*FINAL TERM EXAMINATION FALL 2024 (DSA)*

***Q1) Merge Sort:***

***Code:***

public class Sorting {

public void mergeSort(int[] arr, int left, int right) {

if (left < right) {

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

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

private void merge(int[] arr, int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int[] leftArray = new int[n1];

int[] rightArray = new int[n2];

for (int i = 0; i < n1; i++) leftArray[i] = arr[left + i];

for (int j = 0; j < n2; j++) rightArray[j] = arr[mid + 1 + j];

int i = 0, j = 0, k = left;

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

if (leftArray[i] <= rightArray[j]) arr[k++] = leftArray[i++];

else arr[k++] = rightArray[j++];

}

while (i < n1) arr[k++] = leftArray[i++];

while (j < n2) arr[k++] = rightArray[j++];

}

public static void main(String[] args) {

int[] arr1 = {100, 40, 80, 20, 50, 10, 90, 30, 60, 70};

Sorting sorter = new Sorting();

System.out.println("Before Sorting:");

for (int t : arr1) System.out.print(t + " ");

System.out.println();

sorter.mergeSort(arr1, 0, arr1.length - 1);

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

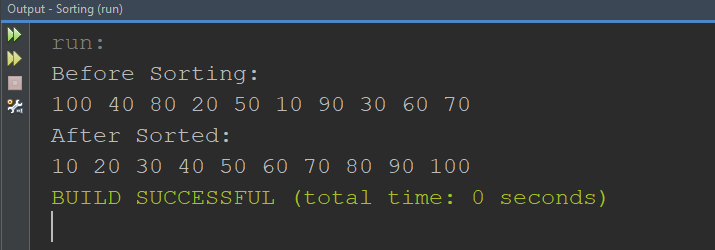
for (int t : arr1) System.out.print(t + " ");

System.out.println();

}

}

***Output:***

******

***Q2) Linear Search:***

***Code:***

import java.util.Arrays;

import java.util.Scanner;

public class Linearsearch {

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

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

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

}

return -1;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int[] ids = {101, 102, 103, 104, 105, 106, 107, 108, 109, 110};

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

int target = sc.nextInt();

Linearsearch search = new Linearsearch();

System.out.println("Perform Linear Search...");

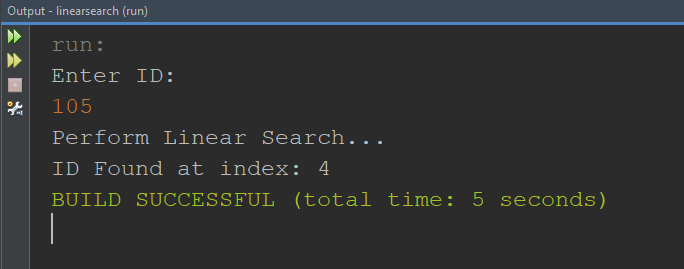
int linearResult = search.linearSearch(ids, target);

System.out.println(linearResult != -1 ? "ID Found at index: " + linearResult : "ID Not Found");

}

}

***Output:***

******

***Q3)Stack***

***Code:***

public class Stack {

private String[] stack;

private int top;

private int capacity;

public Stack(int size) {

stack = new String[size];

top = -1;

capacity = size;

}

public void push(String Integer) {

if (top == capacity - 1) {

System.out.println("Stack Overflow! Cannot add Integer.");

} else {

stack[++top] = Integer;

}

}

public String pop() {

if (top == -1) {

System.out.println("Stack Underflow! No Integer to go back to.");

return null;

} else {

return stack[top--];

}

}

public String peek() {

if (top == -1) {

System.out.println("No Integer");

return null;

} else {

return stack[top];

}

}

public static void main(String[] args) {

Stack stack = new Stack(5);

stack.push("1");

stack.push("2");

stack.push("3");

System.out.println("Current Integer: " + stack.peek());

stack.pop();

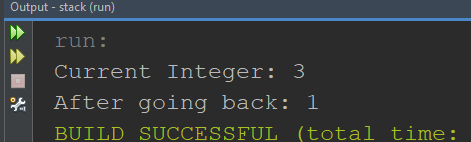
stack.pop();

System.out.println("After going back: " + stack.peek());

}

}

***Output:***

******

***Q4)BinaryTree***

***Code:***

class Node {

String name;

Node left;

Node right;

public Node(String name) {

this.name = name;

this.left = null;

this.right = null;

}

}

class Tree1 {

Node root;

public void inOrderTraversal(Node node) {

if (node != null) {

inOrderTraversal(node.left);

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

inOrderTraversal(node.right);

}

}

public void preOrderTraversal(Node node) {

if (node != null) {

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

preOrderTraversal(node.left);

preOrderTraversal(node.right);

}

}

public void postOrderTraversal(Node node) {

if (node != null) {

postOrderTraversal(node.left);

postOrderTraversal(node.right);

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

}

}

public static void main(String[] args) {

Tree1 tree = new Tree1();

tree.root = new Node("Owner");

tree.root.left = new Node("Supervisor 1");

tree.root.right = new Node("Supervisor 2");

tree.root.left.left = new Node("Employee 1");

tree.root.left.right = new Node("Employee 2");

tree.root.right.left = new Node("Employee 3");

tree.root.right.right = new Node("Employee 4");

System.out.println("In-order Traversal:");

tree.inOrderTraversal(tree.root);

System.out.println();

System.out.println("Pre-order Traversal:");

tree.preOrderTraversal(tree.root);

System.out.println();

System.out.println("Post-order Traversal:");

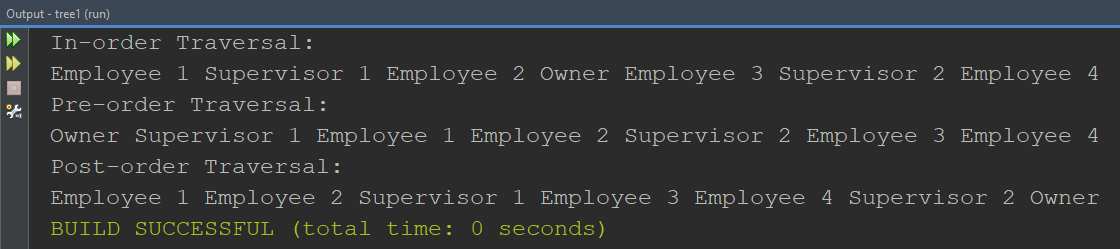
tree.postOrderTraversal(tree.root);

System.out.println();

}

}

***Output:***

******