set-1

a) queue using 2 stacks

import java.util.Scanner;

import java.util.Stack;

class DynamicQueueUsingStacks {

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

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

public void enqueue(int data) {

stack1.push(data);

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

}

public int dequeue() {

if (isEmpty()) {

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

return -1;

}

if (stack2.isEmpty()) {

while (!stack1.isEmpty()) {

stack2.push(stack1.pop());

}

}

return stack2.pop();

}

public boolean isEmpty() {

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

}

public void display() {

if (isEmpty()) {

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

return;

}

if (stack2.isEmpty()) {

while (!stack1.isEmpty()) {

stack2.push(stack1.pop());

}

}

System.out.println("Queue contents (front to rear):");

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

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

}

System.out.println();

}

public static void main(String[] args) {

DynamicQueueUsingStacks queue = new DynamicQueueUsingStacks();

Scanner sc = new Scanner(System.in);

int choice, value;

do {

System.out.println("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit");

System.out.print("Enter your choice: ");

choice = sc.nextInt();

switch (choice) {

case 1:

System.out.print("Enter value to enqueue: ");

value = sc.nextInt();

queue.enqueue(value);

break;

case 2:

int removed = queue.dequeue();

if (removed != -1) {

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

}

break;

case 3:

queue.display();

break;

case 4:

System.out.println("Exiting...");

break;

default:

System.out.println("Invalid choice.");

}

} while (choice != 4);

sc.close();

}

}

1b) Minimum element in rotatedsorted array

import java.util.\*;

public class Program1b{

public static int findMin(int[] arr) {

int low = 0, high = arr.length - 1;

while (low < high) {

int mid = low + (high - low) / 2;

if (arr[mid] > arr[high]) {

low = mid + 1;

} else {

high = mid;

}

}

return arr[low];

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of elements: ");

int n = sc.nextInt();

int[] arr = new int[n];

System.out.print("Enter rotated sorted array: ");

for (int i = 0; i < n; i++)

arr[i] = sc.nextInt();

int min = findMin(arr);

System.out.println("Minimum element: " + min);

}

}

set-2

a) remove duplicates

import java.util.\*;

public class Program2b{

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of elements: ");

int n = sc.nextInt();

int[] arr = new int[n];

System.out.print("Enter array elements: ");

for (int i = 0; i < n; i++)

arr[i] = sc.nextInt();

int total = 0;

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

int[] rotated = new int[n];

for (int j = 0; j < n; j++)

rotated[j] = arr[(i + j) % n];

int min = rotated[0];

for (int val : rotated)

min = Math.min(min, val);

for (int val : rotated) {

if (val > min) {

System.out.println("(" + val + "," + min + ")");

total++;

}

}

}

System.out.println("Total - " + total);

}

}

2b)

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

class Program2b{

Queue<Integer> q1 = new LinkedList<>();

Queue<Integer> q2 = new LinkedList<>();

public void push(int x) {

q2.add(x);

while (!q1.isEmpty()) {

q2.add(q1.remove());

}

Queue<Integer> temp = q1;

q1 = q2;

q2 = temp;

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

}

public int pop() {

if (q1.isEmpty()) {

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

return -1;

}

return q1.remove();

}

public int top() {

if (q1.isEmpty()) {

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

return -1;

}

return q1.peek();

}

public boolean isEmpty() {

return q1.isEmpty();

}

public void display() {

if (q1.isEmpty()) {

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

return;

}

System.out.println("Stack contents (top to bottom): " + q1);

}

public static void main(String[] args) {

StackUsingQueues stack = new StackUsingQueues();

Scanner sc = new Scanner(System.in);

int choice, value;

do {

System.out.println("\n1. Push\n2. Pop\n3. Top\n4. Display\n5. Exit");

System.out.print("Enter your choice: ");

choice = sc.nextInt();

switch (choice) {

case 1:

System.out.print("Enter value to push: ");

value = sc.nextInt();

stack.push(value);

break;

case 2:

int popped = stack.pop();

if (popped != -1)

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

break;

case 3:

int top = stack.top();

if (top != -1)

System.out.println("Top element: " + top);

break;

case 4:

stack.display();

break;

case 5:

System.out.println("Exiting...");

break;

default:

System.out.println("Invalid choice.");

}

} while (choice != 5);

sc.close();

}

}

set-3

import java.util.HashSet;

import java.util.Scanner;

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

next = null;

}

}

public class Program3a {

public static Node removeDuplicates(Node head) {

HashSet<Integer> seen = new HashSet<>();

Node current = head, prev = null;

while (current != null) {

if (seen.contains(current.data)) {

prev.next = current.next;

} else {

seen.add(current.data);

prev = current;

}

current = current.next;

}

return head;

}

public static void printList(Node head) {

while (head != null) {

System.out.print(head.data + " ");

head = head.next;

}

System.out.println();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of nodes: ");

int n = sc.nextInt();

if (n == 0) {

System.out.println("Empty list.");

return;

}

System.out.print("Enter elements: ");

Node head = new Node(sc.nextInt());

Node current = head;

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

current.next = new Node(sc.nextInt());

current = current.next;

}

System.out.print("Original List: ");

printList(head);

head = removeDuplicates(head);

System.out.print("List after removing duplicates: ");

printList(head);

}

}

3b)

import java.util.Scanner;

public class Program3b{

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

int low = 0, high = arr.length - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == target) {

return mid;

}

if (arr[low] <= arr[mid]) {

if (target >= arr[low] && target < arr[mid]) {

high = mid - 1;

} else {

low = mid + 1;

}

} else {

if (target > arr[mid] && target <= arr[high]) {

low = mid + 1;

} else {

high = mid - 1;

}

}

}

return -1;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of elements in the rotated sorted array: ");

int n = sc.nextInt();

int[] arr = new int[n];

System.out.print("Enter the rotated sorted array elements: ");

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

arr[i] = sc.nextInt();

}

System.out.print("Enter the target value to search: ");

int target = sc.nextInt();

int result = search(arr, target);

if (result == -1) {

System.out.println("Target not found in the array.");

} else {

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

}

}

}

set-4

import java.util.Scanner;

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

next = null;

}

}

public class Program4a{

public static Node deleteNthFromEnd(Node head, int n) {

if (head == null) return null;

Node first = head;

Node second = head;

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

if (first == null) return head;

first = first.next;

}

if (first == null) {

head = head.next;

return head;

}

while (first != null && first.next != null) {

first = first.next;

second = second.next;

}

second.next = second.next.next;

return head;

}

public static void printList(Node head) {

while (head != null) {

System.out.print(head.data + " ");

head = head.next;

}

System.out.println();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of nodes: ");

int n = sc.nextInt();

if (n == 0) {

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

return;

}

System.out.print("Enter the elements of the LinkedList: ");

Node head = new Node(sc.nextInt());

Node current = head;

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

current.next = new Node(sc.nextInt());

current = current.next;

}

System.out.print("Enter the position from the end to delete: ");

int positionFromEnd = sc.nextInt();

System.out.print("Original List: ");

printList(head);

head = deleteNthFromEnd(head, positionFromEnd);

System.out.print("List after deleting " + positionFromEnd + "th node from the end: ");

printList(head);

}

}

4b)

import java.util.Arrays;

import java.util.Scanner;

public class Program4b {

public static float findCeil(float[] arr, float x) {

Arrays.sort(arr);

for (float num : arr) {

if (num >= x) {

return num;

}

}

return -1;

}

public static float findFloor(float[] arr, float x) {

Arrays.sort(arr);

float floor = -1;

for (int i = arr.length - 1; i >= 0; i--) {

if (arr[i] <= x) {

floor = arr[i];

break;

}

}

return floor;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = sc.nextInt();

float[] arr = new float[n];

System.out.print("Enter the array elements: ");

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

arr[i] = sc.nextFloat();

}

System.out.print("Enter the target value x: ");

float x = sc.nextFloat();

float ceil = findCeil(arr, x);

float floor = findFloor(arr, x);

if (ceil != -1) {

System.out.println("Ceil of " + x + " is: " + ceil);

} else {

System.out.println("No Ceil found for " + x);

}

if (floor != -1) {

System.out.println("Floor of " + x + " is: " + floor);

} else {

System.out.println("No Floor found for " + x);

}

}

}

set-5

import java.util.Scanner;

public class Program5a{

public static int findUnique(int[] arr) {

int n = arr.length;

for (int i = 0; i < n - 1; i += 2) {

if (arr[i] != arr[i + 1]) {

return arr[i];

}

}

return (n > 0) ? arr[n - 1] : 2;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = sc.nextInt();

int[] arr = new int[n];

System.out.print("Enter the sorted array elements: ");

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

arr[i] = sc.nextInt();

}

int result = findUnique(arr);

System.out.println("The element that occurs only once is: " + result);

}

}

5b)

import java.util.Scanner;

class LinkedList {

Node head;

static class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

public void append(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

return;

}

Node last = head;

while (last.next != null) {

last = last.next;

}

last.next = newNode;

}

public void reverse() {

Node prev = null;

Node current = head;

Node next = null;

while (current != null) {

next = current.next;

current.next = prev;

prev = current;

current = next;

}

head = prev;

}

public void printList() {

Node temp = head;

while (temp != null) {

System.out.print(temp.data + " ");

temp = temp.next;

}

System.out.println();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

LinkedList list = new LinkedList();

System.out.print("Enter the number of elements in the linked list: ");

int n = sc.nextInt();

System.out.println("Enter the elements:");

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

int data = sc.nextInt();

list.append(data);

}

System.out.print("Original Linked List: ");

list.printList();

list.reverse();

System.out.print("Reversed Linked List: ");

list.printList();

}

}

set-6

import java.util.HashMap;

import java.util.Scanner;

public class Program6a{

public static int[] twoSum(int[] nums, int target) {

HashMap<Integer, Integer> map = new HashMap<>();

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

int complement = target - nums[i];

if (map.containsKey(complement)) {

return new int[] { map.get(complement), i };

}

map.put(nums[i], i);

}

)

return new int[] {};

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = sc.nextInt();

int[] nums = new int[n];

System.out.print("Enter the array elements: ");

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

nums[i] = sc.nextInt();

}

System.out.print("Enter the target sum: ");

int target = sc.nextInt();

int[] result = twoSum(nums, target);

if (result.length == 0) {

System.out.println("No solution found.");

} else {

System.out.println("Indices of the two numbers: [" + result[0] + ", " + result[1] + "]");

}

}

}

6b)

import java.util.Scanner;

public class Program6b {

static class Node {

int data;

Node next;

Node(int d) { data = d; next = null; }

}

public static boolean searchIterative(Node head, int target) {

Node curr = head;

while (curr != null) {

if (curr.data == target) return true;

curr = curr.next;

}

return false;

}

public static Node buildList(int[] vals) {

if (vals.length == 0) return null;

Node head = new Node(vals[0]), tail = head;

for (int i = 1; i < vals.length; i++) {

tail.next = new Node(vals[i]);

tail = tail.next;

}

return head;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of nodes: ");

int n = sc.nextInt();

int[] vals = new int[n];

System.out.print("Enter list elements: ");

for (int i = 0; i < n; i++) vals[i] = sc.nextInt();

System.out.print("Enter target to search: ");

int target = sc.nextInt();

Node head = buildList(vals);

boolean found = searchIterative(head, target);

System.out.println("Element " + target +

(found ? " found" : " not found") + " using Iteration.");

}

}

Recursion

import java.util.Scanner;

public class SearchRecursive {

static class Node {

int data;

Node next;

Node(int d) { data = d; next = null; }

}

public static boolean searchRecursive(Node head, int target) {

if (head == null) return false;

if (head.data == target) return true;

return searchRecursive(head.next, target);

}

public static Node buildList(int[] vals) {

if (vals.length == 0) return null;

Node head = new Node(vals[0]), tail = head;

for (int i = 1; i < vals.length; i++) {

tail.next = new Node(vals[i]);

tail = tail.next;

}

return head;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of nodes: ");

int n = sc.nextInt();

int[] vals = new int[n];

System.out.print("Enter list elements: ");

for (int i = 0; i < n; i++) vals[i] = sc.nextInt();

System.out.print("Enter target to search: ");

int target = sc.nextInt();

Node head = buildList(vals);

boolean found = searchRecursive(head, target);

System.out.println("Element " + target +

(found ? " found" : " not found") + " using Recursion.");

}}

set-7

import java.util.Scanner;

import java.util.Set;

import java.util.TreeSet;

public class Program7a{

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of elements: ");

int n = sc.nextInt();

System.out.print("Enter array elements: ");

Set<Integer> uniques = new TreeSet<>();

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

int x = sc.nextInt();

uniques.add(x); // add is O(log n)

}

System.out.print("Array after removing duplicates: ");

for (int x : uniques) {

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

}

System.out.println();

}

}

7b)

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

public class Program7b {

public static void reverseQueue(Queue<Integer> q) {

if (q.isEmpty()) {

return;

}

int front = q.poll();

reverseQueue(q);

q.offer(front);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of elements in queue: ");

int n = sc.nextInt();

Queue<Integer> queue = new LinkedList<>();

System.out.print("Enter queue elements: ");

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

queue.offer(sc.nextInt());

}

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

System.out.println(queue);

reverseQueue(queue);

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

System.out.println(queue);

}

}

set-8

import java.util.Scanner;

public class RemoveConsecutiveDuplicates {

public static String removeConsecutiveDuplicates(String str) {

if (str.length() == 0) return "";

StringBuilder result = new StringBuilder();

result.append(str.charAt(0));

for (int i = 1; i < str.length(); i++) {

if (str.charAt(i) != str.charAt(i - 1)) {

result.append(str.charAt(i));

}

}

return result.toString();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a string: ");

String input = sc.nextLine();

String output = removeConsecutiveDuplicates(input);

System.out.println("After removing consecutive duplicates: " + output);

}

}

8b)

import java.util.Scanner;

import java.util.Stack;

public class SortStack {

public static Stack<Integer> sortStack(Stack<Integer> input) {

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

while (!input.isEmpty()) {

int temp = input.pop();

while (!tempStack.isEmpty() && tempStack.peek() > temp) {

input.push(tempStack.pop());

}

tempStack.push(temp);

}

return tempStack;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

System.out.print("Enter number of elements: ");

int n = sc.nextInt();

System.out.print("Enter elements: ");

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

stack.push(sc.nextInt());

}

Stack<Integer> sorted = sortStack(stack);

System.out.print("Sorted Stack (top to bottom): ");

while (!sorted.isEmpty()) {

System.out.print(sorted.pop() + " ");

}

}

}

set-9

import java.util.Scanner;

public class EquilibriumIndex {

public static int findEquilibriumIndex(int[] arr) {

int totalSum = 0;

for (int num : arr) {

totalSum += num;

}

int leftSum = 0;

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

totalSum -= arr[i];

if (leftSum == totalSum) {

return I;

}

leftSum += arr[i];

}

return -1;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of elements: ");

int n = sc.nextInt();

int[] arr = new int[n];

System.out.print("Enter elements: ");

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

arr[i] = sc.nextInt();

}

int index = findEquilibriumIndex(arr);

if (index != -1) {

System.out.println("Equilibrium index is: " + index);

} else {

System.out.println("No equilibrium index found.");

}

}

}

9b)

import java.util.Scanner;

class Node {

int data;

Node left, right;

Node(int data) {

this.data = data;

left = right = null;

}

}

class BST {

Node root;

Node insert(Node root, int data) {

if (root == null) return new Node(data);

if (data < root.data)

root.left = insert(root.left, data);

else

root.right = insert(root.right, data);

return root;

}

int diameterUtil(Node node, int[] maxDiameter) {

if (node == null) return 0;

int leftHeight = diameterUtil(node.left, maxDiameter);

int rightHeight = diameterUtil(node.right, maxDiameter);

maxDiameter[0] = Math.max(maxDiameter[0], leftHeight + rightHeight + 1);

return 1 + Math.max(leftHeight, rightHeight);

}

int getDiameter() {

int[] maxDiameter = new int[1];

diameterUtil(root, maxDiameter);

return maxDiameter[0];

}

}

public class DiameterOfBST {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

BST tree = new BST();

System.out.print("Enter number of nodes: ");

int n = sc.nextInt();

System.out.print("Enter " + n + " elements: ");

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

int val = sc.nextInt();

tree.root = tree.insert(tree.root, val);

}

System.out.println("Diameter of the BST is: " + tree.getDiameter());

}

}

set-10

import java.util.\*;

public class BSTMirror {

// Node class to represent a tree node

static class Node {

int data;

Node left, right;

Node(int val) {

data = val;

left = right = null;

}

}

// Function to insert a node in BST

static Node insert(Node root, int data) {

if (root == null) {

return new Node(data);

}

if (data < root.data) {

root.left = insert(root.left, data);

} else {

root.right = insert(root.right, data);

}

return root;

}

// Function to print inorder traversal of the tree

static void inorder(Node root) {

if (root != null) {

inorder(root.left);

System.out.print(root.data + " ");

inorder(root.right);

}

}

// Function to generate the mirror of the tree

static Node mirror(Node root) {

if (root == null) return null;

// Recursively mirror the left and right subtrees

Node left = mirror(root.left);

Node right = mirror(root.right);

// Swap left and right children

root.left = right;

root.right = left;

return root;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of nodes in the BST: ");

int n = sc.nextInt();

Node root = null;

System.out.println("Enter the node values:");

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

int value = sc.nextInt();

root = insert(root, value);

}

System.out.print("Inorder traversal of BST: ");

inorder(root);

System.out.println();

root = mirror(root);

System.out.print("Inorder traversal of mirrored BST: ");

inorder(root);

}

}

10b)

import java.util.Scanner;

public class MinOperationsToConvertArrays {

// Method to calculate the minimum operations required

public static int minOperations(int[] arr1, int[] arr2) {

int operations = 0;

// Loop through both arrays and sum up the absolute differences

for (int i = 0; i < arr1.length; i++) {

operations += Math.abs(arr1[i] - arr2[i]); // Absolute difference between arr1 and arr2

}

return operations; // Return the total number of operations

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Read the size of arrays

System.out.print("Enter the number of elements in the arrays: ");

int n = sc.nextInt(); // Size of the arrays

// Declare and read elements for the first array

int[] arr1 = new int[n];

System.out.println("Enter elements for the first array:");

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

arr1[i] = sc.nextInt();

}

// Declare and read elements for the second array

int[] arr2 = new int[n];

System.out.println("Enter elements for the second array:");

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

arr2[i] = sc.nextInt();

}

// Calculate minimum operations to convert arr1 to arr2

int result = minOperations(arr1, arr2);

// Print the result

System.out.println("Minimum number of operations to convert arr1 to arr2: " + result);

}

}

set-11

a) total cost of connections

Ex : [5,4,2,8] - 4+2=6

[5,6,8] - 5+6=11

[11,8] - 11+9=19 Total cost = 6+11+19=36 TC – nlogn

import java.util.PriorityQueue;

import java.util.Scanner;

public class MinConnectionCost {

public static int totalConnectionCost(int[] lengths) {

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

for (int length : lengths) {

minHeap.add(length);

}

int totalCost = 0;

while (minHeap.size() > 1) {

int first = minHeap.poll(); // Smallest

int second = minHeap.poll(); // Second smallest

int cost = first + second;

totalCost += cost;

minHeap.add(cost);

}

return totalCost;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of ropes: ");

int n = scanner.nextInt();

int[] lengths = new int[n];

System.out.println("Enter the lengths of the ropes:");

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

lengths[i] = scanner.nextInt();

}

int cost = totalConnectionCost(lengths);

System.out.println("Total Cost = " + cost);

} }

Output:

Enter the number of ropes: 4

Enter the lengths of the ropes:

5 4 2 8

Total Cost = 36

11b) Contruct a BST and remove all half nodes Half nodes - nodes with single child

import java.util.Scanner;

class Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

public class DynamicBSTHalfNodeRemoval {

public static Node insert(Node root, int key) {

if (root == null) return new Node(key);

if (key < root.data)

root.left = insert(root.left, key);

else

root.right = insert(root.right, key);

return root;

}

public static Node removeHalfNodes(Node root) {

if (root == null) return null;

root.left = removeHalfNodes(root.left);

root.right = removeHalfNodes(root.right);

if (root.left == null && root.right != null)

return root.right;

if (root.left != null && root.right == null)

return root.left;

return root;

}

public static void inorder(Node root) {

if (root != null) {

inorder(root.left);

System.out.print(root.data + " ");

inorder(root.right);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Node root = null;

System.out.println("Enter number of nodes to insert:");

int n = sc.nextInt();

System.out.println("Enter " + n + " values:");

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

int val = sc.nextInt();

root = insert(root, val);

}

System.out.print("\nInorder before removing half nodes: ");

inorder(root);

root = removeHalfNodes(root);

System.out.print("\nInorder after removing half nodes: ");

inorder(root);

}

}

Output:

Enter number of nodes to insert:

6 Enter 6 values:

10 5 2 7 15 12

Inorder before removing half nodes: 2 5 7 10 12 15

Inorder after removing half nodes: 2 5 7 10 15

set-12

a )bst print left nodes

import java.util.Scanner;

class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

class BinarySearchTree {

Node root;

public void insert(int data) {

root = insertRec(root, data);

}

private Node insertRec(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data)

root.left = insertRec(root.left, data);

else if (data > root.data)

root.right = insertRec(root.right, data);

return root;

}

public void printLeftNodes() {

printLeftNodesRec(root);

}

private void printLeftNodesRec(Node root) {

if (root != null) {

if (root.left != null) {

System.out.print(root.left.data + " ");

}

printLeftNodesRec(root.left);

printLeftNodesRec(root.right);

}

}

}

public class Main {

public static void main(String[] args) {

BinarySearchTree bst = new BinarySearchTree();

Scanner scanner = new Scanner(System.in);

System.out.println("Enter numbers for BST (type 'done' to finish):");

while (true) {

String input = scanner.nextLine();

if (input.equalsIgnoreCase("done")) break;

try {

int value = Integer.parseInt(input);

bst.insert(value);

} catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter a valid integer.");

}

}

System.out.println("\nLeft nodes of the BST:");

bst.printLeftNodes();

scanner.close();

}

}

Output:

Enter numbers for BST (type 'done' to finish):

50

30

20

40

70

Done

Left nodes of the BST:

30 20 40

12 b)product of an array without division operator

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the size of the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter elements of the array:");

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

arr[i] = scanner.nextInt();

}

int[] result = productArray(arr, n);

System.out.println("Product of array elements excluding each element:");

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

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

}

scanner.close();

}

public static int[] productArray(int[] arr, int n) {

int[] result = new int[n];

int[] leftProd = new int[n];

int[] rightProd = new int[n];

leftProd[0] = 1;

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

leftProd[i] = leftProd[i - 1] \* arr[i - 1];

}

rightProd[n - 1] = 1;

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

rightProd[i] = rightProd[i + 1] \* arr[i + 1];

}

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

result[i] = leftProd[i] \* rightProd[i];

}

return result;

}

}

Output:

Enter the size of the array: 4

Enter elements of the array:

2

3

4

5

Product of array elements excluding each element:

60 40 30 24

set-13

a) Assign cookies (gfg)

import java.util.Arrays;

public class AssignCookies {

public static void main(String[] args) {

int[] greed = {1, 2, 3};

int[] cookies = {1, 1};

System.out.println("Number of children satisfied: " + findContentChildren(greed,

cookies));

}

public static int findContentChildren(int[] greed, int[] cookies) {

Arrays.sort(greed);

Arrays.sort(cookies);

int childIndex = 0;

int cookieIndex = 0;

int satisfiedChildren = 0;

while (childIndex < greed.length && cookieIndex < cookies.length) {

if (cookies[cookieIndex] >= greed[childIndex]) {

satisfiedChildren++;

childIndex++;

}

cookieIndex++;

}

return satisfiedChildren;

}

}

Output:

Greed factors of children: [1, 2, 3]

Cookie sizes: [1, 1]

Number of children satisfied: 1

13 b) Width of binary search tree at given level

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

class BinarySearchTree {

Node root;

public int widthAtLevel(Node root, int level) {

if (root == null) {

return 0;

}

Queue<Node> queue = new LinkedList<>();

queue.add(root);

int currentLevel = 0;

int widthAtGivenLevel = 0;

while (!queue.isEmpty()) {

int nodeCount = queue.size();

if (currentLevel == level) {

widthAtGivenLevel = nodeCount;

}

while (nodeCount-- > 0) {

Node currentNode = queue.poll();

if (currentNode.left != null) {

queue.add(currentNode.left);

}

if (currentNode.right != null) {

queue.add(currentNode.right);

}

}

currentLevel++;

if (currentLevel > level) {

break;

}

}

return widthAtGivenLevel;

}

public void insert(int data) {

root = insertRec(root, data);

}

private Node insertRec(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data) {

root.left = insertRec(root.left, data);

} else if (data > root.data) {

root.right = insertRec(root.right, data);

}

return root;

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BinarySearchTree tree = new BinarySearchTree();

System.out.print("Enter the number of nodes to insert into the BST: ");

int n = scanner.nextInt();

System.out.println("Enter the values for the BST nodes:");

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

int value = scanner.nextInt();

tree.insert(value);

}

System.out.print("Enter the level at which to calculate the width: ");

int level = scanner.nextInt();

int width = tree.widthAtLevel(tree.root, level);

System.out.println("Width of the BST at level " + level + " is: " + width);

scanner.close();

}

}

Output:

Enter the number of nodes to insert into the BST: 7

Enter the values for the BST nodes:

50 30 20 40 70 60 80

Enter the level at which to calculate the width: 2

Width of the BST at level 2 is: 4

set-14

a) lemonade. Exchange change of all the queue of customers

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

class Customer {

int payment;

public Customer(int payment) {

this.payment = payment;

}

}

class LemonadeStand {

private static final int LEMONADE\_PRICE = 5;

private int changeInRegister;

public LemonadeStand(int initialChange) {

this.changeInRegister = initialChange;

}

public boolean serveCustomer(Customer customer) {

if (customer.payment < LEMONADE\_PRICE) {

System.out.println("Customer does not have enough money.");

return false;

}

int changeRequired = customer.payment - LEMONADE\_PRICE;

if (changeRequired > changeInRegister) {

System.out.println("Not enough change in register to serve the customer.");

return false;

}

changeInRegister -= changeRequired;

System.out.println("Customer served. Change given: " + changeRequired);

return true;

}

}

public class LemonadeStandSimulation {

public static void main(String[] args) {

LemonadeStand stand = new LemonadeStand(20);

Queue<Customer> customers = new LinkedList<>();

Scanner scanner = new Scanner(System.in);

System.out.println("Welcome to the Lemonade Stand!

while (true) {

System.out.print("Enter payment amount of the customer (or type 'exit' to quit): ");

String input = scanner.nextLine();

if (input.equalsIgnoreCase("exit")) {

break;

}

try {

int payment = Integer.parseInt(input);

customers.add(new Customer(payment));

} catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter a valid amount or 'exit' to quit.");

continue;

}

System.out.println("Customer added with payment: " + input);

}

while (!customers.isEmpty()) {

Customer currentCustomer = customers.poll();

boolean served = stand.serveCustomer(currentCustomer);

if (!served) {

System.out.println("Unable to serve this customer.");

break; // If one customer can't be served, stop serving further customers

}

}

scanner.close();

}

}

Output:

Welcome to the Lemonade Stand!

Enter payment amount of the customer (or type 'exit' to quit): 10

Customer added with payment: 10

Enter payment amount of the customer (or type 'exit' to quit): 5

Customer added with payment: 5

Enter payment amount of the customer (or type 'exit' to quit): 20

Customer added with payment: 20

Enter payment amount of the customer (or type 'exit' to quit): exit

Customer served. Change given: 5

Customer served. Change given: 0

Not enough change in register to serve the customer.

Unable to serve this customer.

14)B) binary search tree to find height of the tree

import java.util.Scanner;

class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

class BinarySearchTree {

Node root;

public void insert(int data) {

root = insertRec(root, data);

}

private Node insertRec(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data) root.left = insertRec(root.left, data);

else if (data > root.data) root.right = insertRec(root.right, data);

return root;

}

public int height() {

return heightRec(root);

}

private int heightRec(Node node) {

if (node == null) return -1;

return Math.max(heightRec(node.left), heightRec(node.right)) + 1;

}

public void inorder() {

inorderRec(root);

}

private void inorderRec(Node root) {

if (root != null) {

inorderRec(root.left);

System.out.print(root.data + " ");

inorderRec(root.right);

}

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BinarySearchTree tree = new BinarySearchTree();

while (true) {

System.out.print("Enter a number (or 'exit' to stop): ");

String input = scanner.nextLine();

if (input.equalsIgnoreCase("exit")) break;

try {

int value = Integer.parseInt(input);

tree.insert(value);

} catch (NumberFormatException e) {

System.out.println("Invalid input.");

}

}

System.out.println("\nIn-order traversal of the BST:");

tree.inorder();

System.out.println("\nHeight of the tree: " + tree.height());

scanner.close();

}

}

Output:

Enter a number (or 'exit' to stop): 50

Enter a number (or 'exit' to stop): 30

Enter a number (or 'exit' to stop): 20

Enter a number (or 'exit' to stop): 40

Enter a number (or 'exit' to stop): 70

Enter a number (or 'exit' to stop): exit

In-order traversal of the BST:

20 30 40 50 70

Height of the tree: 2

set-15

a) Given 3 stacks with positive numbers...Find maximum equal sum among the 3

stacks...

import java.util.Scanner;

import java.util.Stack;

public class EqualSumStacks {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in stack 1: ");

int n1 = scanner.nextInt();

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

System.out.println("Enter the elements of stack 1:");

for (int i = 0; i < n1; i++) {

stack1.push(scanner.nextInt());

}

System.out.print("Enter the number of elements in stack 2: ");

int n2 = scanner.nextInt();

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

System.out.println("Enter the elements of stack 2:");

for (int i = 0; i < n2; i++) {

stack2.push(scanner.nextInt());

}

System.out.print("Enter the number of elements in stack 3: ");

int n3 = scanner.nextInt();

Stack<Integer> stack3 = new Stack<>();

System.out.println("Enter the elements of stack 3:");

for (int i = 0; i < n3; i++) {

stack3.push(scanner.nextInt());

}

System.out.println("Maximum equal sum among the 3 stacks: " +

findMaxEqualSum(stack1, stack2, stack3));

scanner.close();

}

public static int findMaxEqualSum(Stack<Integer> stack1, Stack<Integer> stack2,

Stack<Integer> stack3) {

int sum1 = getSum(stack1);

int sum2 = getSum(stack2);

int sum3 = getSum(stack3);

while (sum1 != sum2 || sum2 != sum3) {

if (sum1 > sum2 && sum1 > sum3) {

sum1 -= stack1.pop();

} else if (sum2 > sum1 && sum2 > sum3) {

sum2 -= stack2.pop();

} else {

sum3 -= stack3.pop();

}

if (stack1.isEmpty() || stack2.isEmpty() || stack3.isEmpty()) {

return 0;

}

}

return sum1;

}

private static int getSum(Stack<Integer> stack) {

int sum = 0;

for (int num : stack) {

sum += num;

}

return sum;

}

}

Output:

Enter the number of elements in stack 1: 4

Enter the elements of stack 1:

3 2 1 1

Enter the number of elements in stack 2: 3

Enter the elements of stack 2:

4 1 1

Enter the number of elements in stack 3: 4

Enter the elements of stack 3:

2 3 1 1

Maximum equal sum among the 3 stacks: 3

15)b)Create a BST and find the max width of a BST

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

class BinarySearchTree {

Node root;

public BinarySearchTree() {

root = null;

}

public void insert(int data) {

root = insertRec(root, data);

}

private Node insertRec(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data) {

root.left = insertRec(root.left, data);

} else if (data > root.data) {

root.right = insertRec(root.right, data);

}

return root;

}

public int maxWidth() {

return getMaxWidth(root);

}

private int getMaxWidth(Node root) {

if (root == null) {

return 0;

}

Queue<Node> queue = new LinkedList<>();

queue.add(root);

int maxWidth = 0;

while (!queue.isEmpty()) {

int levelWidth = queue.size();

maxWidth = Math.max(maxWidth, levelWidth);

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

Node node = queue.poll();

if (node.left != null) {

queue.add(node.left);

}

if (node.right != null) {

queue.add(node.right);

}

}

}

return maxWidth;

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BinarySearchTree tree = new BinarySearchTree();

System.out.print("Enter the number of nodes in the BST: ");

int n = scanner.nextInt();

System.out.println("Enter the elements of the BST:");

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

int value = scanner.nextInt();

tree.insert(value);

}

int maxWidth = tree.maxWidth();

System.out.println("Maximum width of the BST: " + maxWidth);

scanner.close();

}

}

Output:

Enter the number of nodes in the BST: 7

Enter the elements of the BST:

10 20 30 5 15 25 35

Maximum width of the BST: 4

set-16

)a)Reverse level order traserval in bst

import java.util.LinkedList;

import java.util.Queue;

import java.util.Stack;

import java.util.Scanner;

class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

class BinarySearchTree {

Node root;

public BinarySearchTree() {

root = null;

}

public void insert(int data) {

root = insertRec(root, data);

}

private Node insertRec(Node root, int data) {

if (root == null) {

root = new Node(data);

return root;

}

if (data < root.data) {

root.left = insertRec(root.left, data);

} else if (data > root.data) {

root.right = insertRec(root.right, data);

}

return root;

}

public void reverseLevelOrder() {

if (root == null) {

return;

}

Queue<Node> queue = new LinkedList<>();

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

queue.add(root);

while (!queue.isEmpty()) {

Node node = queue.poll();

stack.push(node);

if (node.right != null) {

queue.add(node.right);

}

if (node.left != null) {

queue.add(node.left);

}

}

while (!stack.isEmpty()) {

System.out.print(stack.pop().data + " ");

}

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BinarySearchTree tree = new BinarySearchTree();

System.out.print("Enter the number of nodes in the BST: ");

int n = scanner.nextInt();

System.out.println("Enter the elements of the BST:");

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

int value = scanner.nextInt();

tree.insert(value);

}

System.out.println("Reverse level-order traversal of the BST:");

tree.reverseLevelOrder();

scanner.close();

}

}

Output:

Enter the number of nodes in the BST: 7

Enter the elements of the BST:

10 20 30 5 15 25 35

Reverse level-order traversal of the BST:

35 30 25 20 15 5 10

16)b)knapsack problem

import java.util.Scanner;

public class Knapsack {

public static int knapsack(int W, int[] weights, int[] values, int n) {

int[][] dp = new int[n + 1][W + 1];

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

for (int w = 0; w <= W; w++) {

if (i == 0 || w == 0) {

dp[i][w] = 0;

} else if (weights[i - 1] <= w) {

dp[i][w] = Math.max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);

} else {

dp[i][w] = dp[i - 1][w];

}

}

}

return dp[n][W];

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of items: ");

int n = scanner.nextInt();

System.out.print("Enter the capacity of the knapsack: ");

int W = scanner.nextInt();

int[] weights = new int[n];

int[] values = new int[n];

System.out.println("Enter the weights and values of the items:");

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

System.out.print("Weight of item " + (i + 1) + ": ");

weights[i] = scanner.nextInt();

System.out.print("Value of item " + (i + 1) + ": ");

values[i] = scanner.nextInt();

}

int maxValue = knapsack(W, weights, values, n);

System.out.println("Maximum value that can be obtained: " + maxValue);

scanner.close();

}

}

Output:

Enter the number of items: 4

Enter the capacity of the knapsack: 5

Enter the weights and values of the items:

Weight of item 1: 1

Value of item 1: 1

Weight of item 2: 3

Value of item 2: 4

Weight of item 3: 4

Value of item 3: 5

Weight of item 4: 2

Value of item 4: 3

Maximum value that can be obtained: 7

set-17

a)count all paths

class Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

public class CountPathsBinaryTree {

public static int countPaths(Node root) {

return countPathsUtil(root, 0);

}

public static int countPathsUtil(Node root, int currentSum) {

if (root == null) {

return 0;

}

currentSum += root.data;

if (root.left == null && root.right == null) {

return 1;

}

return countPathsUtil(root.left, currentSum) + countPathsUtil(root.right, currentSum);

}

public static Node insert(Node root, int key) {

if (root == null) return new Node(key);

if (key < root.data)

root.left = insert(root.left, key);

else

root.right = insert(root.right, key);

return root;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Node root = null;

System.out.print("Enter number of nodes: ");

int n = sc.nextInt();

System.out.println("Enter " + n + " node values:");

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

int val = sc.nextInt();

root = insert(root, val);

}

int totalPaths = countPaths(root);

System.out.println("Total number of root-to-leaf paths: " + totalPaths);

}

}

Output:

Enter number of nodes: 5

Enter 5 node values:

10 5 20 3 7

Total number of root-to-leaf paths: 4

17b) breadth first traversal BST

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

class Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

public class BFSBinaryTree {

public static void levelOrderTraversal(Node root) {

if (root == null) {

return;

}

Queue<Node> queue = new LinkedList<>();

queue.add(root);

while (!queue.isEmpty()) {

Node current = queue.poll();

System.out.print(current.data + " ");

if (current.left != null) {

queue.add(current.left);

}

if (current.right != null) {

queue.add(current.right);

}

}

}

public static Node insert(Node root, int key) {

if (root == null) return new Node(key);

if (key < root.data)

root.left = insert(root.left, key);

else

root.right = insert(root.right, key);

return root;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Node root = null;

System.out.print("Enter number of nodes: ");

int n = sc.nextInt();

System.out.println("Enter " + n + " node values:");

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

int val = sc.nextInt();

root = insert(root, val);

}

System.out.print("\nLevel-Order Traversal: ");

levelOrderTraversal(root);

}

}

Output:

Enter number of nodes: 7

Enter 7 node values:

10 5 20 3 7 15 25 Level-Order Traversal: 10 5 20 3 7 15 25

set-18

a)kadane algorithm

import java.util.Scanner;

public class KadanesAlgorithm {

public static int maxSubArraySum(int[] arr) {

int maxSoFar = arr[0];

int maxEndingHere = arr[0];

for (int i = 1; i < arr.length; i++) {

maxEndingHere = Math.max(arr[i], maxEndingHere + arr[i]);

maxSoFar = Math.max(maxSoFar, maxEndingHere);

}

return maxSoFar;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of elements: ");

int n = sc.nextInt();

int[] arr = new int[n];

System.out.println("Enter array elements:");

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

arr[i] = sc.nextInt();

}

int maxSum = maxSubArraySum(arr);

System.out.println("Maximum Subarray Sum = " + maxSum);

}

}

Output:

Enter number of elements: 8

Enter array elements: -2 -3 4 -1 -2 1 5 -3

Maximum Subarray Sum = 7

18b) traversals -inorder,preorder,postorder

import java.util.Scanner;

class Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

public class BinaryTreeTraversals {

public static void inorder(Node root) {

if (root != null) {

inorder(root.left);

System.out.print(root.data + " ");

inorder(root.right);

}

}

public static void preorder(Node root) {

if (root != null) {

System.out.print(root.data + " ");

preorder(root.left);

preorder(root.right);

}

}

public static void postorder(Node root) {

if (root != null) {

postorder(root.left);

postorder(root.right);

System.out.print(root.data + " ");

}

}

public static Node insert(Node root, int key) {

if (root == null) return new Node(key);

if (key < root.data)

root.left = insert(root.left, key);

else

root.right = insert(root.right, key);

return root;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Node root = null;

System.out.print("Enter number of nodes: ");

int n = sc.nextInt();

System.out.println("Enter " + n + " node values:");

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

int val = sc.nextInt();

root = insert(root, val); // building a BST for simplicity

}

System.out.print("\nInorder Traversal: ");

inorder(root);

System.out.print("\nPreorder Traversal: ");

preorder(root);

System.out.print("\nPostorder Traversal: ");

postorder(root);

}

}

Output:

Enter number of nodes: 5

Enter 5 node values:

10 5 20 3 7

Inorder Traversal: 3 5 7 10 20

Preorder Traversal: 10 5 3 7 20

Postorder Traversal: 3 7 5 20 10

set-19)

a) Reverse characters of the string using two pointers

import java.util.Scanner;

public class ReverseStringTwoPointers {

public static String reverse(String str) {

char[] chars = str.toCharArray();

int left = 0;

int right = chars.length - 1;

while (left < right) {

char temp = chars[left];

chars[left] = chars[right];

chars[right] = temp;

left++;

right--;

}

return new String(chars);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a string to reverse: ");

String input = sc.nextLine();

String reversed = reverse(input);

System.out.println("Reversed string: " + reversed);

}

}

Output:

Enter a string to reverse: hello world

Reversed string: dlrow olleh

19)b. Implement two stacks in one array

import java.util.Scanner;

public class TwoStacksDynamic {

int size;

int top1, top2;

int[] arr;

public TwoStacksDynamic(int n) {

size = n;

arr = new int[n];

top1 = -1;

top2 = n;

}

public void push1(int x) {

if (top1 < top2 - 1) {

arr[++top1] = x;

} else {

System.out.println("Stack 1 Overflow");

}

}

public void push2(int x) {

if (top1 < top2 - 1) {

arr[--top2] = x;

} else {

System.out.println("Stack 2 Overflow");

}

}

public int pop1() {

if (top1 >= 0) {

return arr[top1--];

} else {

System.out.println("Stack 1 Underflow");

return -1;

}

}

public int pop2() {

if (top2 < size) {

return arr[top2++];

} else {

System.out.println("Stack 2 Underflow");

return -1;

}

}

public void displayStacks() {

System.out.print("Stack 1: ");

for (int i = 0; i <= top1; i++) {

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

}

System.out.println();

System.out.print("Stack 2: ");

for (int i = size - 1; i >= top2; i--) {

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

}

System.out.println();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter size of the array: ");

int n = sc.nextInt();

TwoStacksDynamic ts = new TwoStacksDynamic(n);

int choice;

do {

System.out.println("\n1. Push to Stack 1");

System.out.println("2. Push to Stack 2");

System.out.println("3. Pop from Stack 1");

System.out.println("4. Pop from Stack 2");

System.out.println("5. Display Stacks");

System.out.println("6. Exit");

System.out.print("Choose an option: ");

choice = sc.nextInt();

switch (choice) {

case 1:

System.out.print("Enter value to push to Stack 1: ");

ts.push1(sc.nextInt());

break;

case 2:

System.out.print("Enter value to push to Stack 2: ");

ts.push2(sc.nextInt());

break;

case 3:

System.out.println("Popped from Stack 1: " + ts.pop1());

break;

case 4:

System.out.println("Popped from Stack 2: " + ts.pop2());

break;

case 5:

ts.displayStacks();

break;

case 6:

System.out.println("Exiting...");

break;

default:

System.out.println("Invalid choice!");

}

} while (choice != 6);

sc.close();

}

}

Output:

Enter size of the array: 6

1. Push to Stack 1

2. Push to Stack 2

3. Pop from Stack 1

4. Pop from Stack 2

5. Display Stacks

6. Exit

Choose an option: 1

Enter value to push to Stack 1: 10

Choose an option: 2

Enter value to push to Stack 2: 100

Choose an option: 5

Stack 1: 10

Stack 2: 100

set-20

a) given array sum of two numbers in array is equal to third numbers (triplet program)

input array:{3,1,17,19,21,2} Output array:19,21,2 Input array:{3,1,17,19,21,0} Triplet not

exists

import java.util.Scanner;

public class TripletCheckerDynamic {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of elements in the array:");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array:");

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

arr[i] = scanner.nextInt();

}

checkTriplet(arr);

}

public static void checkTriplet(int[] arr) {

boolean found = false;

int length = arr.length;

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

for (int j = i + 1; j < length; j++) {

for (int k = 0; k < length; k++) {

if (k != i && k != j && arr[i] + arr[j] == arr[k]) {

System.out.println(arr[i] + ", " + arr[j] + ", " + arr[k]);

found = true;

return; // Stop after finding the first triplet

}

}

}

}

if (!found) {

System.out.println("Triplet not exists");

}

}

}

Output:

Enter the number of elements in the array:

6

Enter the elements of the array:

3 1 17 19 21 2

20)b) build code to implement DFS using adjacency list without recursion

import java.util.\*;

public class DFSWithoutRecursionDynamic {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of nodes in the graph:");

int nodes = scanner.nextInt();

Map<Integer, List<Integer>> graph = new HashMap<>();

System.out.println("Enter the edges in the graph (as pairs of nodes):");

System.out.println("Type '-1 -1' to stop entering edges.");

while (true) {

int from = scanner.nextInt();

int to = scanner.nextInt();

if (from == -1 && to == -1) break;

graph.putIfAbsent(from, new ArrayList<>());

graph.get(from).add(to);

graph.putIfAbsent(to, new ArrayList<>());

graph.get(to).add(from);

}

System.out.println("Enter the starting node for DFS:");

int startNode = scanner.nextInt();

System.out.println("DFS Traversal:");

dfs(startNode, graph);

}

public static void dfs(int startNode, Map<Integer, List<Integer>> graph) {

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

Set<Integer> visited = new HashSet<>();

stack.push(startNode);

visited.add(startNode);

while (!stack.isEmpty()) {

int currentNode = stack.pop();

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

List<Integer> neighbors = graph.getOrDefault(currentNode, new ArrayList<>());

for (int neighbor : neighbors) {

if (!visited.contains(neighbor)) {

stack.push(neighbor);

visited.add(neighbor);

}

}

}

}

}

Output:

Enter the number of nodes in the graph:5

Enter the edges in the graph (as pairs of nodes):

0 1

0 2

1 3

1 4 -1 -1

Enter the starting node for DFS:0

DFS Traversal:0 2 1 4 3

set-21

a)

import java.util.\*;

public class Program21a {

public static int distributeCandies(int[] rankings) {

int n = rankings.length;

int[] candies = new int[n];

Arrays.fill(candies, 1);

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

if (rankings[i] > rankings[i - 1]) {

candies[i] = candies[i - 1] + 1;

}

}

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

if (rankings[i] > rankings[i + 1]) {

candies[i] = Math.max(candies[i], candies[i + 1] + 1);

}

}

int totalCandies = Arrays.stream(candies).sum();

return totalCandies;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of students: ");

int n = scanner.nextInt();

int[] rankings = new int[n];

System.out.println("Enter rankings of students:");

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

rankings[i] = scanner.nextInt();

}

int result = distributeCandies(rankings);

System.out.println("Minimum candies required: " + result);

scanner.close();

}

}

Output:

Enter number of students: 6

Enter rankings of students:

1 2 2 3 4 2

Minimum candies required: 10

21b)

import java.util.\*;

public class Program21b {

private static void dfs(int node, List<List<Integer>> adjList, boolean[] visited) {

visited[node] = true;

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

for (int neighbor : adjList.get(node)) {

if (!visited[neighbor]) {

dfs(neighbor, adjList, visited);

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of vertices: ");

int V = scanner.nextInt();

System.out.print("Enter number of edges: ");

int E = scanner.nextInt();

List<List<Integer>> adjList = new ArrayList<>();

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

adjList.add(new ArrayList<Integer>()); // Explicitly specify Integer type

}

System.out.println("Enter edges (format: u v):");

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

int u = scanner.nextInt();

int v = scanner.nextInt();

adjList.get(u).add(v);

adjList.get(v).add(u);

}

boolean[] visited = new boolean[V];

System.out.println("DFS Traversal starting from node 0:");

dfs(0, adjList, visited);

scanner.close();

}

}

Output:

Enter number of vertices: 5

Enter number of edges: 4

Enter edges (format: u v):

0 1

0 2

1 3

1 4

DFS Traversal starting from node 0:

0 1 3 4 2

set-22

a)

import java.util.Scanner;

public class Program22a {

public static int countWays(int n) {

if (n < 0) return 0;

if (n == 0) return 1;

int[] dp = new int[n + 1];

dp[0] = 1;

if (n >= 1) dp[1] = 1;

if (n >= 2) dp[2] = 2;

for (int i = 3; i <= n; i++) {

dp[i] = dp[i - 1] + dp[i - 2] + dp[i - 3];

}

return dp[n];

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of stairs: ");

int n = scanner.nextInt();

int ways = countWays(n);

System.out.println("Number of ways to climb " + n + " stairs: " + ways);

scanner.close();

}

}

Output:

Enter the number of stairs: 5

Number of ways to climb 5 stairs: 13

22b)

import java.util–.\*;

public class Program22b {

private static void dfs(int node, List<List<Integer>> adjList, boolean[] visited) {

visited[node] = true;

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

for (int neighbor : adjList.get(node)) {

if (!visited[neighbor]) {

dfs(neighbor, adjList, visited);

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of vertices: ");

int V = scanner.nextInt();

System.out.print("Enter number of edges: ");

int E = scanner.nextInt();

List<List<Integer>> adjList = new ArrayList<>();

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

adjList.add(new ArrayList<Integer>());

}

System.out.println("Enter edges (format: u v):");

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

int u = scanner.nextInt();

int v = scanner.nextInt();

adjList.get(u).add(v);

adjList.get(v).add(u);

}

boolean[] visited = new boolean[V];

System.out.println("DFS Traversal starting from node 0:");

dfs(0, adjList, visited);

scanner.close();

}

}

Output:

Enter number of vertices: 5

Enter number of edges: 4

Enter edges:

0 1

0 2

1 3

2 4

22b)without recursion

import java.util.\*;

public class Program22b {

public static void dfsTraversal(int[][] graph, int n) {

boolean[] visited = new boolean[n];

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

stack.push(0);

while (!stack.isEmpty()) {

int node = stack.pop();

if (!visited[node]) {

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

visited[node] = true;

// Push adjacent nodes in reverse order for correct traversal

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

if (graph[node][i] == 1 && !visited[i]) {

stack.push(i);

}

}

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of vertices: ");

int n = scanner.nextInt();

System.out.print("Enter number of edges: ");

int e = scanner.nextInt();

int[][] graph = new int[n][n];

System.out.println("Enter edges (u v):");

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

int u = scanner.nextInt();

int v = scanner.nextInt();

graph[u][v] = 1;

graph[v][u] = 1; // because undirected

}

System.out.println("DFS traversal starting from node 0:");

dfsTraversal(graph, n);

scanner.close();

}

}

set-23

a)

import java.util.Scanner;

public class Program23a {

public static int lcs(String str1, String str2) {

int m = str1.length();

int n = str2.length();

int[][] dp = new int[m + 1][n + 1];

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

for (int j = 1; j <= n; j++) {

if (str1.charAt(i - 1) == str2.charAt(j - 1)) {

dp[i][j] = 1 + dp[i - 1][j - 1]; // Match

} else {

dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);

}

}

}

return dp[m][n];

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter first string: ");

String str1 = scanner.nextLine();

System.out.print("Enter second string: ");

String str2 = scanner.nextLine();

int result = lcs(str1, str2);

System.out.println("Length of Longest Common Subsequence: " + result);

scanner.close();

}

}

Output:

Enter first string: ABCDGH

Enter second string: AEDFHR

Length of Longest Common Subsequence: 3

23b)

import java.util.\*;

public class Program23b {

private static void dfs(int node, List<List<Integer>> adjList, boolean[] visited) {

visited[node] = true;

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

for (int neighbor : adjList.get(node)) {

if (!visited[neighbor]) {

dfs(neighbor, adjList, visited);

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of vertices: ");

int V = scanner.nextInt();

System.out.print("Enter number of edges: ");

int E = scanner.nextInt();

List<List<Integer>> adjList = new ArrayList<>();

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

adjList.add(new ArrayList<>());

}

System.out.println("Enter edges (format: u v):");

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

int u = scanner.nextInt();

int v = scanner.nextInt();

adjList.get(u).add(v);

adjList.get(v).add(u); // Assuming undirected graph

}

boolean[] visited = new boolean[V];

System.out.print("Enter the starting node for DFS: ");

int startNode = scanner.nextInt();

System.out.println("DFS Traversal:");

dfs(startNode, adjList, visited);

scanner.close();

}

}

Output:

Enter number of vertices: 5

Enter number of edges: 4

Enter edges:

0 1

0 2

1 3

2 4

Enter the starting node for DFS: 0

DFS Traversal:

0 1 3 2 4

set-24

a)

import java.util.\*;

public class Program24a {

public static List<List<Integer>> findTriplets(int[] nums) {

List<List<Integer>> result = new ArrayList<>();

Arrays.sort(nums); // Sort array

int n = nums.length;

for (int i = 0; i < n - 2; i++) {

if (i > 0 && nums[i] == nums[i - 1]) continue;

int left = i + 1, right = n - 1;

while (left < right) {

int sum = nums[i] + nums[left] + nums[right];

if (sum == 0) {

result.add(Arrays.asList(nums[i], nums[left], nums[right]));

while (left < right && nums[left] == nums[left + 1]) left++;

while (left < right && nums[right] == nums[right - 1]) right--;

left++;

right--;

} else if (sum < 0) {

left++;

} else {

right--;

}

}

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements: ");

int n = scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter array elements:");

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

nums[i] = scanner.nextInt();

}

List<List<Integer>> result = findTriplets(nums);

System.out.println("Triplets with sum zero: " + result);

scanner.close();

}

}

Output:

Enter number of elements: 6

Enter array elements: -1 0 1 2 -1 -4

Triplets with sum zero: [[-1, -1, 2], [-1, 0, 1]]

24b)

import java.util.\*;

public class Program24b {

public static void bfs(int startNode, int[][] adjMatrix, int V) {

boolean[] visited = new boolean[V];

Queue<Integer> queue = new LinkedList<>();

visited[startNode] = true;

queue.add(startNode);

System.out.println("BFS Traversal:");

while (!queue.isEmpty()) {

int node = queue.poll();

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

for (int neighbor = 0; neighbor < V; neighbor++) {

if (adjMatrix[node][neighbor] == 1 && !visited[neighbor]) {

visited[neighbor] = true;

queue.add(neighbor);

}

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of vertices: ");

int V = scanner.nextInt();

int[][] adjMatrix = new int[V][V];

System.out.print("Enter number of edges: ");

int E = scanner.nextInt();

System.out.println("Enter edges (format: u v):");

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

int u = scanner.nextInt();

int v = scanner.nextInt();

adjMatrix[u][v] = 1;

adjMatrix[v][u] = 1; // Assuming an undirected graph

}

System.out.print("Enter the starting node for BFS: ");

int startNode = scanner.nextInt();

bfs(startNode, adjMatrix, V);

scanner.close();

}

}

Output:

Enter number of vertices: 5

Enter number of edges: 4

Enter edges:

0 1

0 2

1 3

2 4

Enter the starting node for BFS: 0

BFS Traversal:

0 1 2 3 4

set-25

a)

import java.util.\*;

public class Program25a {

public static int findMinPlatforms(int[] arrival, int[] departure) {

Arrays.sort(arrival);

Arrays.sort(departure);

int platforms = 1, maxPlatforms = 1;

int i = 1, j = 0; // i -> arrival, j -> departure

int n = arrival.length;

while (i < n && j < n) {

if (arrival[i] <= departure[j]) {

platforms++; // New train arrives, need more platforms

i++;

} else {

platforms--; // Train departs, free up a platform

j++;

}

maxPlatforms = Math.max(maxPlatforms, platforms);

}

return maxPlatforms;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of trains: ");

int n = scanner.nextInt();

int[] arrival = new int[n];

int[] departure = new int[n];

System.out.println("Enter arrival times:");

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

arrival[i] = scanner.nextInt();

}

System.out.println("Enter departure times:");

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

departure[i] = scanner.nextInt();

}

int result = findMinPlatforms(arrival, departure);

System.out.println("Minimum platforms required: " + result);

scanner.close();

}

}

Output:

Enter number of trains: 6

Enter arrival times:

900 940 950 1100 1500 1800

Enter departure times:

910 1200 1120 1130 1900 2000

Minimum platforms required: 3

25b)

import java.util.\*;

public class Program25b {

public static void bfs(int startNode, List<List<Integer>> adjList, int V) {

boolean[] visited = new boolean[V];

Queue<Integer> queue = new LinkedList<>();

visited[startNode] = true;

queue.add(startNode);

System.out.println("BFS Traversal:");

while (!queue.isEmpty()) {

int node = queue.poll();

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

for (int neighbor : adjList.get(node)) {

if (!visited[neighbor]) {

visited[neighbor] = true;

queue.add(neighbor);

}

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of vertices: ");

int V = scanner.nextInt();

System.out.print("Enter number of edges: ");

int E = scanner.nextInt();

List<List<Integer>> adjList = new ArrayList<>();

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

adjList.add(new ArrayList<Integer>());

}

System.out.println("Enter edges (format: u v):");

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

int u = scanner.nextInt();

int v = scanner.nextInt();

adjList.get(u).add(v);

adjList.get(v).add(u); // Assuming an undirected graph

}

System.out.print("Enter the starting node for BFS: ");

int startNode = scanner.nextInt();

bfs(startNode, adjList, V);

scanner.close();

}

}

Output:

Enter number of vertices: 5

Enter number of edges: 4

Enter edges:

0 1

0 2

1 3

2 4

Enter the starting node for BFS: 0

BFS Traversal:

0 1 2 3 4

set-26

a)

import java.util.Scanner;

public class Program26a {

public static String alternateStrings(String str1, String str2) {

StringBuilder result = new StringBuilder();

int len1 = str1.length(), len2 = str2.length();

int maxLen = Math.max(len1, len2);

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

if (i < len1) result.append(str1.charAt(i));

if (i < len2) result.append(str2.charAt(i));

}

return result.toString();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter first string: ");

String str1 = scanner.nextLine();

System.out.print("Enter second string: ");

String str2 = scanner.nextLine();

String result = alternateStrings(str1, str2);

System.out.println("Alternating merged string: " + result);

scanner.close();

}

}

Output:

Enter first string: ABC

Enter second string: 123

Alternating merged string: A1B2C3

Enter first string: Hello

Enter second string: 123

Alternating merged string: H1e2l3lo

26b)

import java.util.\*;

public class Program26b {

public static boolean isBalanced(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()) return false;

char top = stack.pop();

if ((ch == ')' && top != '(') ||

(ch == '}' && top != '{') ||

(ch == ']' && top != '[')) {

return false;

}

}

}

return stack.isEmpty();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a string with parentheses: ");

String input = scanner.nextLine();

boolean result = isBalanced(input);

System.out.println("Balanced: " + result);

scanner.close();

}

}

Output:

Enter a string with parentheses: ({)

Balanced: false

Enter a string with parentheses: ({})

Balanced: true

Enter a string with parentheses: {[()]}

Balanced: true

Enter a string with parentheses: ({[)])

Balanced: false

set-27

a)

import java.util.\*;

public class Program27a {

public static List<List<Integer>> fourSum(int[] nums, int target) {

List<List<Integer>> result = new ArrayList<>();

Arrays.sort(nums); // Sort the array

int n = nums.length;

for (int i = 0; i < n - 3; i++) {

if (i > 0 && nums[i] == nums[i - 1]) continue;

for (int j = i + 1; j < n - 2; j++) {

if (j > i + 1 && nums[j] == nums[j - 1]) continue;

int left = j + 1, right = n - 1;

while (left < right) {

int sum = nums[i] + nums[j] + nums[left] + nums[right];

if (sum == target) {

result.add(Arrays.asList(nums[i], nums[j], nums[left], nums[right]));

while (left < right && nums[left] == nums[left + 1]) left++; // Skip duplicates

while (left < right && nums[right] == nums[right - 1]) right--; // Skip duplicates

left++;

right--;

} else if (sum < target) {

left++;

} else {

right--;

}

}

}

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements: ");

int n = scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter array elements:");

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

nums[i] = scanner.nextInt();

}

System.out.print("Enter target sum: ");

int target = scanner.nextInt();

List<List<Integer>> result = fourSum(nums, target);

System.out.println("Quadruplets with sum " + target + ": " + result);

scanner.close();

}

}

Output:

Enter number of elements: 6

Enter array elements:

1 0 -1 0 -2 2

Enter target sum: 0

Quadruplets with sum 0: [[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]

27b)

import java.util.\*;

public class Program27b {

public static void deleteMiddle(Stack<Integer> stack, int midIndex) {

if (stack.size() == midIndex + 1) {

stack.pop();

return;

}

int temp = stack.pop();

deleteMiddle(stack, midIndex);

stack.push(temp);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

System.out.print("Enter number of elements in the stack: ");

int n = scanner.nextInt();

System.out.println("Enter stack elements:");

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

stack.push(scanner.nextInt());

}

int midIndex = n / 2;

deleteMiddle(stack, midIndex);

System.out.println("Stack after deleting middle element: " + stack);

scanner.close();

}

}

Output:

Enter number of elements in the stack: 5

Enter stack elements:

1 2 3 4 5

Stack after deleting middle element: [1, 2, 4, 5]

set-28

a)

import java.util.\*;

public class Program28a {

public static int findMaxInQueue(Queue<Integer> queue) {

PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());

for (int num : queue) {

maxHeap.add(num);

}

return maxHeap.peek();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Queue<Integer> queue = new LinkedList<>();

System.out.print("Enter number of elements in the queue: ");

int n = scanner.nextInt();

System.out.println("Enter queue elements:");

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

queue.add(scanner.nextInt());

}

int maxElement = findMaxInQueue(queue);

System.out.println("Maximum element in queue: " + maxElement);

scanner.close();

}

}

Output:

Enter number of elements in the queue: 6

Enter queue elements:

3 1 7 2 9 5

Maximum element in queue: 9

28b)

import java.util.\*;

public class Program28b {

static class StackUsingQueue {

Queue<Integer> queue1 = new LinkedList<>();

Queue<Integer> queue2 = new LinkedList<>();

public void push(int x) {

queue1.add(x);

}

public int pop() {

if (queue1.isEmpty()) {

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

return -1;

}

while (queue1.size() > 1) {

queue2.add(queue1.poll());

}

int popped = queue1.poll();

Queue<Integer> temp = queue1;

queue1 = queue2;

queue2 = temp;

return popped;

}

public int top() {

if (queue1.isEmpty()) {

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

return -1;

}

while (queue1.size() > 1) {

queue2.add(queue1.poll());

}

int topElement = queue1.peek();

queue2.add(queue1.poll());

Queue<Integer> temp = queue1;

queue1 = queue2;

queue2 = temp;

return topElement;

}

public boolean isEmpty() {

return queue1.isEmpty();

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

StackUsingQueue stack = new StackUsingQueue();

System.out.print("Enter number of elements to push into stack: ");

int n = scanner.nextInt();

System.out.println("Enter elements:");

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

stack.push(scanner.nextInt());

}

System.out.println("Top element: " + stack.top());

System.out.println("Popped element: " + stack.pop());

System.out.println("Top element after pop: " + stack.top());

scanner.close();

}

}

Output:

Enter number of elements to push into stack: 5

Enter elements:

10 20 30 40 50

Top element: 50

Popped element: 50

Top element after pop: 40

set-29

a)

import java.util.\*;

public class Program29a {

public static int uniquePathsWithObstacles(int[][] grid) {

int m = grid.length, n = grid[0].length;

if (grid[0][0] == 1 || grid[m - 1][n - 1] == 1) return 0;

int[][] dp = new int[m][n];

dp[0][0] = 1;

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

for (int j = 0; j < n; j++) {

if (grid[i][j] == 1) {

dp[i][j] = 0;

} else {

if (i > 0) dp[i][j] += dp[i - 1][j];

if (j > 0) dp[i][j] += dp[i][j - 1];

}

}

}

return dp[m - 1][n - 1];

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of rows: ");

int m = scanner.nextInt();

System.out.print("Enter number of columns: ");

int n = scanner.nextInt();

int[][] grid = new int[m][n];

System.out.println("Enter grid (0 for open cell, 1 for obstacle):");

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

for (int j = 0; j < n; j++) {

grid[i][j] = scanner.nextInt();

}

}

int result = uniquePathsWithObstacles(grid);

System.out.println("Number of unique paths: " + result);

scanner.close();

}

}

Output:

Enter number of rows: 3

Enter number of columns: 3

Enter grid:

0 0 0

0 1 0

0 0 0

Number of unique paths: 2

29b)

import java.util.\*;

public class Program29b {

public static void reverseQueue(Queue<Integer> queue) {

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

while (!queue.isEmpty()) {

stack.push(queue.poll());

}

while (!stack.isEmpty()) {

queue.add(stack.pop());

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Queue<Integer> queue = new LinkedList<>();

System.out.print("Enter number of elements in the queue: ");

int n = scanner.nextInt();

System.out.println("Enter queue elements:");

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

queue.add(scanner.nextInt());

}

reverseQueue(queue);

System.out.println("Queue after reversal: " + queue);

scanner.close();

}

}

Output:

Enter number of elements in the queue: 5

Enter queue elements:

1 2 3 4 5

Queue after reversal: [5, 4, 3, 2, 1]

set-30

a)

import java.util.\*;

public class Program30a {

public static int[] removeDuplicates(int[] nums) {

LinkedHashSet<Integer> set = new LinkedHashSet<>();

for (int num : nums) {

set.add(num);

}

int[] uniqueArray = new int[set.size()];

int i = 0;

for (int num : set) {

uniqueArray[i++] = num;

}

return uniqueArray;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements: ");

int n = scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter array elements:");

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

nums[i] = scanner.nextInt();

}

int[] uniqueArray = removeDuplicates(nums);

System.out.println("Unique array: " + Arrays.toString(uniqueArray));

System.out.println("New length: " + uniqueArray.length);

scanner.close();

}

}

Output:

Enter number of elements: 7

Enter array elements:

1 2 2 3 4 4 5

Unique array: [1, 2, 3, 4, 5]

New length: 5

30b)

import java.util.\*;

public class Program30b {

static class MinStack {

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

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

public void push(int x) {

mainStack.push(x);

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

minStack.push(x);

}

}

public void pop() {

if (!mainStack.isEmpty()) {

int popped = mainStack.pop();

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

minStack.pop();

}

}

}

public int getMin() {

return minStack.isEmpty() ? Integer.MAX\_VALUE : minStack.peek();

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

MinStack stack = new MinStack();

System.out.print("Enter number of elements in the stack: ");

int n = scanner.nextInt();

System.out.println("Enter stack elements:");

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

stack.push(scanner.nextInt());

}

System.out.println("Minimum element in stack: " + stack.getMin());

scanner.close();

}

}

Output:

Enter number of elements in the stack: 5

Enter stack elements:

3 5 2 1 4

Minimum element in stack: 1

set-31

a)

import java.util.Scanner;

public class Program31a {

public static int findMissingTerm(int[] sequence) {

int n = sequence.length;

int r = sequence[1] / sequence[0];

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

if (sequence[i] != sequence[i - 1] \* r) {

return sequence[i - 1] \* r;

}

}

return -1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of terms: ");

int n = scanner.nextInt();

int[] sequence = new int[n];

System.out.println("Enter logarithmic sequence (one missing term):");

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

sequence[i] = scanner.nextInt();

}

int missingTerm = findMissingTerm(sequence);

System.out.println("Missing term: " + missingTerm);

scanner.close();

}

}

Output:

Enter number of terms: 5

Enter logarithmic sequence:

2 6 \_ 54 162

Missing term: 18

31b)

import java.util.\*;

public class Program31b {

static class MinStack {

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

int minElement;

public void push(int x) {

if (stack.isEmpty()) {

minElement = x;

stack.push(x);

} else {

if (x < minElement) {

stack.push(2 \* x - minElement);

minElement = x;

} else {

stack.push(x);

}

}

}

public void pop() {

if (!stack.isEmpty()) {

int popped = stack.pop();

if (popped < minElement) {

minElement = 2 \* minElement - popped;

}

}

}

public int getMin() {

return stack.isEmpty() ? Integer.MAX\_VALUE : minElement;

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

MinStack stack = new MinStack();

System.out.print("Enter number of elements in the stack: ");

int n = scanner.nextInt();

System.out.println("Enter stack elements:");

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

stack.push(scanner.nextInt());

}

System.out.println("Minimum element in stack: " + stack.getMin());

scanner.close();

}

}

Output:

Enter number of elements in the stack: 5

Enter stack elements:

3 5 2 1 4

Minimum element in stack: 1

32) A) Intersection of two arrays in log n time complexity

import java.util.Arrays;

public class Intersection {

public static int[] intersection(int[] arr1, int[] arr2) {

Arrays.sort(arr1);

Arrays.sort(arr2);

int i = 0;

int j = 0;

int k = 0;

int[] result = new int[Math.min(arr1.length, arr2.length)];

while (i < arr1.length && j < arr2.length) {

if (arr1[i] < arr2[j]) {

i++;

} else if (arr1[i] > arr2[j]) {

j++;

} else {

result[k++] = arr1[i];

i++;

j++;

}

}

return Arrays.copyOfRange(result, 0, k);

}

public static void main(String[] args) {

int[] arr1 = {1, 2, 3, 4, 5};

int[] arr2 = {4, 5, 6, 7, 8};

int[] result = intersection(arr1, arr2);

System.out.println("Intersection: " + Arrays.toString(result));

}

}

B) Find a string is palindrome or not using stack and queue

import java.util.Stack;

import java.util.Queue;

import java.util.LinkedList;

public class Palindrome {

public static boolean isPalindrome(String str) {

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

Queue<Character> queue = new LinkedList<>();

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

stack.push(c);

queue.add(c);

}

while (!stack.isEmpty()) {

if (stack.pop() != queue.poll()) {

return false;

}

}

return true;

}

public static void main(String[] args) {

String str = "madam";

System.out.println("Is palindrome: " + isPalindrome(str));

}

}

33a)

import java.util.\*;

public class ArrayIntersection {

public static Set<Integer> findIntersection(int[] arr1, int[] arr2) {

Set<Integer> set1 = new HashSet<>();

Set<Integer> result = new HashSet<>();

for (int num : arr1) {

set1.add(num);

}

for (int num : arr2) {

if (set1.contains(num)) {

result.add(num);

}

}

return result;

}

public static void main(String[] args) {

int[] arr1 = {7, 1, 5, 2, 3, 6};

int[] arr2 = {3, 8, 6, 20, 7};

Set<Integer> intersection = findIntersection(arr1, arr2);

System.out.println("Intersection: " + intersection);

}

}

33b)

import java.util.\*;

public class PalindromeCheck {

public static boolean isPalindrome(String str) {

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

Queue<Character> queue = new LinkedList<>();

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

stack.push(c);

queue.offer(c);

}

while (!stack.isEmpty()) {

if (stack.pop() != queue.poll()) {

return false;

}

}

return true;

}

public static void main(String[] args) {

String str = "madam";

System.out.println("Is palindrome: " + isPalindrome(str));

}

}

set-34

a)

import java.util.Scanner;

public class Program34a {

public static int ternarySearch(int[] arr, int low, int high, int target) {

if (low > high) return -1;

int mid1 = low + (high - low) / 3;

int mid2 = high - (high - low) / 3;

if (arr[mid1] == target) return mid1;

if (arr[mid2] == target) return mid2;

if (target < arr[mid1]) {

return ternarySearch(arr, low, mid1 - 1, target);

} else if (target > arr[mid2]) {

return ternarySearch(arr, mid2 + 1, high, target);

} else {

return ternarySearch(arr, mid1 + 1, mid2 - 1, target);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter sorted array elements:");

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

arr[i] = scanner.nextInt();

}

System.out.print("Enter target element: ");

int target = scanner.nextInt();

int result = ternarySearch(arr, 0, n - 1, target);

System.out.println(result != -1 ? "Element found at index: " + result : "Element not found");

scanner.close();

}

}

Output:

Enter number of elements: 6

Enter sorted array elements:

1 2 3 4 5 6

Enter target element: 4

Element found at index: 3

34b)

import java.util.\*;

class ListNode {

int val;

ListNode next;

ListNode(int val) { this.val = val; this.next = null; }

}

public class Program34b {

public static boolean isPalindrome(ListNode head) {

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

ListNode temp = head;

while (temp != null) {

stack.push(temp.val);

temp = temp.next;

}

temp = head;

while (temp != null) {

if (temp.val != stack.pop()) return false;

temp = temp.next;

}

return true;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements in linked list: ");

int n = scanner.nextInt();

System.out.println("Enter linked list elements:");

ListNode head = null, tail = null;

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

int value = scanner.nextInt();

ListNode newNode = new ListNode(value);

if (head == null) {

head = newNode;

} else {

tail.next = newNode;

}

tail = newNode;

}

System.out.println("Is the linked list a palindrome? " + isPalindrome(head));

scanner.close();

}

}

Output:

Enter number of elements in linked list: 5

Enter linked list elements:

1 2 3 2 1

Is the linked list a palindrome? True

set-35

a)

import java.util.\*;

public class Program35a {

public static int[] sortedSquares(int[] arr) {

int n = arr.length;

int[] result = new int[n];

int left = 0, right = n - 1;

int index = n - 1;

while (left <= right) {

int leftSquare = arr[left] \* arr[left];

int rightSquare = arr[right] \* arr[right];

if (leftSquare > rightSquare) {

result[index--] = leftSquare;

left++;

} else {

result[index--] = rightSquare;

right--;

}

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter sorted array elements:");

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

arr[i] = scanner.nextInt();

}

int[] result = sortedSquares(arr);

System.out.println("Sorted array after squaring: " + Arrays.toString(result));

scanner.close();

}

}

Output:

Enter number of elements: 6

Enter sorted array elements: -4 -2 0 1 3 5

Sorted array after squaring: [0, 1, 4, 9, 16, 25]

35b)

import java.util.\*;

class ListNode {

int val;

ListNode next;

ListNode(int val) { this.val = val; this.next = null; }

}

public class Program35b {

public static ListNode removeNthFromEnd(ListNode head, int n) {

ListNode dummy = new ListNode(0);

dummy.next = head;

ListNode slow = dummy, fast = dummy;

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

fast = fast.next;

}

while (fast != null) {

slow = slow.next;

fast = fast.next;

}

slow.next = slow.next.next;

return dummy.next;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements in linked list: ");

int n = scanner.nextInt();

System.out.println("Enter linked list elements:");

ListNode head = null, tail = null;

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

int value = scanner.nextInt();

ListNode newNode = new ListNode(value);

if (head == null) {

head = newNode;

} else {

tail.next = newNode;

}

tail = newNode;

}

System.out.print("Enter Nth element to remove from end: ");

int removePosition = scanner.nextInt();

head = removeNthFromEnd(head, removePosition);

System.out.print("Linked list after deletion: ");

ListNode temp = head;

while (temp != null) {

System.out.print(temp.val + " ");

temp = temp.next;

}

scanner.close();

}

}

Output:

Enter number of elements in linked list: 5

Enter linked list elements:

1 2 3 4 5

Enter Nth element to remove from end: 2

Linked list after deletion: 1 2 3 5

set-36

a)

import java.util.\*;

public class Program36a {

public static void paritySort(int[] arr) {

int left = 0, right = arr.length - 1;

while (left < right) {

if (arr[left] % 2 == 0) {

left++;

} else if (arr[right] % 2 != 0) {

right--;

} else {

int temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

left++;

right--;

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter array elements:");

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

arr[i] = scanner.nextInt();

}

paritySort(arr);

System.out.println("Sorted array (Even first, Odd last): " + Arrays.toString(arr));

scanner.close();

}

}

Output:

Enter number of elements: 6

Enter array elements:

3 1 2 4 5 6

Sorted array (Even first, Odd last): [6, 4, 2, 1, 5, 3]

set-36

b)

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) { this.val = val; }

}

public class Program36b {

public static TreeNode insert(TreeNode root, int val) {

if (root == null) return new TreeNode(val);

if (val < root.val) root.left = insert(root.left, val);

else root.right = insert(root.right, val);

return root;

}

public static int diameter(TreeNode root) {

int[] diameter = new int[1];

height(root, diameter);

return diameter[0];

}

private static int height(TreeNode node, int[] diameter) {

if (node == null) return 0;

int leftHeight = height(node.left, diameter);

int rightHeight = height(node.right, diameter);

diameter[0] = Math.max(diameter[0], leftHeight + rightHeight);

return 1 + Math.max(leftHeight, rightHeight);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

TreeNode root = null;

System.out.print("Enter number of nodes in BST: ");

int n = scanner.nextInt();

System.out.println("Enter BST elements:");

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

int value = scanner.nextInt();

root = insert(root, value);

}

int result = diameter(root);

System.out.println("Diameter of the BST: " + result);

scanner.close();

}

}

Output:

Enter number of nodes in BST: 6

Enter BST elements:

5 3 8 2 4 9

Diameter of the BST: 3

set-37

a)

import java.util.\*;

public class Program37a {

public static int findLIS(int[] arr) {

int n = arr.length;

int[] dp = new int[n];

Arrays.fill(dp, 1);

int maxLIS = 1;

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

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

if (arr[i] > arr[j]) {

dp[i] = Math.max(dp[i], dp[j] + 1);

}

}

maxLIS = Math.max(maxLIS, dp[i]);

}

return maxLIS;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter array elements:");

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

arr[i] = scanner.nextInt();

}

int result = findLIS(arr);

System.out.println("Length of Longest Increasing Subsequence: " + result);

scanner.close();

}

}

Output:

Enter number of elements: 6

Enter array elements:

10 22 9 33 21 50

Length of Longest Increasing Subsequence: 4

37b)

import java.util.\*;

class TrieNode {

Map<Character, TrieNode> children = new HashMap<>();

boolean isEndOfWord;

}

public class Program37b {

private TrieNode root;

public Program37b() {

root = new TrieNode();

}

public void insert(String word) {

TrieNode node = root;

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

node.children.putIfAbsent(ch, new TrieNode());

node = node.children.get(ch);

}

node.isEndOfWord = true;

}

public String longestCommonPrefix() {

TrieNode node = root;

StringBuilder prefix = new StringBuilder();

while (node.children.size() == 1 && !node.isEndOfWord) {

char ch = node.children.keySet().iterator().next();

prefix.append(ch);

node = node.children.get(ch);

}

return prefix.toString();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Program37b trie = new Program37b();

System.out.print("Enter number of words: ");

int n = scanner.nextInt();

scanner.nextLine();

System.out.println("Enter words:");

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

trie.insert(scanner.nextLine());

}

System.out.println("Longest Common Prefix: " + trie.longestCommonPrefix());

scanner.close();

}

}

Output:

Enter number of words: 4

Enter words:

flower

flow

flight

flame

Longest Common Prefix: fl

set-38

a)

Longest Bitonic Subsequence in Java:

public class LongestBitonicSubsequence {

public static int longestBitonicSubsequence(int[] nums) {

int n = nums.length;

// Calculate LIS for each element

int[] lis = new int[n];

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

lis[i] = 1;

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

if (nums[i] > nums[j]) {

lis[i] = Math.max(lis[i], lis[j] + 1);

}

}

}

// Calculate LDS for each element

int[] lds = new int[n];

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

lds[i] = 1;

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

if (nums[i] > nums[j]) {

lds[i] = Math.max(lds[i], lds[j] + 1);

}

}

}

// Find the maximum length of bitonic subsequence

int maxLength = 0;

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

if (lis[i] > 1 && lds[i] > 1) {

maxLength = Math.max(maxLength, lis[i] + lds[i] - 1);

}

}

return maxLength > 0 ? maxLength : 0;

}

public static void main(String[] args) {

int[] nums = {1, 2, 5, 3, 2};

System.out.println("Length of longest bitonic subsequence: " + longestBitonicSubsequence(nums));

}

}

b)

Simple Solution using Substring:

public class QueryStringOccurrences {

public static int[] countOccurrences(String[] A, String[] Q) {

int[] result = new int[Q.length];

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

String query = Q[i];

for (String str : A) {

int count = 0, index = 0;

while (index < str.length()) {

if (str.substring(index).startsWith(query)) {

count++;

index += query.length();

} else {

index++;

}

}

result[i] += count;

}

}

return result;

}

public static void main(String[] args) {

String[] A = {"banana", "bandana", "cabana"};

String[] Q = {"ana"};

int[] result = countOccurrences(A, Q);

for (int count : result) {

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

}

}

}

set-39

a)import java.util.\*;

public class LongestDecreasingSubsequence {

public static int longestDecreasingSubsequence(int[] arr) {

int n = arr.length;

int[] dp = new int[n];

Arrays.fill(dp, 1);

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

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

if (arr[j] > arr[i]) {

dp[i] = Math.max(dp[i], dp[j] + 1);

}

}

}

int max = 0;

for (int value : dp) {

if (value > max) max = value;

}

return max;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter size of array: ");

int n = sc.nextInt();

int[] arr = new int[n];

System.out.print("Enter array elements: ");

for (int i = 0; i < n; i++) arr[i] = sc.nextInt();

int result = longestDecreasingSubsequence(arr);

System.out.println("Length of longest decreasing subsequence: " + result);

}

}

b)

import java.util.\*;

public class StringFrequencySort {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of strings: ");

int n = sc.nextInt();

sc.nextLine();

String[] arr = new String[n];

System.out.println("Enter strings:");

for (int i = 0; i < n; i++) arr[i] = sc.nextLine();

Map<String, Integer> freqMap = new HashMap<>();

for (String s : arr) freqMap.put(s, freqMap.getOrDefault(s, 0) + 1);

Set<String> unique = new HashSet<>(Arrays.asList(arr));

List<String> result = new ArrayList<>(unique);

Collections.sort(result, (a, b) -> {

int f1 = freqMap.get(a);

int f2 = freqMap.get(b);

if (f1 != f2) return f1 - f2;

return a.compareTo(b);

});

System.out.println("Sorted strings by frequency:");

for (String s : result) System.out.println(s);

}

}

set-40

40a)Basic Stock Buying and Selling (Conceptual)

import java.util.Scanner;

public class StockTrader {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String stockSymbol;

int quantity;

double purchasePrice = 0.0;

boolean hasStock = false;

System.out.println("Simple Stock Trading Simulation");

while (true) {

System.out.println("\nOptions:");

System.out.println("1. Buy Stock");

System.out.println("2. Sell Stock");

System.out.println("3. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline

switch (choice) {

case 1:

if (hasStock) {

System.out.println("You already own stock. Sell before buying more.");

break;

}

System.out.print("Enter the stock symbol: ");

stockSymbol = scanner.nextLine();

System.out.print("Enter the quantity to buy: ");

quantity = scanner.nextInt();

System.out.print("Enter the purchase price per share: ");

purchasePrice = scanner.nextDouble();

scanner.nextLine(); // Consume newline

System.out.println("Bought " + quantity + " shares of " + stockSymbol + " at $" + purchasePrice + " per share.");

hasStock = true;

break;

case 2:

if (!hasStock) {

System.out.println("You don't own any stock to sell.");

break;

}

System.out.print("Enter the selling price per share: ");

double sellingPrice = scanner.nextDouble();

scanner.nextLine(); // Consume newline

double profit = (sellingPrice - purchasePrice) \* quantity;

System.out.println("Sold " + quantity + " shares of the stock for $" + sellingPrice + " per share.");

System.out.println("Profit/Loss: $" + String.format("%.2f", profit)); // Format to 2 decimal places

hasStock = false;

purchasePrice = 0.0; // Reset purchase price

break;

case 3:

System.out.println("Exiting the simulation.");

scanner.close();

return;

default:

System.out.println("Invalid choice. Please try again.");

}

}

}

}

40b)

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) { this.val = val; }

}

public class Program41b {

public static int countHalfNodes(TreeNode root) {

if (root == null) return 0;

int count = 0;

if ((root.left == null && root.right != null) || (root.left != null && root.right == null)) {

count = 1;

}

return count + countHalfNodes(root.left) + countHalfNodes(root.right);

}

public static TreeNode insert(TreeNode root, int val) {

if (root == null) return new TreeNode(val);

if (val < root.val) root.left = insert(root.left, val);

else root.right = insert(root.right, val);

return root;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

TreeNode root = null;

System.out.print("Enter number of nodes in BST: ");

int n = scanner.nextInt();

System.out.println("Enter BST elements:");

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

int value = scanner.nextInt();

root = insert(root, value);

}

int result = countHalfNodes(root);

System.out.println("Number of half nodes in BST: " + result);

scanner.close();

}

}

Output:

Enter number of nodes in BST: 6

Enter BST elements:

5 3 8 2 4 9

Number of half nodes in BST: 1

set-41

a)

import java.util.\*;

public class Program41a {

public static int maxProfit(int[] prices) {

if (prices.length == 0) return 0;

int minPrice = Integer.MAX\_VALUE, maxProfit = 0;

for (int price : prices) {

minPrice = Math.min(minPrice, price);

maxProfit = Math.max(maxProfit, price - minPrice);

}

return maxProfit;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of days: ");

int n = scanner.nextInt();

int[] prices = new int[n];

System.out.println("Enter stock prices:");

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

prices[i] = scanner.nextInt();

}

int result = maxProfit(prices);

System.out.println("Maximum profit: " + result);

scanner.close();

}

}

Output:

Enter number of days: 6

Enter stock prices:

7 1 5 3 6 4

Maximum profit: 5

41a)

import java.util.Scanner;

public class Program41a {

public static int maxSubArraySum(int[] nums) {

int maxSoFar = nums[0];

int currentMax = nums[0];

for (int i = 1; i < nums.length; i++) {

currentMax = Math.max(nums[i], currentMax + nums[i]);

maxSoFar = Math.max(maxSoFar, currentMax);

}

return maxSoFar;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of elements: ");

int n = scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter array elements:");

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

nums[i] = scanner.nextInt();

}

int result = maxSubArraySum(nums);

System.out.println("Maximum subarray sum: " + result);

scanner.close();

}

}

41b)

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) { this.val = val; }

}

public class Program41b {

public static TreeNode insert(TreeNode root, int val) {

if (root == null) return new TreeNode(val);

if (val < root.val) root.left = insert(root.left, val);

else root.right = insert(root.right, val);

return root;

}

public static List<Integer> rightView(TreeNode root) {

List<Integer> result = new ArrayList<>();

if (root == null) return result;

Queue<TreeNode> queue = new LinkedList<>();

queue.offer(root);

while (!queue.isEmpty()) {

int size = queue.size();

TreeNode rightmost = null;

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

TreeNode current = queue.poll();

if (i == size - 1) rightmost = current;

if (current.left != null) queue.offer(current.left);

if (current.right != null) queue.offer(current.right);

}

result.add(rightmost.val);

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

TreeNode root = null;

System.out.print("Enter number of nodes in BST: ");

int n = scanner.nextInt();

System.out.println("Enter BST elements:");

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

int value = scanner.nextInt();

root = insert(root, value);

}

List<Integer> view = rightView(root);

System.out.println("Right view of BST: " + view);

scanner.close();

}

}

Questions:

Set-1

1a.

Write a Java program to implement a queue using two stacks. Include operations to enqueue, dequeue, check if the queue is empty, and display the queue contents.

1b.

Write a Java program to find the minimum element in a rotated sorted array using a modified binary search.

Set-2

2a.

Write a Java program that, for a given array, generates all (value, minimum) pairs for each rotation of the array and counts the total number of such pairs where value > minimum.

2b.

Write a Java program to implement a stack using two queues. Include operations for push, pop, top, display, and check if the stack is empty.

Set-3

3a.

Write a Java program to remove duplicates from an unsorted linked list using a HashSet.

3b.

Write a Java program to search an element in a rotated sorted array using binary search. If found, return its index; otherwise, return -1.

Set-4

4a.

Write a Java program to delete the nth node from the end of a singly linked list.

4b.

Write a Java program to find the ceiling and floor of a given number x in a floating-point array.

Set-5

5a.

Write a Java program to find the element that occurs only once in a sorted array where every other element appears exactly twice.

5b.

Write a Java program to reverse a singly linked list and display both the original and the reversed lists.

Set-6

6a) Two Sum (HashMap Approach)

Question:

Write a Java program to find the indices of two numbers in an array that add up to a given target using a HashMap.

6b) Search in Linked List (Iterative)

Question:

Write a Java program to perform an iterative search for a given element in a singly linked list.

6c) Search in Linked List (Recursive)

Question:

Write a Java program to perform a recursive search for a given element in a singly linked list.

Set-7

7a) Remove Duplicates and Sort (TreeSet)

Question:

Write a Java program to remove duplicates from an array and print the elements in sorted order using a TreeSet.

7b) Reverse a Queue (Recursion)

Question:

Write a Java program to reverse a queue using recursion.

Set-8

8a) Remove Consecutive Duplicates in String

Question:

Write a Java program to remove consecutive duplicate characters from a string.

8b) Sort a Stack Using Another Stack

Question:

Write a Java program to sort a stack using only another stack as auxiliary space.

Set-9

9a) Equilibrium Index in Array

Question:

Write a Java program to find the equilibrium index of an array where the sum of elements on the left is equal to the sum on the right.

9b) Diameter of Binary Search Tree

Question:

Write a Java program to construct a Binary Search Tree (BST) and find its diameter (the length of the longest path between any two nodes).

Set-10

10a) Mirror of a BST

Question:

Write a Java program to create the mirror of a Binary Search Tree and print its inorder traversal before and after mirroring.

10b) Minimum Operations to Convert Arrays

Question:

Write a Java program to calculate the minimum number of operations to convert one array to another. An operation is defined as incrementing or decrementing an element by 1.

Set-11

Total Cost of Connections: Given a set of ropes with different lengths, calculate the total cost to connect all ropes. The cost to connect two ropes is the sum of their lengths, and the total cost is the sum of all connection costs. (Use a priority queue/heap to solve this efficiently.)

Remove Half Nodes from a BST: Given a binary search tree, remove all half nodes. A half node is a node with only one child. Implement this and print the inorder traversal of the tree before and after removing half nodes.

Set-12

Print Left Nodes of a BST: Given a binary search tree (BST), print the data of all left child nodes.

Product of Array Elements Without Division: Given an array, calculate a new array where each element is the product of all the elements in the original array except the current element, without using division.

Set-13

Assign Cookies to Children: Given an array representing children's greed factors and another array representing cookie sizes, determine how many children can be satisfied with the available cookies. Each cookie can satisfy one child if its size is greater than or equal to the child's greed factor.

Width of a Binary Search Tree at a Given Level: Given a binary search tree, calculate the width of the tree at a specific level. The width at a level is the number of nodes at that level.

Set-14

Lemonade Stand Simulation: Implement a simulation where customers arrive with different payment amounts, and the lemonade stand provides lemonade and change. The stand has a limited amount of change and can only serve a customer if it has enough change for the transaction.

Height of a Binary Search Tree: Given a binary search tree, calculate the height of the tree. The height of a tree is the number of edges on the longest path from the root to a leaf.

Set-15

Maximum Equal Sum Among 3 Stacks: Given three stacks with positive integers, find the maximum equal sum that can be achieved by removing elements from the stacks. You can only remove elements from the top of the stacks.

Max Width of a Binary Search Tree: Given a binary search tree, find its maximum width. The width of a binary tree at any level is the number of nodes at that level, and the maximum width is the highest width among all levels of the tree.

Set-16

Reverse level order traversal in BST

Knapsack problem (0/1 Knapsack using Dynamic Programming)

Set-17

Count all root-to-leaf paths in a binary tree.

Breadth-first traversal of a binary search tree (BST).

Set-18

Kadane's Algorithm for finding the maximum sum subarray.

Inorder, Preorder, and Postorder Traversals of a binary tree.

Set-19

Reverse characters of a string using two pointers technique.

Implement two stacks in one array (Dynamic implementation).

Set-20

Find a triplet in an array such that the sum of two elements equals the third element.

Implement Depth-First Search (DFS) using an adjacency list without recursion.

set-21

21a. Implement DFS Traversal in an Undirected Graph

Write a program that takes the number of vertices and edges of an undirected graph, followed by the edges themselves. Perform a DFS traversal starting from node 0 and print the visited nodes.

21b. DFS Traversal in an Undirected Graph

Modify the program in question 1a to perform DFS traversal starting from any given node (not just node 0).

set-22

22a. Count the Number of Ways to Climb Stairs

Write a program that calculates the number of ways to climb a staircase where at each step, you can climb either 1, 2, or 3 stairs. The program should take the number of stairs n as input.

22b. DFS Traversal in an Undirected Graph

Implement DFS traversal in an undirected graph, where the number of vertices and edges is provided, and the edges are entered in the format u v. The traversal should start from node 0.

22b. dfs without recurssion

set-23

23a. Longest Common Subsequence

Write a program that computes the length of the longest common subsequence (LCS) of two strings. The program should take two strings as input and output the length of their LCS.

23b. DFS Traversal in an Undirected Graph (Custom Start)

Implement a DFS traversal where the starting node for the traversal is provided by the user as input. The program should take the number of vertices and edges as input, along with the edges, and start DFS from the specified node.

set-24

24a. Find Triplets with Sum Zero

Write a program to find all unique triplets in an array that sum to zero. The program should take the number of elements and the elements of the array as input.

24b. BFS Traversal in an Undirected Graph

Implement BFS traversal in an undirected graph. The program should take the number of vertices, the number of edges, and the edges as input, and it should start the BFS traversal from a specified node.

set-25

25a. Find Minimum Platforms for Trains

Write a program to find the minimum number of platforms required to accommodate a set of trains given their arrival and departure times. The program should take the number of trains, along with their arrival and departure times, as input.

25b. BFS Traversal in an Undirected Graph (Custom Start)

Modify the BFS traversal program to allow the user to specify the starting node for the BFS traversal. The program should take the number of vertices, the number of edges, and the edges as input, along with the starting node.

set-26

1a. Write a Java program to check if the parentheses in a given string are balanced.

1b. Implement a program that checks if the parentheses in a given string are balanced using a stack.

set-27

1a. Write a Java program to find all quadruplets in an array that sum to a given target.

1b. Implement a program to delete the middle element of a stack using recursion.

set-28

1a. Write a Java program to find the maximum element in a queue.

1b. Implement a stack using two queues and provide methods to push, pop, top, and check if the stack is empty.

set-29

1a. Write a Java program to find the number of unique paths in a grid with obstacles, where you can only move right or down.

1b. Implement a program to reverse a queue using a stack.

set-30

1a. Write a Java program to remove duplicates from a given array.

1b. Implement a MinStack that supports push, pop, and retrieving the minimum element efficiently.

set-31

1a. Write a Java program to find the missing term in a logarithmic sequence where each term is multiplied by a constant ratio. The sequence will have one missing term.

1b. Write a Java program to implement a stack that supports retrieving the minimum element in constant time. Implement the push, pop, and getMin operations.

set-32

1a. Write a Java program to find the intersection of two arrays in O(log n) time complexity using binary search.

1b. Write a Java program to check if a string is a palindrome using a stack and a queue.

set-33

1a. Write a Java program to find the intersection of two arrays using a HashSet.

1b. Write a Java program to check if a string is a palindrome using a stack and a queue.

set-34

1a. Write a Java program to implement a ternary search algorithm for finding a target element in a sorted array.

1b. Write a Java program to check if a linked list is a palindrome using a stack.

set-35

1a. Write a Java program to return a sorted array of squares of the elements from a given sorted array.

1b. Write a Java program to remove the Nth node from the end of a singly linked list.

Set-36

36a. Diameter of Binary Search Tree (BST)

36b. Insertion in BST and Diameter Calculation

Set-37

37a. Longest Increasing Subsequence (LIS)

37b. Longest Common Prefix Using Trie

Set-38

38a. Longest Bitonic Subsequence

38b. Count Substring Occurrences in Array of Strings

Set-39

39a. Longest Decreasing Subsequence

39b. String Frequency Sort

Set-40

40a. Basic Stock Buying and Selling Simulation

40b. Count half nodes in bst

Set-41

41a. Maximum Profit from Stock Prices

41 a:sum of sub arrays (kadane algorithm)

B:bst find right view of tree