# Linear Search

public class Search {

public static int linearSearch(int[] arr, int target) {

for (int i = 0; i < arr.length; i++) { if (arr[i] == target) {

return i;

}

}

return -1;

}

public static void main(String[] args) { int[] arr = { 12, 45, 23, 67, 34, 89 };

int target = 23;

int index = linearSearch(arr, target); if (index != -1) {

System.out.println("Element found at index: " + index);

} else {

System.out.println("Element not found");

}

}

}

# Binary Search

# public class Search {

public static int binarySearch(int[] arr, int target) { int left = 0;

int right = arr.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

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

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

} else {

right = mid - 1;

}

}

return -1;

}

public static void main(String[] args) {

int[] arr = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};

int target = 11;

int index = binarySearch(arr, target); if (index != -1) {

System.out.println("Element found at index: " + index);

} else {

System.out.println("Element not found");

}

}

}

# Sort elements by frequency

import java.util.\*;

public class Search {

// Driver Code

public static void main(String[] args)

{

int[] array = { 4, 4, 2, 2, 2, 2, 3, 3, 1, 1, 6, 7, 5 };

Map<Integer, Integer> map = new HashMap<>(); List<Integer> outputArray = new ArrayList<>();

for (int current : array) {

int count = map.getOrDefault(current, 0);

map.put(current, count + 1); outputArray.add(current);

}

SortComparator comp = new SortComparator(map);

Collections.sort(outputArray, comp);

for (Integer i : outputArray) { System.out.print(i + " ");

}

}

}

class SortComparator implements Comparator<Integer> { private final Map<Integer, Integer> freqMap;

SortComparator(Map<Integer, Integer> tFreqMap)

{

this.freqMap = tFreqMap;

}

@Override

public int compare(Integer k1, Integer k2)

{

int freqCompare = freqMap.get(k2).compareTo(freqMap.get(k1));

int valueCompare = k1.compareTo(k2);

if (freqCompare == 0)

return valueCompare;

else

return freqCompare;

}

}

# Sort an array of 0s, 1s and 2s

import java.util.Arrays;

public class Search {

public static void main(String[] args) { int[] arr = {0, 1, 2, 1, 0, 2, 1, 0};

sort012(arr);

System.out.println("Sorted array: " + Arrays.toString(arr));

}

public static void sort012(int[] arr) { int low = 0;

int mid = 0;

int high = arr.length - 1;

while (mid <= high) { switch (arr[mid]) {

case 0:

swap(arr, low, mid); low++;

mid++; break;

case 1:

mid++; break;

case 2:

swap(arr, mid, high); high--;

break;

}

}

}

private static void swap(int[] arr, int i, int j) { int temp = arr[i];

arr[i] = arr[j]; arr[j] = temp;

}

}

# Java Program to Check for balanced parenthesis by using Stacks

import java.util.\*;

public class Search {

public static boolean checkBalancedParentheses(String str) { Stack<Character> stack = new Stack<>();

for (char ch : str.toCharArray()) {

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

}

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

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

}

}

}

return stack.isEmpty();

}

public static boolean isMatchingPair(char opening, char closing) { return (opening == '(' && closing == ')') ||

(opening == '[' && closing == ']') || (opening == '{' && closing == '}');

}

public static void main(String[] args) { String expression1 = "{[()]}";

String expression2 = "{[(])}";

System.out.println("Expression 1 is balanced: " + checkBalancedParentheses(expression1)); System.out.println("Expression 2 is balanced: " + checkBalancedParentheses(expression2));

}

}

# Java Program to Implement Stack

import java.util.\*;

public class StackExample { static final int MAX = 1000; int top;

int a[] = new int[MAX]; // Maximum size of Stack

boolean isEmpty() { return (top < 0);

}

StackExample() { 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 + " pushed into stack"); return true;

}

}

int pop() {

if (top < 0) {

System.out.println("Stack Underflow"); return 0;

} else {

int x = a[top--]; return x;

}

}

int peek() {

if (top < 0) {

System.out.println("Stack Underflow"); return 0;

} else {

int x = a[top]; return x;

}

}

public static void main(String args[]) { StackExample s = new StackExample(); s.push(10);

s.push(20);

s.push(30);

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

}

}

# Java Program to Implement Queue

import java.util.\*;

public class Search {

private static final int MAX = 1000; private int front, rear, size;

private int[] arr = new int[MAX];

public Search () {

front = 0;

rear = -1;

size = 0;

}

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

}

public boolean isFull() { return (size == MAX);

}

public void enqueue(int item) { if (isFull()) {

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

}

rear = (rear + 1) % MAX; arr[rear] = item;

size++;

System.out.println(item + " enqueued to queue");

}

public int dequeue() { if (isEmpty()) {

System.out.println("Queue is empty"); return -1;

}

int item = arr[front]; front = (front + 1) % MAX; size--;

return item;

}

public int peek() { if (isEmpty()) {

System.out.println("Queue is empty"); return -1;

}

return arr[front];

}

public static void main(String[] args) { Search queue = new Search(); queue.enqueue(10);

queue.enqueue(20); queue.enqueue(30);

System.out.println(queue.dequeue() + " dequeued from queue");

}

}

# Java Program to Implement Dequeue.

import java.util.\*;

public class Search {

private LinkedList<Integer> deque;

public Search() {

deque = new LinkedList<>();

}

public void insertFront(int item) { deque.addFirst(item);

System.out.println(item + " inserted at front");

}

public void insertRear(int item) { deque.addLast(item); System.out.println(item + " inserted at rear");

}

public int deleteFront() { if (deque.isEmpty()) {

System.out.println("Deque is empty"); return -1;

}

int item = deque.removeFirst(); System.out.println("Deleted " + item + " from front"); return item;

}

public int deleteRear() { if (deque.isEmpty()) {

System.out.println("Deque is empty"); return -1;

}

int item = deque.removeLast(); System.out.println("Deleted " + item + " from rear"); return item;

}

public int getFront() {

if (deque.isEmpty()) { System.out.println("Deque is empty");

return -1;

}

return deque.getFirst();

}

public int getRear() {

if (deque.isEmpty()) { System.out.println("Deque is empty"); return -1;

}

return deque.getLast();

}

public static void main(String[] args) { Search deque = new Search(); deque.insertFront(10); deque.insertRear(20); deque.deleteFront(); deque.deleteRear();

}

}

# Java Program to Implement Stack Using Two Queues

import java.util.LinkedList; import java.util.Queue;

public class Search {

Queue<Integer> q1 = new LinkedList<>(); Queue<Integer> q2 = new LinkedList<>(); int top;

public Search() {

}

public void push(int x) { q2.add(x);

top = x;

while (!q1.isEmpty()) {

q2.add(q1.remove());

}

Queue<Integer> temp = q1; q1 = q2;

q2 = temp;

}

public int pop() {

int popped = q1.remove(); if (!q1.isEmpty()) {

top = q1.peek();

}

return popped;

}

public int top() { return top;

}

public boolean empty() { return q1.isEmpty();

}

public static void main(String[] args) { Search stack = new Search(); stack.push(1);

stack.push(2);

System.out.println("Top element: " + stack.top()); System.out.println("Pop element: " + stack.pop()); System.out.println("Is the stack empty? " + stack.empty());

}

}

# Java Program to Implement Queue Using Two Stacks

import java.util.Stack;

public class Search {

Stack<Integer> stack1; Stack<Integer> stack2;

public Search() {

stack1 = new Stack<>(); stack2 = new Stack<>();

}

public void enqueue(int x) { stack1.push(x);

}

public int dequeue() { if (stack2.isEmpty()) {

while (!stack1.isEmpty()) { stack2.push(stack1.pop());

}

}

return stack2.pop();

}

public int peek() {

if (stack2.isEmpty()) {

while (!stack1.isEmpty()) { stack2.push(stack1.pop());

}

}

return stack2.peek();

}

public boolean isEmpty() {

return stack1.isEmpty() && stack2.isEmpty();

}

public int size() {

return stack1.size() + stack2.size();

}

public static void main(String[] args) { Search queue = new Search();

// Enqueue elements queue.enqueue(1); queue.enqueue(2); queue.enqueue(3);

// Dequeue and print elements

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

// Enqueue more elements queue.enqueue(4); queue.enqueue(5);

// Peek and print the front element System.out.println("Front element: " + queue.peek());

// Dequeue remaining elements while (!queue.isEmpty()) {

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

}

}

}