " + queue2.isEmpty());

}

}

**Output:**

