**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 ArrayStack{

static final int max=100;

int top;

int a[] = new int[max];

boolean isEmpty()

{

return(top<0);

}

ArrayStack()

{

top=-1;

}

boolean push(int x)

{

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

System.out.println("Stack Overflow");

return false;

}

else{

a[++top]=x;

return true;

}

}

int pop()

{

if(top<0)

{

System.out.println("Stack Underflow");

return 0;

}

return a[top--];

}

void display(){

System.out.print("[");

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

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

if(i<top){

System.out.print(", ");

}

}

System.out.print("]");

}

}

class Main {

public static void main(String args[])

{

ArrayStack s = new ArrayStack();

s.push(5);

s.push(3);

s.push(7);

System.out.print("Stack =" );

int a=s.pop();

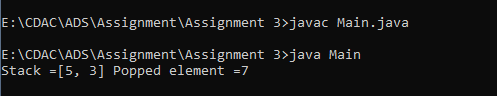
s.display();

System.out.println(" Popped element ="+a);

}

}

Output:



**Explanation:**

1.Initialize an array to represent the stack.

2.Use the end of the array to represent the top of the stack.

3.Implement push (add to end), pop (remove from the end), and Display operations, push operation will handle empty and full stack conditions.

Time Complexity: O(1)

Space Complexity: O(n)

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

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

**Explaination:**

* Initialize an empty stack.
* Iterate i from 0 to length(expression).
* Store the current character in a variable ‘ch’.
* If stack is empty: Push ‘ch’ to the stack
* Else if current character is a closing bracket and of the top of the stack contains an opening bracket of the same type then remove the top of the stack: Else, push ‘ch’ to the stack
* If the stack is empty, return true, else false

Program:

import java.util.\*;

public class test {

public static boolean balancedParenthesis(String str) {

Stack stack = new Stack();

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

char x = str.charAt(i);

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

stack.push(x);

continue;

}

if (stack.isEmpty()) return false;

char check;

switch (x) {

case ')':

check = (char) stack.pop();

if (check == '{' || check == '[') return false;

break;

case '}':

check = (char) stack.pop();

if (check == '(' || check == '[') return false;

break;

case ']':

check = (char) stack.pop();

if (check == '(' || check == '{') return false;

break;

}

}

return (stack.isEmpty());

}

public static void main(String[] args) {

String str = "({[()]})";

if (balancedParenthesis(str))

System.out.println("Balanced");

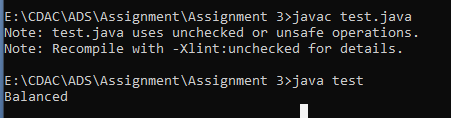
else

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

}

}

Output:



Time complexity: O(n)

Space Complexity: O(n)

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

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

**Explanation:**

**import java.util.\*;**

**class Stack**

**{**

**int size;**

**int top;**

**char[] a;**

**boolean isEmpty()**

**{**

**return (top < 0);**

**}**

**Stack(int n)**

**{**

**top = -1;**

**size = n;**

**a = new char[size];**

**}**

**boolean push(char x)**

**{**

**if (top >= size)**

**{**

**System.out.println("Stack Overflow");**

**return false;**

**}**

**else**

**{**

**a[++top] = x;**

**return true;**

**}**

**}**

**char pop()**

**{**

**if (top < 0)**

**{**

**System.out.println("Stack Underflow");**

**return 0;**

**}**

**else**

**{**

**char x = a[top--];**

**return x;**

**}**

**}**

**}**

**public class revStack**

**{**

**public static String reverseString(String str)**

**{**

**String reversedString = "";**

**int lenghOfString = str.length();**

**Stack stack = new Stack(lengthOfString);**

**for (int i = 0; i < lengthOfString; i++) {**

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

**}**

**for (int i = 0; i < lengthOfString; i++)**

**{**

**char ch = stack.pop();**

**reversedString = reversedString+ ch;**

**}**

**return reversedString;**

**}**

**public static void main(String args[])**

**{**

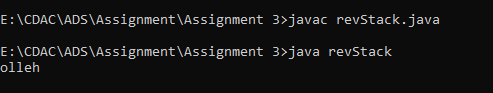
**String s= new String("hello");**

**String result = reverseString(s);**

**System.out.println( + result);**

**}**

**}**



**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

public class postfix {

static int MAX\_SIZE = 100;

int[] stack;

int top;

public postfix() {

stack = new int[MAX\_SIZE];

top = -1;

}

boolean isEmpty() {

return top == -1;

}

void push(int value) {

if (top < MAX\_SIZE - 1) {

stack[++top] = value;

} else {

System.out.println("Stack Overflow");

}

}

int pop() {

if (!isEmpty()) {

return stack[top--];

} else {

System.out.println("Stack Underflow");

return -1;

}

}

public int evaluatePostfix(String expression) {

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

for (String token : tokens) {

if (isNumeric(token)) {

push(Integer.parseInt(token));

} else {

int b = pop();

int a = pop();

int result = performOperation(a, b, token);

push(result);

}

}

return pop();

}

boolean isNumeric(String str) {

try {

Integer.parseInt(str);

return true;

} catch (NumberFormatException e) {

return false;

}

}

int performOperation(int a, int b, String operator) {

switch (operator) {

case "+":

return a + b;

case "-":

return a - b;

case "\*":

return a \* b;

case "/":

return a / b;

default:

return 0;

}

}

public static void main(String[] args) {

postfix p = new postfix();

String testCase1 = "5 3 + 2 \*";

int result1 = p.evaluatePostfix(testCase1);

System.out.println("Output for Test Case 1: " + result1);

String testCase2 = "4 5 \* 6 /";

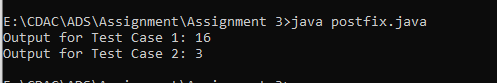
int result2 = p.evaluatePostfix(testCase2);

System.out.println("Output for Test Case 2: " + result2);

}

}

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 / +"

class Stack {

int max = 100;

char[] array = new char[max];

int top = -1;

void push(char c) {

if (top < max - 1) {

array[++top] = c;

} else {

System.out.println("Stack Overflow");

}

}

char pop() {

if (top >= 0) {

return array[top--];

} else {

System.out.println("Stack Underflow");

return '\0';

}

}

char peek() {

if (top >= 0) {

return array[top];

} else {

return '\0';

}

}

boolean isEmpty() {

return top == -1;

}

}

public class InfixToPostfix {

public static int precedence(char op) {

switch (op) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

default:

return 0;

}

}

public static String convert(String infix) {

Stack stack = new Stack();

StringBuilder postfix = new StringBuilder();

for (char c : infix.toCharArray()) {

if (Character.isLetter(c)) {

postfix.append(c).append(" ");

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

stack.push(c);

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

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

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

}

stack.pop();

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

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

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

}

stack.push(c);

}

}

while (!stack.isEmpty()) {

if (stack.peek() == '(') {

stack.pop();

} else {

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

}

}

return postfix.toString().trim();

}

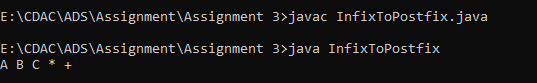
public static void main(String[] args) {

System.out.println(convert("A + B \* C"));

}

}

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

class Queue {

int max = 100;

int[] array = new int[max];

int front = 0;

int rear = -1;

int size = 0;

void enqueue(int element) {

if (size < max) {

array[++rear] = element;

size++;

} else {

System.out.println("Queue Overflow");

}

}

int dequeue() {

if (size > 0) {

int temp = array[front];

for (int i = 0; i < size - 1; i++) {

array[i] = array[i + 1];

}

size--;

rear--;

return temp;

} else {

System.out.println("Queue Underflow");

return -1;

}

}

void display() {

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

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

System.out.print(array[i]);

if (i < size - 1) {

System.out.print(", ");

}

}

System.out.println("]");

}

}

public class arrayQueue {

public static void main(String[] args) {

Queue queue = new Queue();

queue.enqueue(5);

queue.enqueue(10);

System.out.print("Dequeued element = ");

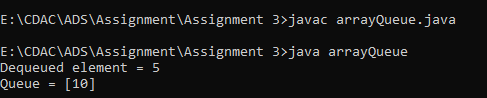
System.out.println(queue.dequeue());

queue.display();

}

}

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]

class CircularQueue {

int max = 5;

int[] array = new int[max];

int front = 0;

int rear = -1;

int size = 0;

void enqueue(int element) {

if (isFull()) {

System.out.println("Queue Overflow");

} else {

rear = (rear + 1) % max;

array[rear] = element;

size++;

}

}

int dequeue() {

if (isEmpty()) {

System.out.println("Queue Underflow");

return -1;

} else {

int temp = array[front];

front = (front + 1) % max;

size--;

return temp;

}

}

boolean isFull() {

return size == max;

}

boolean isEmpty() {

return size == 0;

}

void display() {

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

int temp = front;

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

System.out.print(array[temp]);

temp = (temp + 1) % max;

if (i < size - 1) {

System.out.print(", ");

}

}

System.out.println("]");

}

}

public class cQueue {

public static void main(String[] args) {

CircularQueue queue = new CircularQueue();

queue.enqueue(4);

queue.enqueue(5);

queue.enqueue(6);

queue.enqueue(7);

queue.dequeue();

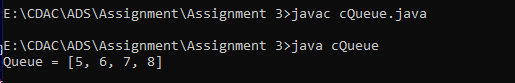
queue.enqueue(8);

queue.display();

}

}

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

Program:

class Stack {

int max = 100;

int[] array = new int[max];

int top = -1;

void push(int element) {

if (top < max - 1) {

array[++top] = element;

} else {

System.out.println("Stack Overflow");

}

}

int pop() {

if (top >= 0) {

return array[top--];

} else {

System.out.println("Stack Underflow");

return -1;

}

}

boolean isEmpty() {

return top == -1;

}

}

class Queue {

Stack stack1;

Stack stack2;

Queue() {

stack1 = new Stack();

stack2 = new Stack();

}

void enqueue(int element) {

stack1.push(element);

}

int dequeue() {

if (stack2.isEmpty()) {

while (!stack1.isEmpty()) {

stack2.push(stack1.pop());

}

}

return stack2.pop();

}

void display() {

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

if (!stack2.isEmpty()) {

int[] temp = new int[stack2.top + 1];

int index = stack2.top;

while (!stack2.isEmpty()) {

temp[index--] = stack2.pop();

}

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

System.out.print(temp[i]);

if (i < temp.length - 1) {

System.out.print(", ");

}

}

} else {

int[] temp = new int[stack1.top + 1];

int index = stack1.top;

while (!stack1.isEmpty()) {

temp[index--] = stack1.pop();

}

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

System.out.print(temp[i]);

if (i < temp.length - 1) {

System.out.print(", ");

}

stack1.push(temp[i]);

}

}

System.out.println("]");

}

}

public class queueTwoStack {

public static void main(String[] args) {

Queue queue = new Queue();

queue.enqueue(3);

queue.enqueue(7);

System.out.print("Dequeued element = ");

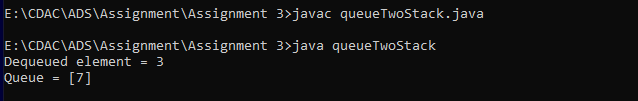
System.out.println(queue.dequeue());

queue.display();

}

}

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

class MinHeap {

int[] heap;

int size;

int capacity;

MinHeap(int capacity) {

this.capacity = capacity;

this.heap = new int[capacity + 1];

this.size = 0;

}

void insert(int value) {

if (size == capacity) {

System.out.println("Heap Overflow");

return;

}

heap[++size] = value;

int index = size;

while (index > 1 && heap[index] < heap[parent(index)]) {

swap(index, parent(index));

index = parent(index);

}

}

int extractMin() {

if (size == 0) {

System.out.println("Heap Underflow");

return -1;

}

int min = heap[1];

heap[1] = heap[size--];

heapify(1);

return min;

}

void heapify(int index) {

int smallest = index;

int left = leftChild(index);

int right = rightChild(index);

if (left <= size && heap[left] < heap[smallest]) {

smallest = left;

}

if (right <= size && heap[right] < heap[smallest]) {

smallest = right;

}

if (smallest != index) {

swap(index, smallest);

heapify(smallest);

}

}

int parent(int index) {

return index / 2;

}

int leftChild(int index) {

return 2 \* index;

}

int rightChild(int index) {

return 2 \* index + 1;

}

void swap(int i, int j) {

int temp = heap[i];

heap[i] = heap[j];

heap[j] = temp;

}

void display() {

System.out.print("Min-Heap = [");

for (int i = 1; i <= size; i++) {

System.out.print(heap[i]);

if (i < size) {

System.out.print(", ");

}

}

System.out.println("]");

}

}

public class Heap1 {

public static void main(String[] args) {

MinHeap minHeap = new MinHeap(10);

minHeap.insert(10);

minHeap.insert(15);

minHeap.insert(20);

minHeap.insert(17);

System.out.print("Extracted Min = ");

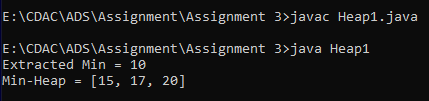
System.out.println(minHeap.extractMin());

minHeap.display();

}

}

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:

class MaxHeap {

int[] heap;

int size;

int capacity;

MaxHeap(int capacity) {

this.capacity = capacity;

this.heap = new int[capacity + 1];

this.size = 0;

}

void insert(int value) {

if (size == capacity) {

System.out.println("Heap Overflow");

return;

}

heap[++size] = value;

int index = size;

while (index > 1 && heap[index] > heap[parent(index)]) {

swap(index, parent(index));

index = parent(index);

}

}

int extractMax() {

if (size == 0) {

System.out.println("Heap Underflow");

return -1;

}

int max = heap[1];

heap[1] = heap[size--];

heapify(1);

return max;

}

void heapify(int index) {

int largest = index;

int left = leftChild(index);

int right = rightChild(index);

if (left <= size && heap[left] > heap[largest]) {

largest = left;

}

if (right <= size && heap[right] > heap[largest]) {

largest = right;

}

if (largest != index) {

swap(index, largest);

heapify(largest);

}

}

int parent(int index) {

return index / 2;

}

int leftChild(int index) {

return 2 \* index;

}

int rightChild(int index) {

return 2 \* index + 1;

}

void swap(int i, int j) {

int temp = heap[i];

heap[i] = heap[j];

heap[j] = temp;

}

void display() {

System.out.print("Max-Heap = [");

for (int i = 1; i <= size; i++) {

System.out.print(heap[i]);

if (i < size) {

System.out.print(", ");

}

}

System.out.println("]");

}

}

public class Heap2 {

public static void main(String[] args) {

MaxHeap maxHeap = new MaxHeap(10);

maxHeap.insert(12);

maxHeap.insert(7);

maxHeap.insert(15);

maxHeap.insert(5);

System.out.print("Extracted Max = ");

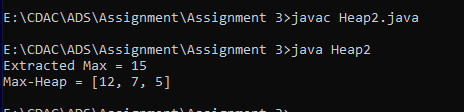
System.out.println(maxHeap.extractMax());

maxHeap.display();

}

}

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]

class HeapSort {

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);

}

}

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

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

void sort(int[] array) {

int n = array.length;

// Build max heap

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

heapify(array, n, i);

}

// Extract elements one by one

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

swap(array, 0, i);

heapify(array, i, 0);

}

}

void display(int[] array) {

System.out.print("[");

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

System.out.print(array[i]);

if (i < array.length - 1) {

System.out.print(", ");

}

}

System.out.println("]");

}

}

public class hSort {

public static void main(String[] args) {

HeapSort heapSort = new HeapSort();

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

System.out.println("Original array:");

heapSort.display(array);

heapSort.sort(array);

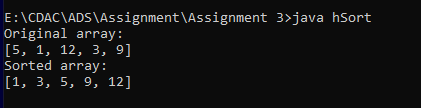
System.out.println("Sorted array:");

heapSort.display(array);

}

}

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:

class MinHeap {

int[] heap;

int size;

int capacity;

MinHeap(int capacity) {

this.capacity = capacity;

this.heap = new int[capacity + 1];

this.size = 0;

}

void insert(int num) {

if (size == capacity) {

return;

}

heap[++size] = num;

int index = size;

while (index > 1 && heap[index] < heap[parent(index)]) {

swap(index, parent(index));

index = parent(index);

}

}

void deleteMin() {

if (size == 0) {

return;

}

heap[1] = heap[size--];

heapify(1);

}

int peek() {

return heap[1];

}

int parent(int index) {

return index / 2;

}

void heapify(int index) {

int smallest = index;

int left = leftChild(index);

int right = rightChild(index);

if (left <= size && heap[left] < heap[smallest]) {

smallest = left;

}

if (right <= size && heap[right] < heap[smallest]) {

smallest = right;

}

if (smallest != index) {

swap(index, smallest);

heapify(smallest);

}

}

int leftChild(int index) {

return 2 \* index;

}

int rightChild(int index) {

return 2 \* index + 1;

}

void swap(int i, int j) {

int temp = heap[i];

heap[i] = heap[j];

heap[j] = temp;

}

}

class KthLargest {

MinHeap minHeap;

int k;

KthLargest(int k) {

this.k = k;

this.minHeap = new MinHeap(k);

}

void add(int num) {

if (minHeap.size < k) {

minHeap.insert(num);

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

minHeap.deleteMin();

minHeap.insert(num);

}

}

int getKthLargest() {

return minHeap.peek();

}

}

public class kthHeap {

public static void main(String[] args) {

KthLargest kthLargest = new KthLargest(3);

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

for (int num : stream) {

kthLargest.add(num);

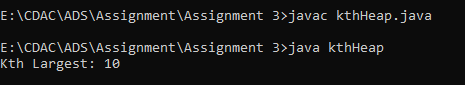
}

System.out.println("Kth Largest: " + kthLargest.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]

Program:

class PriorityQueue {

int[] heap;

int size;

int capacity;

PriorityQueue(int capacity) {

this.capacity = capacity;

this.heap = new int[capacity + 1];

this.size = 0;

}

void enqueue(int value, int priority) {

if (size == capacity) {

System.out.println("Priority Queue Overflow");

return;

}

heap[++size] = value;

int index = size;

while (index > 1 && getPriority(index) > getPriority(parent(index))) {

swap(index, parent(index));

index = parent(index);

}

}

int dequeue() {

if (size == 0) {

System.out.println("Priority Queue Underflow");

return -1;

}

int value = heap[1];

heap[1] = heap[size--];

heapify(1);

return value;

}

int getPriority(int index) {

// Assuming priority is stored along with value in the heap array

// Here, we're using a simple representation where higher value means higher priority

return heap[index];

}

int parent(int index) {

return index / 2;

}

void heapify(int index) {

int largest = index;

int left = leftChild(index);

int right = rightChild(index);

if (left <= size && getPriority(left) > getPriority(largest)) {

largest = left;

}

if (right <= size && getPriority(right) > getPriority(largest)) {

largest = right;

}

if (largest != index) {

swap(index, largest);

heapify(largest);

}

}

int leftChild(int index) {

return 2 \* index;

}

int rightChild(int index) {

return 2 \* index + 1;

}

void swap(int i, int j) {

int temp = heap[i];

heap[i] = heap[j];

heap[j] = temp;

}

void display() {

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

for (int i = 1; i <= size; i++) {

System.out.print(heap[i]);

if (i < size) {

System.out.print(", ");

}

}

System.out.println("]");

}

}

class PriorityElement {

int value;

int priority;

PriorityElement(int value, int priority) {

this.value = value;

this.priority = priority;

}

}

public class pQ {

public static void main(String[] args) {

PriorityQueue priorityQueue = new PriorityQueue(10);

priorityQueue.enqueue(3, 1);

priorityQueue.enqueue(10, 3);

priorityQueue.enqueue(5, 2);

System.out.print("Dequeued element = ");

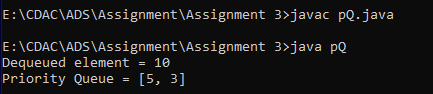
System.out.println(priorityQueue.dequeue());

priorityQueue.display();

}

}

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

class Stack {

int[] stack;

int[] minStack;

int top;

int minTop;

Stack(int size) {

stack = new int[size];

minStack = new int[size];

top = -1;

minTop = -1;

}

void push(int value) {

if (top == stack.length - 1) {

System.out.println("Stack Overflow");

return;

}

stack[++top] = value;

if (minTop == -1 || value <= minStack[minTop]) {

minStack[++minTop] = value;

}

}

int pop() {

if (top == -1) {

System.out.println("Stack Underflow");

return -1;

}

int value = stack[top--];

if (value == minStack[minTop]) {

minTop--;

}

return value;

}

int getMin() {

if (minTop == -1) {

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

return -1;

}

return minStack[minTop];

}

void display() {

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

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

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

}

System.out.println();

}

}

public class getStack {

public static void main(String[] args) {

Stack stack = new Stack(10);

stack.push(5);

stack.push(3);

stack.push(7);

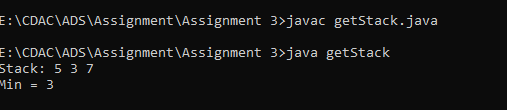
stack.display();

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 CircularQueue {

int[] queue;

int front;

int rear;

int size;

int capacity;

CircularQueue(int capacity) {

this.capacity = capacity;

this.queue = new int[capacity];

this.front = 0;

this.rear = 0;

this.size = 0;

}

void enqueue(int value) {

if (isFull()) {

System.out.println("Queue Overflow");

return;

}

queue[rear] = value;

rear = (rear + 1) % capacity;

size++;

}

int dequeue() {

if (isEmpty()) {

System.out.println("Queue Underflow");

return -1;

}

int value = queue[front];

front = (front + 1) % capacity;

size--;

return value;

}

boolean isFull() {

return size == capacity;

}

boolean isEmpty() {

return size == 0;

}

void display() {

if (isEmpty()) {

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

return;

}

int tempFront = front;

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

System.out.print(queue[tempFront] + " ");

tempFront = (tempFront + 1) % capacity;

}

System.out.println();

}

}

public class cQueue2 {

public static void main(String[] args) {

CircularQueue queue = new CircularQueue(4);

queue.enqueue(1);

queue.enqueue(2);

queue.enqueue(3);

queue.enqueue(4);

System.out.println("Is Full: " + queue.isFull());

queue.display();

}

}

Output:

