1. **Problem Statement:**

Write a program in java to Create an array of integer with size n. Return the difference between the largest and the smallest value inside that array.

Answer:

import java.util.Scanner;

public class ArrayDifference {

// Method to find the maximum value in an array

public static int findMax(int[] arr) {

int max = arr[0];

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

if (arr[i] > max) {

max = arr[i];

}

}

return max;

}

// Method to find the minimum value in an array

public static int findMin(int[] arr) {

int min = arr[0];

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

if (arr[i] < min) {

min = arr[i];

}

}

return min;

}

// Main method

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input size of the array

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

int n = scanner.nextInt();

// Declare and initialize the array

int[] arr = new int[n];

// Input elements of the array

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

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

arr[i] = scanner.nextInt();

}

// Find maximum and minimum values

int max = findMax(arr);

int min = findMin(arr);

// Calculate and print the difference

int difference = max - min;

System.out.println("The difference between the largest and smallest values is: " + difference);

}

}

1. **Problem Statement:**

Write a java program that initializes an array with ten random integers and then prints four lines of output, containing:

* Every element at an even index
* Every odd element
* All elements in reverse order

Only the first and last element

import java.util.Random;

public class ArrayOperations {

public static void main(String[] args) {

// Initialize the array with 10 random integers

int[] arr = new int[10];

Random rand = new Random();

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

arr[i] = rand.nextInt(100); // Random integers between 0 and 99

}

// Print the original array

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

for (int num : arr) {

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

}

System.out.println("\n");

// Print every element at an even index

System.out.println("Elements at even indices:");

for (int i = 0; i < arr.length; i += 2) {

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

}

System.out.println();

// Print every odd element

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

for (int num : arr) {

if (num % 2 != 0) {

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

}

}

System.out.println();

// Print all elements in reverse order

System.out.println("Elements in reverse order:");

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

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

}

System.out.println();

}

}

**Only the last element**

import java.util.Random;

public class ArrayOperations {

public static void main(String[] args) {

// Initialize the array with 10 random integers

int[] arr = new int[10];

Random rand = new Random();

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

arr[i] = rand.nextInt(100); // Random integers between 0 and 99

}

// Print the original array

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

for (int num : arr) {

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

}

System.out.println("\n");

// Print every element at an even index

System.out.println("Elements at even indices:");

for (int i = 0; i < arr.length; i += 2) {

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

}

System.out.println();

// Print every odd element

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

for (int num : arr) {

if (num % 2 != 0) {

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

}

}

System.out.println();

// Print all elements in reverse order

System.out.println("Elements in reverse order:");

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

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

}

System.out.println();

// Print only the first and last element

System.out.println("First and last elements:");

if (arr.length > 0) {

System.out.print(arr[0] + " ");

if (arr.length > 1) {

System.out.print(arr[arr.length - 1]);

}

}

System.out.println();

}

}

1. **Problem Statement:**

Write a program to read numbers in an integer array of size 5 and display the following:

* Sum of all the elements
* Sum of alternate elements in the array

Second highest element in the array

import java.util.Scanner;

public class ArrayOperations {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int[] arr = new int[5];

// Read 5 numbers into the array

System.out.println("Enter 5 integers:");

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

arr[i] = scanner.nextInt();

}

// Sum of all elements

int totalSum = 0;

for (int num : arr) {

totalSum += num;

}

System.out.println("Sum of all elements: " + totalSum);

// Sum of alternate elements (considering the first element as alternate)

int alternateSum = 0;

for (int i = 0; i < arr.length; i += 2) {

alternateSum += arr[i];

}

System.out.println("Sum of alternate elements: " + alternateSum);

// Finding the second highest element in the array

if (arr.length < 2) {

System.out.println("Array must have at least two elements to find the second highest element.");

return;

}

int highest = Integer.MIN\_VALUE;

int secondHighest = Integer.MIN\_VALUE;

for (int num : arr) {

if (num > highest) {

secondHighest = highest;

highest = num;

} else if (num > secondHighest && num != highest) {

secondHighest = num;

}

}

// Check if secondHighest was updated

if (secondHighest == Integer.MIN\_VALUE) {

System.out.println("No second highest element found.");

} else {

System.out.println("Second highest element: " + secondHighest);

}

}

}

1. **Problem Statement:**

Write a program to create a singly linked list of n nodes and perform:

• Insertion

* At the beginning
* At the end
* At a specific location

• Deletion

* At the beginning
* At the end
* At a specific location

import java.util.Scanner;

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class SinglyLinkedList {

Node head;

// Method to insert a node at the beginning

public void insertAtBeginning(int data) {

Node newNode = new Node(data);

newNode.next = head;

head = newNode;

}

// Method to insert a node at the end

public void insertAtEnd(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

} else {

Node temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

}

}

// Method to insert a node at a specific position

public void insertAtPosition(int data, int position) {

Node newNode = new Node(data);

if (position <= 0) {

System.out.println("Invalid position. Inserting at the beginning.");

insertAtBeginning(data);

} else if (head == null && position > 0) {

System.out.println("List is empty. Inserting at the beginning.");

insertAtBeginning(data);

} else {

Node temp = head;

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

if (temp == null || temp.next == null) {

System.out.println("Position out of bounds. Inserting at the end.");

break;

}

temp = temp.next;

}

newNode.next = temp.next;

temp.next = newNode;

}

}

// Method to delete a node from the beginning

public void deleteFromBeginning() {

if (head == null) {

System.out.println("List is empty. Nothing to delete.");

} else {

head = head.next;

}

}

// Method to delete a node from the end

public void deleteFromEnd() {

if (head == null) {

System.out.println("List is empty. Nothing to delete.");

} else if (head.next == null) {

head = null;

} else {

Node temp = head;

while (temp.next.next != null) {

temp = temp.next;

}

temp.next = null;

}

}

// Method to delete a node from a specific position

public void deleteFromPosition(int position) {

if (head == null) {

System.out.println("List is empty. Nothing to delete.");

} else if (position <= 0) {

deleteFromBeginning();

} else {

Node temp = head;

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

if (temp.next == null) {

System.out.println("Position out of bounds. Nothing to delete.");

return;

}

temp = temp.next;

}

if (temp.next == null) {

System.out.println("Position out of bounds. Nothing to delete.");

} else {

temp.next = temp.next.next;

}

}

}

// Method to display the linked list

public void display() {

Node temp = head;

while (temp != null) {

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

temp = temp.next;

}

System.out.println("null");

}

}

public class LinkedListDemo {

public static void main(String[] args) {

SinglyLinkedList list = new SinglyLinkedList();

Scanner scanner = new Scanner(System.in);

boolean exit = false;

while (!exit) {

System.out.println("\nSingly Linked List Operations:");

System.out.println("1. Insert at Beginning");

System.out.println("2. Insert at End");

System.out.println("3. Insert at Position");

System.out.println("4. Delete from Beginning");

System.out.println("5. Delete from End");

System.out.println("6. Delete from Position");

System.out.println("7. Display");

System.out.println("8. Exit");

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

int choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter data to insert at beginning: ");

int dataBeginning = scanner.nextInt();

list.insertAtBeginning(dataBeginning);

break;

case 2:

System.out.print("Enter data to insert at end: ");

int dataEnd = scanner.nextInt();

list.insertAtEnd(dataEnd);

break;

case 3:

System.out.print("Enter data to insert: ");

int dataPosition = scanner.nextInt();

System.out.print("Enter position to insert (0-based index): ");

int position = scanner.nextInt();

list.insertAtPosition(dataPosition, position);

break;

case 4:

list.deleteFromBeginning();

break;

case 5:

list.deleteFromEnd();

break;

case 6:

System.out.print("Enter position to delete (0-based index): ");

int delPosition = scanner.nextInt();

list.deleteFromPosition(delPosition);

break;

case 7:

list.display();

break;

case 8:

exit = true;

break;

default:

System.out.println("Invalid choice. Please try again.");

}

}

scanner.close();

}

}

1. **Problem Statement:**

Write a program to create a doubly linked list of n nodes and perform:

• Insertion

* At the beginning
* At the end
* At a specific location

• Deletion

* At the beginning
* At the end

At a specific location

import java.util.Scanner;

class Node {

int data;

Node next;

Node prev;

Node(int data) {

this.data = data;

this.next = null;

this.prev = null;

}

}

class DoublyLinkedList {

Node head;

// Method to insert a node at the beginning

public void insertAtBeginning(int data) {

Node newNode = new Node(data);

if (head != null) {

head.prev = newNode;

newNode.next = head;

}

head = newNode;

}

// Method to insert a node at the end

public void insertAtEnd(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

} else {

Node temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

newNode.prev = temp;

}

}

// Method to insert a node at a specific position

public void insertAtPosition(int data, int position) {

if (position <= 0) {

System.out.println("Invalid position. Inserting at the beginning.");

insertAtBeginning(data);

return;

}

Node newNode = new Node(data);

if (head == null) {

System.out.println("List is empty. Inserting at the beginning.");

head = newNode;

return;

}

Node temp = head;

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

if (temp.next == null) {

System.out.println("Position out of bounds. Inserting at the end.");

break;

}

temp = temp.next;

}

newNode.next = temp.next;

if (temp.next != null) {

temp.next.prev = newNode;

}

temp.next = newNode;

newNode.prev = temp;

}

// Method to delete a node from the beginning

public void deleteFromBeginning() {

if (head == null) {

System.out.println("List is empty. Nothing to delete.");

return;

}

head = head.next;

if (head != null) {

head.prev = null;

}

}

// Method to delete a node from the end

public void deleteFromEnd() {

if (head == null) {

System.out.println("List is empty. Nothing to delete.");

return;

}

if (head.next == null) {

head = null;

return;

}

Node temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.prev.next = null;

}

// Method to delete a node from a specific position

public void deleteFromPosition(int position) {

if (head == null) {

System.out.println("List is empty. Nothing to delete.");

return;

}

if (position <= 0) {

deleteFromBeginning();

return;

}

Node temp = head;

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

if (temp.next == null) {

System.out.println("Position out of bounds. Nothing to delete.");

return;

}

temp = temp.next;

}

if (temp.next != null) {

temp.next.prev = temp.prev;

}

if (temp.prev != null) {

temp.prev.next = temp.next;

} else {

head = temp.next;

}

}

// Method to display the linked list

public void display() {

Node temp = head;

while (temp != null) {

System.out.print(temp.data + " <-> ");

temp = temp.next;

}

System.out.println("null");

}

}

public class DoublyLinkedListDemo {

public static void main(String[] args) {

DoublyLinkedList list = new DoublyLinkedList();

Scanner scanner = new Scanner(System.in);

boolean exit = false;

while (!exit) {

System.out.println("\nDoubly Linked List Operations:");

System.out.println("1. Insert at Beginning");

System.out.println("2. Insert at End");

System.out.println("3. Insert at Position");

System.out.println("4. Delete from Beginning");

System.out.println("5. Delete from End");

System.out.println("6. Delete from Position");

System.out.println("7. Display");

System.out.println("8. Exit");

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

int choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter data to insert at beginning: ");

int dataBeginning = scanner.nextInt();

list.insertAtBeginning(dataBeginning);

break;

case 2:

System.out.print("Enter data to insert at end: ");

int dataEnd = scanner.nextInt();

list.insertAtEnd(dataEnd);

break;

case 3:

System.out.print("Enter data to insert: ");

int dataPosition = scanner.nextInt();

System.out.print("Enter position to insert (0-based index): ");

int position = scanner.nextInt();

list.insertAtPosition(dataPosition, position);

break;

case 4:

list.deleteFromBeginning();

break;

case 5:

list.deleteFromEnd();

break;

case 6:

System.out.print("Enter position to delete (0-based index): ");

int delPosition = scanner.nextInt();

list.deleteFromPosition(delPosition);

break;

case 7:

list.display();

break;

case 8:

exit = true;

break;

default:

System.out.println("Invalid choice. Please try again.");

}

}

scanner.close();

}

}

1. **Problem Statement:**

Write a program to create a Circular linked list of n nodes and perform:

• Insertion

* At the beginning
* At the end
* At a specific location

• Deletion

* At the beginning
* At the end
* At a specific location

import java.util.Scanner;

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class CircularLinkedList {

Node last;

// Method to insert a node at the beginning

public void insertAtBeginning(int data) {

Node newNode = new Node(data);

if (last == null) {

last = newNode;

last.next = last;

} else {

newNode.next = last.next;

last.next = newNode;

}

}

// Method to insert a node at the end

public void insertAtEnd(int data) {

Node newNode = new Node(data);

if (last == null) {

last = newNode;

last.next = last;

} else {

newNode.next = last.next;

last.next = newNode;

last = newNode;

}

}

// Method to insert a node at a specific position

public void insertAtPosition(int data, int position) {

if (position <= 0) {

System.out.println("Invalid position. Inserting at the beginning.");

insertAtBeginning(data);

return;

}

Node newNode = new Node(data);

if (last == null) {

System.out.println("List is empty. Inserting at the beginning.");

last = newNode;

last.next = last;

return;

}

Node temp = last.next;

for (int i = 1; i < position && temp != last; i++) {

temp = temp.next;

}

newNode.next = temp.next;

temp.next = newNode;

if (temp == last) {

last = newNode;

}

}

// Method to delete a node from the beginning

public void deleteFromBeginning() {

if (last == null) {

System.out.println("List is empty. Nothing to delete.");

return;

}

Node temp = last.next;

if (last == last.next) {

last = null;

} else {

last.next = temp.next;

}

}

// Method to delete a node from the end

public void deleteFromEnd() {

if (last == null) {

System.out.println("List is empty. Nothing to delete.");

return;

}

Node temp = last.next;

if (last == last.next) {

last = null;

} else {

while (temp.next != last) {

temp = temp.next;

}

temp.next = last.next;

last = temp;

}

}

// Method to delete a node from a specific position

public void deleteFromPosition(int position) {

if (last == null) {

System.out.println("List is empty. Nothing to delete.");

return;

}

if (position <= 0) {

deleteFromBeginning();

return;

}

Node temp = last.next;

for (int i = 1; i < position && temp.next != last.next; i++) {

temp = temp.next;

}

if (temp.next == last.next) {

System.out.println("Position out of bounds. Nothing to delete.");

return;

}

if (temp.next == last) {

deleteFromEnd();

} else {

temp.next = temp.next.next;

}

}

// Method to display the circular linked list

public void display() {

if (last == null) {

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

return;

}

Node temp = last.next;

do {

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

temp = temp.next;

} while (temp != last.next);

System.out.println("(back to start)");

}

}

public class CircularLinkedListDemo {

public static void main(String[] args) {

CircularLinkedList list = new CircularLinkedList();

Scanner scanner = new Scanner(System.in);

boolean exit = false;

while (!exit) {

System.out.println("\nCircular Linked List Operations:");

System.out.println("1. Insert at Beginning");

System.out.println("2. Insert at End");

System.out.println("3. Insert at Position");

System.out.println("4. Delete from Beginning");

System.out.println("5. Delete from End");

System.out.println("6. Delete from Position");

System.out.println("7. Display");

System.out.println("8. Exit");

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

int choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter data to insert at beginning: ");

int dataBeginning = scanner.nextInt();

list.insertAtBeginning(dataBeginning);

break;

case 2:

System.out.print("Enter data to insert at end: ");

int dataEnd = scanner.nextInt();

list.insertAtEnd(dataEnd);

break;

case 3:

System.out.print("Enter data to insert: ");

int dataPosition = scanner.nextInt();

System.out.print("Enter position to insert (0-based index): ");

int position = scanner.nextInt();

list.insertAtPosition(dataPosition, position);

break;

case 4:

list.deleteFromBeginning();

break;

case 5:

list.deleteFromEnd();

break;

case 6:

System.out.print("Enter position to delete (0-based index): ");

int delPosition = scanner.nextInt();

list.deleteFromPosition(delPosition);

break;

case 7:

list.display();

break;

case 8:

exit = true;

break;

default:

System.out.println("Invalid choice. Please try again.");

}

}

scanner.close();

}

}

1. **Problem Statement:**

Write a program to create a stack and perform:

* POP
* PUSH
* PEEK
* ISEMPTY
* ISFULL

1. Use Arrays for Implementation
2. Use Linked List for Implementation

import java.util.Scanner;

class ArrayStack {

private int maxSize;

private int[] stackArray;

private int top;

public ArrayStack(int size) {

this.maxSize = size;

this.stackArray = new int[maxSize];

this.top = -1;

}

public void push(int value) {

if (isFull()) {

System.out.println("Stack is full. Cannot push value.");

} else {

stackArray[++top] = value;

}

}

public int pop() {

if (isEmpty()) {

System.out.println("Stack is empty. Cannot pop value.");

return -1;

} else {

return stackArray[top--];

}

}

public int peek() {

if (isEmpty()) {

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

return -1;

} else {

return stackArray[top];

}

}

public boolean isEmpty() {

return top == -1;

}

public boolean isFull() {

return top == maxSize - 1;

}

public void display() {

if (isEmpty()) {

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

} else {

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

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

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

}

System.out.println();

}

}

}

public class ArrayStackDemo {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter stack size: ");

int size = scanner.nextInt();

ArrayStack stack = new ArrayStack(size);

boolean exit = false;

while (!exit) {

System.out.println("\nStack Operations using Array:");

System.out.println("1. Push");

System.out.println("2. Pop");

System.out.println("3. Peek");

System.out.println("4. Is Empty");

System.out.println("5. Is Full");

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

System.out.println("7. Exit");

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

int choice = scanner.nextInt();

switch (choice) {

case 1:

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

int value = scanner.nextInt();

stack.push(value);

break;

case 2:

int poppedValue = stack.pop();

if (poppedValue != -1) {

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

}

break;

case 3:

int peekedValue = stack.peek();

if (peekedValue != -1) {

System.out.println("Top value: " + peekedValue);

}

break;

case 4:

System.out.println("Is stack empty? " + stack.isEmpty());

break;

case 5:

System.out.println("Is stack full? " + stack.isFull());

break;

case 6:

stack.display();

break;

case 7:

exit = true;

break;

default:

System.out.println("Invalid choice. Please try again.");

}

}

scanner.close();

}

}

1. **Problem Statement:**

Write a program to create a stack and perform:

Reversal of a sentence using stack.

Given a string str consisting of a sentence, the task is to reverse the entire sentence word by word.

Examples:

Input: str = “data structures and algorithms”  
Output:  algorithms and structures data

import java.util.Scanner;

import java.util.Stack;

public class SentenceReversal {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read the sentence from the user

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

String sentence = scanner.nextLine();

// Reverse the sentence using stack

String reversedSentence = reverseSentence(sentence);

// Print the reversed sentence

System.out.println("Reversed sentence: " + reversedSentence);

scanner.close();

}

public static String reverseSentence(String sentence) {

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

String[] words = sentence.split(" ");

// Push all words onto the stack

for (String word : words) {

stack.push(word);

}

// Pop words from the stack to get them in reverse order

StringBuilder reversed = new StringBuilder();

while (!stack.isEmpty()) {

reversed.append(stack.pop());

if (!stack.isEmpty()) {

reversed.append(" ");

}

}

return reversed.toString();

}

}

1. **Problem Statement**

Write a program to check whether the parenthesis in the expression are balanced or not.

Given a string str consisting of an expression

Examples:

Input: str = (a+b)\*c

Output: Parenthesis Balanced

­­­ import java.util.Scanner;

import java.util.Stack;

public class ParenthesisBalancer {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read the expression from the user

System.out.print("Enter an expression: ");

String expression = scanner.nextLine();

// Check if the parentheses are balanced

boolean isBalanced = areParenthesesBalanced(expression);

// Print the result

if (isBalanced) {

System.out.println("The parentheses in the expression are balanced.");

} else {

System.out.println("The parentheses in the expression are not balanced.");

}

scanner.close();

}

public static boolean areParenthesesBalanced(String expression) {

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

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

switch (ch) {

case '(':

case '[':

case '{':

stack.push(ch);

break;

case ')':

if (stack.isEmpty() || stack.pop() != '(') {

return false;

}

break;

case ']':

if (stack.isEmpty() || stack.pop() != '[') {

return false;

}

break;

case '}':

if (stack.isEmpty() || stack.pop() != '{') {

return false;

}

break;

}

}

return stack.isEmpty();

}

}

1. **Problem Statement:**

Write a program to convert Infix expression into Postfix.

**Given a string str consisting of an infix expression, convert it into Postfix**

**Examples:**

**Input: str = (a+b)\*c**

**Output: ab+\***

import java.util.Stack;

import java.util.Scanner;

public class InfixToPostfixConverter {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read the infix expression from the user

System.out.print("Enter an infix expression: ");

String infixExpression = scanner.nextLine();

// Convert infix expression to postfix expression

String postfixExpression = infixToPostfix(infixExpression);

// Print the postfix expression

System.out.println("Postfix expression: " + postfixExpression);

scanner.close();

}

// Method to get the precedence of operators

private static int precedence(char operator) {

switch (operator) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

default:

return -1;

}

}

// Method to convert infix to postfix expression

public static String infixToPostfix(String infix) {

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

StringBuilder postfix = new StringBuilder();

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

// If the character is an operand, add it to the postfix expression

if (Character.isLetterOrDigit(ch)) {

postfix.append(ch);

}

// If the character is '(', push it to the stack

else if (ch == '(') {

stack.push(ch);

}

// If the character is ')', pop and output from the stack

// until an '(' is encountered

else if (ch == ')') {

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

postfix.append(stack.pop());

}

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

stack.pop();

}

}

// An operator is encountered

else {

while (!stack.isEmpty() && precedence(stack.peek()) >= precedence(ch)) {

postfix.append(stack.pop());

}

stack.push(ch);

}

}

// Pop all the operators from the stack

while (!stack.isEmpty()) {

postfix.append(stack.pop());

}

return postfix.toString();

}

}

1. **Problem Statement:**

Write a program to implement Tower of Hanoi.

import java.util.Scanner;

public class TowerOfHanoi {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read the number of disks from the user

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

int numDisks = scanner.nextInt();

// Perform the Tower of Hanoi algorithm

towerOfHanoi(numDisks, 'A', 'C', 'B');

scanner.close();

}

/\*\*

\* Recursive method to solve Tower of Hanoi problem.

\*

\* @param n the number of disks

\* @param from the source rod

\* @param to the destination rod

\* @param aux the auxiliary rod

\*/

public static void towerOfHanoi(int n, char from, char to, char aux) {

if (n == 1) {

System.out.println("Move disk 1 from rod " + from + " to rod " + to);

return;

}

// Move n-1 disks from 'from' to 'aux' using 'to' as auxiliary

towerOfHanoi(n - 1, from, aux, to);

// Move the nth disk from 'from' to 'to'

System.out.println("Move disk " + n + " from rod " + from + " to rod " + to);

// Move the n-1 disks from 'aux' to 'to' using 'from' as auxiliary

towerOfHanoi(n - 1, aux, to, from);

}

}

1. **Problem Statement:**

Write a program to implement Following operations using Queue:

1. Enqueue()
2. Dequeue()
3. Isfull()
4. Isempty()
5. Peek()
6. Using array implementation
7. Using Linked List Implementation

Implementation using Array:

import java.util.Scanner;

class ArrayQueue {

private int maxSize;

private int[] queueArray;

private int front;

private int rear;

private int nItems;

public ArrayQueue(int size) {

this.maxSize = size;

this.queueArray = new int[maxSize];

this.front = 0;

this.rear = -1;

this.nItems = 0;

}

public void enqueue(int value) {

if (isFull()) {

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

} else {

if (rear == maxSize - 1) {

rear = -1;

}

queueArray[++rear] = value;

nItems++;

}

}

public int dequeue() {

if (isEmpty()) {

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

return -1;

} else {

int temp = queueArray[front++];

if (front == maxSize) {

front = 0;

}

nItems--;

return temp;

}

}

public boolean isFull() {

return nItems == maxSize;

}

public boolean isEmpty() {

return nItems == 0;

}

public int peek() {

if (isEmpty()) {

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

return -1;

} else {

return queueArray[front];

}

}

public void display() {

if (isEmpty()) {

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

} else {

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

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

System.out.print(queueArray[(front + i) % maxSize] + " ");

}

System.out.println();

}

}

}

public class ArrayQueueDemo {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int size = scanner.nextInt();

ArrayQueue queue = new ArrayQueue(size);

boolean exit = false;

while (!exit) {

System.out.println("\nQueue Operations using Array:");

System.out.println("1. Enqueue");

System.out.println("2. Dequeue");

System.out.println("3. Is Full");

System.out.println("4. Is Empty");

System.out.println("5. Peek");

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

System.out.println("7. Exit");

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

int choice = scanner.nextInt();

switch (choice) {

case 1:

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

int value = scanner.nextInt();

queue.enqueue(value);

break;

case 2:

int dequeuedValue = queue.dequeue();

if (dequeuedValue != -1) {

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

}

break;

case 3:

System.out.println("Is queue full? " + queue.isFull());

break;

case 4:

System.out.println("Is queue empty? " + queue.isEmpty());

break;

case 5:

int peekedValue = queue.peek();

if (peekedValue != -1) {

System.out.println("Front value: " + peekedValue);

}

break;

case 6:

queue.display();

break;

case 7:

exit = true;

break;

default:

System.out.println("Invalid choice. Please try again.");

}

}

scanner.close();

}

}

**Implementation using Linked List**

import java.util.Scanner;

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

class LinkedListQueue {

private Node front;

private Node rear;

public LinkedListQueue() {

this.front = this.rear = null;

}

public void enqueue(int value) {

Node newNode = new Node(value);

if (rear == null) {

front = rear = newNode;

} else {

rear.next = newNode;

rear = newNode;

}

}

public int dequeue() {

if (isEmpty()) {

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

return -1;

} else {

int value = front.data;

front = front.next;

if (front == null) {

rear = null;

}

return value;

}

}

public boolean isFull() {

return false; // Linked list implementation of queue is never full unless memory is exhausted

}

public boolean isEmpty() {

return front == null;

}

public int peek() {

if (isEmpty()) {

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

return -1;

} else {

return front.data;

}

}

public void display() {

if (isEmpty()) {

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

} else {

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

Node temp = front;

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

}

}

public class LinkedListQueueDemo {

public static void main(String[] args) {

LinkedListQueue queue = new LinkedListQueue();

Scanner scanner = new Scanner(System.in);

boolean exit = false;

while (!exit) {

System.out.println("\nQueue Operations using Linked List:");

System.out.println("1. Enqueue");

System.out.println("2. Dequeue");

System.out.println("3. Is Full");

System.out.println("4. Is Empty");

System.out.println("5. Peek");

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

System.out.println("7. Exit");

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

int choice = scanner.nextInt();

switch (choice) {

case 1:

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

int value = scanner.nextInt();

queue.enqueue(value);

break;

case 2:

int dequeuedValue = queue.dequeue();

if (dequeuedValue != -1) {

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

}

break;

case 3:

System.out.println("Is queue full? " + queue.isFull());

break;

case 4:

System.out.println("Is queue empty? " + queue.isEmpty());

break;

case 5:

int peekedValue = queue.peek();

if (peekedValue != -1) {

System.out.println("Front value: " + peekedValue);

}

break;

case 6:

queue.display();

break;

case 7:

exit = true;

break;

default:

System.out.println("Invalid choice. Please try again.");

}

}

scanner.close();

}

}

1. **Problem Statement:**

Write a program to implement Following operations using Circular Queue:

1. Enqueue()
2. Dequeue()

Using array implementation

import java.util.Scanner;

class CircularQueue {

private int maxSize;

private int[] queueArray;

private int front;

private int rear;

private int nItems;

public CircularQueue(int size) {

this.maxSize = size;

this.queueArray = new int[maxSize];

this.front = 0;

this.rear = -1;

this.nItems = 0;

}

public void enqueue(int value) {

if (isFull()) {

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

} else {

if (rear == maxSize - 1) {

rear = -1;

}

queueArray[++rear] = value;

nItems++;

}

}

public int dequeue() {

if (isEmpty()) {

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

return -1;

} else {

int temp = queueArray[front++];

if (front == maxSize) {

front = 0;

}

nItems--;

return temp;

}

}

public boolean isFull() {

return nItems == maxSize;

}

public boolean isEmpty() {

return nItems == 0;

}

public void display() {

if (isEmpty()) {

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

} else {

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

int i = front;

for (int count = 0; count < nItems; count++) {

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

i = (i + 1) % maxSize;

}

System.out.println();

}

}

}

public class CircularQueueDemo {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int size = scanner.nextInt();

CircularQueue queue = new CircularQueue(size);

boolean exit = false;

while (!exit) {

System.out.println("\nQueue Operations using Circular Array:");

System.out.println("1. Enqueue");

System.out.println("2. Dequeue");

System.out.println("3. Is Full");

System.out.println("4. Is Empty");

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

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

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

int choice = scanner.nextInt();

switch (choice) {

case 1:

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

int value = scanner.nextInt();

queue.enqueue(value);

break;

case 2:

int dequeuedValue = queue.dequeue();

if (dequeuedValue != -1) {

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

}

break;

case 3:

System.out.println("Is queue full? " + queue.isFull());

break;

case 4:

System.out.println("Is queue empty? " + queue.isEmpty());

break;

case 5:

queue.display();

break;

case 6:

exit = true;

break;

default:

System.out.println("Invalid choice. Please try again.");

}

}

scanner.close();

}

}

**Implementation using Linked List**

import java.util.Scanner;

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

class CircularQueueLinkedList {

private Node rear;

private Node front;

public CircularQueueLinkedList() {

this.front = null;

this.rear = null;

}

public void enqueue(int value) {

Node newNode = new Node(value);

if (rear == null) {

front = rear = newNode;

rear.next = front; // Circular link

} else {

rear.next = newNode;

rear = newNode;

rear.next = front; // Circular link

}

}

public int dequeue() {

if (isEmpty()) {

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

return -1;

} else {

int value = front.data;

if (front == rear) {

front = rear = null;

} else {

front = front.next;

rear.next = front; // Maintain circular link

}

return value;

}

}

public boolean isEmpty() {

return front == null;

}

public void display() {

if (isEmpty()) {

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

} else {

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

Node temp = front;

do {

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

temp = temp.next;

} while (temp != front);

System.out.println();

}

}

}

public class CircularQueueLinkedListDemo {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

CircularQueueLinkedList queue = new CircularQueueLinkedList();

boolean exit = false;

while (!exit) {

System.out.println("\nQueue Operations using Circular Linked List:");

System.out.println("1. Enqueue");

System.out.println("2. Dequeue");

System.out.println("3. Is Empty");

System.out.println("4. Display");

System.out.println("5. Exit");

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

int choice = scanner.nextInt();

switch (choice) {

case 1:

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

int value = scanner.nextInt();

queue.enqueue(value);

break;

case 2:

int dequeuedValue = queue.dequeue();

if (dequeuedValue != -1) {

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

}

break;

case 3:

System.out.println("Is queue empty? " + queue.isEmpty());

break;

case 4:

queue.display();

break;

case 5:

exit = true;

break;

default:

System.out.println("Invalid choice. Please try again.");

}

}

scanner.close();

}

}

1. **Problem Statement:**

Write a program to implement Following operations using Doubly ended Queue:

1. Enqueue()
2. Dequeue()
3. Isfull()
4. Isempty()
5. Peek()

Using array implementation

import java.util.Scanner;

class Deque {

private int maxSize;

private int[] dequeArray;

private int front;

private int rear;

private int nItems;

public Deque(int size) {

this.maxSize = size;

this.dequeArray = new int[maxSize];

this.front = 0;

this.rear = -1;

this.nItems = 0;

}

public void enqueue(int value) {

if (isFull()) {

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

} else {

if (rear == maxSize - 1) {

rear = -1;

}

dequeArray[++rear] = value;

nItems++;

}

}

public int dequeue() {

if (isEmpty()) {

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

return -1;

} else {

int temp = dequeArray[front++];

if (front == maxSize) {

front = 0;

}

nItems--;

return temp;

}

}

public boolean isFull() {

return nItems == maxSize;

}

public boolean isEmpty() {

return nItems == 0;

}

public int peek() {

if (isEmpty()) {

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

return -1;

} else {

return dequeArray[front];

}

}

public void display() {

if (isEmpty()) {

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

} else {

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

int i = front;

for (int count = 0; count < nItems; count++) {

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

i = (i + 1) % maxSize;

}

System.out.println();

}

}

}

public class DequeDemo {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int size = scanner.nextInt();

Deque deque = new Deque(size);

boolean exit = false;

while (!exit) {

System.out.println("\nDeque Operations using Array:");

System.out.println("1. Enqueue at Rear");

System.out.println("2. Dequeue from Front");

System.out.println("3. Is Full");

System.out.println("4. Is Empty");

System.out.println("5. Peek");

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

System.out.println("7. Exit");

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

int choice = scanner.nextInt();

switch (choice) {

case 1:

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

int value = scanner.nextInt();

deque.enqueue(value);

break;

case 2:

int dequeuedValue = deque.dequeue();

if (dequeuedValue != -1) {

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

}

break;

case 3:

System.out.println("Is deque full? " + deque.isFull());

break;

case 4:

System.out.println("Is deque empty? " + deque.isEmpty());

break;

case 5:

int peekedValue = deque.peek();

if (peekedValue != -1) {

System.out.println("Front value: " + peekedValue);

}

break;

case 6:

deque.display();

break;

case 7:

exit = true;

break;

default:

System.out.println("Invalid choice. Please try again.");

}

}

scanner.close();

}

}

1. **Problem Statement:**

Write a program to implement Following operations using Binary Search Tree:

1. Insertion
2. Deletion
3. Traversal [PREORDER, POSTORDER, INORDER]

import java.util.Scanner;

class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

class BinarySearchTree {

Node root;

BinarySearchTree() {

root = null;

}

// Insert a node in the BST

void insert(int key) {

root = insertRec(root, key);

}

Node insertRec(Node root, int key) {

if (root == null) {

root = new Node(key);

return root;

}

if (key < root.data) {

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

} else if (key > root.data) {

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

}

return root;

}

// Delete a node in the BST

void delete(int key) {

root = deleteRec(root, key);

}

Node deleteRec(Node root, int key) {

if (root == null) {

return root;

}

if (key < root.data) {

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

} else if (key > root.data) {

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

} else {

if (root.left == null) {

return root.right;

} else if (root.right == null) {

return root.left;

}

root.data = minValue(root.right);

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

}

return root;

}

int minValue(Node root) {

int minValue = root.data;

while (root.left != null) {

minValue = root.left.data;

root = root.left;

}

return minValue;

}

// Preorder traversal (Root, Left, Right)

void preorder() {

preorderRec(root);

System.out.println();

}

void preorderRec(Node root) {

if (root != null) {

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

preorderRec(root.left);

preorderRec(root.right);

}

}

// Inorder traversal (Left, Root, Right)

void inorder() {

inorderRec(root);

System.out.println();

}

void inorderRec(Node root) {

if (root != null) {

inorderRec(root.left);

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

inorderRec(root.right);

}

}

// Postorder traversal (Left, Right, Root)

void postorder() {

postorderRec(root);

System.out.println();

}

void postorderRec(Node root) {

if (root != null) {

postorderRec(root.left);

postorderRec(root.right);

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

}

}

}

public class BinarySearchTreeDemo {

public static void main(String[] args) {

BinarySearchTree bst = new BinarySearchTree();

Scanner scanner = new Scanner(System.in);

boolean exit = false;

while (!exit) {

System.out.println("\nBinary Search Tree Operations:");

System.out.println("1. Insert");

System.out.println("2. Delete");

System.out.println("3. Preorder Traversal");

System.out.println("4. Inorder Traversal");

System.out.println("5. Postorder Traversal");

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

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

int choice = scanner.nextInt();

switch (choice) {

case 1:

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

int insertValue = scanner.nextInt();

bst.insert(insertValue);

break;

case 2:

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

int deleteValue = scanner.nextInt();

bst.delete(deleteValue);

break;

case 3:

System.out.println("Preorder traversal:");

bst.preorder();

break;

case 4:

System.out.println("Inorder traversal:");

bst.inorder();

break;

case 5:

System.out.println("Postorder traversal:");

bst.postorder();

break;

case 6:

exit = true;

break;

default:

System.out.println("Invalid choice. Please try again.");

}

}

scanner.close();

}

}

1. **Problem Statement:**

Write a program to implement:

1. Bubble Sort
2. Insertions Sort
3. Selection Sort
4. Quick Sort
5. Merge Sort

import java.util.Arrays;

public class SortingAlgorithms {

// Bubble Sort

public static void bubbleSort(int[] arr) {

int n = arr.length;

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

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

if (arr[j] > arr[j + 1]) {

// Swap arr[j] and arr[j + 1]

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

// Insertion Sort

public static void insertionSort(int[] arr) {

int n = arr.length;

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

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

}

}

// Selection Sort

public static void selectionSort(int[] arr) {

int n = arr.length;

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

int minIndex = i;

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

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

// Swap arr[i] and arr[minIndex]

int temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

}

}

// Quick Sort

public static void quickSort(int[] arr, int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

private static int partition(int[] arr, int low, int high) {

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

// Swap arr[i] and arr[j]

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

// Swap arr[i + 1] and arr[high]

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

// Merge Sort

public static void mergeSort(int[] arr, int l, int r) {

if (l < r) {

int m = (l + r) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

private static void merge(int[] arr, int l, int m, int r) {

int n1 = m - l + 1;

int n2 = r - m;

int[] L = new int[n1];

int[] R = new int[n2];

System.arraycopy(arr, l, L, 0, n1);

System.arraycopy(arr, m + 1, R, 0, n2);

int i = 0, j = 0;

int k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

// Helper method to print the array

public static void printArray(int[] arr) {

for (int num : arr) {

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

}

System.out.println();

}

public static void main(String[] args) {

int[] array = {64, 34, 25, 12, 22, 11, 90};

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

printArray(array);

// Bubble Sort

int[] bubbleArray = Arrays.copyOf(array, array.length);

bubbleSort(bubbleArray);

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

printArray(bubbleArray);

// Insertion Sort

int[] insertionArray = Arrays.copyOf(array, array.length);

insertionSort(insertionArray);

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

printArray(insertionArray);

// Selection Sort

int[] selectionArray = Arrays.copyOf(array, array.length);

selectionSort(selectionArray);

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

printArray(selectionArray);

// Quick Sort

int[] quickArray = Arrays.copyOf(array, array.length);

quickSort(quickArray, 0, quickArray.length - 1);

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

printArray(quickArray);

// Merge Sort

int[] mergeArray = Arrays.copyOf(array, array.length);

mergeSort(mergeArray, 0, mergeArray.length - 1);

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

printArray(mergeArray);

}

}

**Using dynamic size array:**

import java.util.Arrays;

import java.util.Scanner;

public class SortingAlgorithms {

// Bubble Sort

public static void bubbleSort(int[] arr) {

int n = arr.length;

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

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

if (arr[j] > arr[j + 1]) {

// Swap arr[j] and arr[j + 1]

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

// Insertion Sort

public static void insertionSort(int[] arr) {

int n = arr.length;

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

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

}

}

// Selection Sort

public static void selectionSort(int[] arr) {

int n = arr.length;

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

int minIndex = i;

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

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

// Swap arr[i] and arr[minIndex]

int temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

}

}

// Quick Sort

public static void quickSort(int[] arr, int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

private static int partition(int[] arr, int low, int high) {

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

// Swap arr[i] and arr[j]

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

// Swap arr[i + 1] and arr[high]

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

// Merge Sort

public static void mergeSort(int[] arr, int l, int r) {

if (l < r) {

int m = (l + r) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

private static void merge(int[] arr, int l, int m, int r) {

int n1 = m - l + 1;

int n2 = r - m;

int[] L = new int[n1];

int[] R = new int[n2];

System.arraycopy(arr, l, L, 0, n1);

System.arraycopy(arr, m + 1, R, 0, n2);

int i = 0, j = 0;

int k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

// Helper method to print the array

public static void printArray(int[] arr) {

for (int num : arr) {

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

}

System.out.println();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int[] array = new int[n];

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

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

array[i] = scanner.nextInt();

}

// Bubble Sort

int[] bubbleArray = Arrays.copyOf(array, array.length);

bubbleSort(bubbleArray);

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

printArray(bubbleArray);

// Insertion Sort

int[] insertionArray = Arrays.copyOf(array, array.length);

insertionSort(insertionArray);

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

printArray(insertionArray);

// Selection Sort

int[] selectionArray = Arrays.copyOf(array, array.length);

selectionSort(selectionArray);

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

printArray(selectionArray);

// Quick Sort

int[] quickArray = Arrays.copyOf(array, array.length);

quickSort(quickArray, 0, quickArray.length - 1);

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

printArray(quickArray);

// Merge Sort

int[] mergeArray = Arrays.copyOf(array, array.length);

mergeSort(mergeArray, 0, mergeArray.length - 1);

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

printArray(mergeArray);

scanner.close();

}

}