DSA PRACTICE-1

1)  Maximum Subarray Sum – Kadane’s Algorithm

import java.util.Arrays;

class GfG {

    static int maxSubarraySum(int[] arr) {

        int res = arr[0];

        int maxEnding = arr[0];

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

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

             res = Math.max(res, maxEnding);

        }

        return res;

    }

    public static void main(String[] args) {

        int[] arr = {2, 3, -8, 7, -1, 2, 3};

        System.out.println(maxSubarraySum(arr));

    }

}

Test cases:

Input: {-2, -4}

Output: -2

Input: {2, 3, -8, 7, -1, 2, 3}

Output: 11

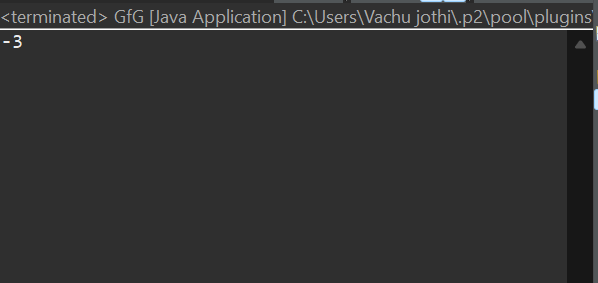
Hidden test case:

Input: {1,6,3,8,4}

Output: 22

Input: {3}

Output: -3



time complexity: O(N)

1. Maximum Product Subarray

package training;

public class Maxproduct {

static int maxProduct(int arr[]) {

int n = arr.length;

int result = arr[0];

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

int mul = 1;

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

mul \*= arr[j];

result = Math.max(result, mul);

}

}

return result;

}

public static void main(String[] args) {

int arr[] = { -2, 6, -3, -10, 0, 2 };

System.out.println(maxProduct(arr));

}

}

Input: {-2, 6, -3, -10, 0, 2}

Output: 180

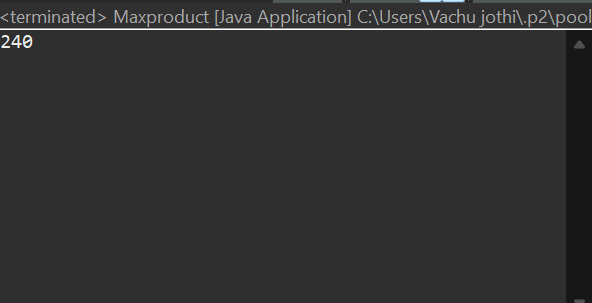
Input: {-1, -3, -10, 0, 60}

Output: 60

Hidden test case:

Input: { -2, 2,-3,4,5,-1 }

Output: 240



time complexity:O(N)

1. Search in a sorted and rotated Array

package training;

import java.util.\*;

public class GFG {

public static int pivotedSearch(List<Integer> arr, int key) {

int low = 0, high = arr.size() - 1;

while (low <= high) {

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

if (arr.get(mid) == key)

return mid;

if (arr.get(mid) >= arr.get(low)) {

if (key >= arr.get(low) && key < arr.get(mid))

high = mid - 1;

else

low = mid + 1;

}

else {

if (key > arr.get(mid) && key <= arr.get(high))

low = mid + 1;

else

high = mid - 1;

}

}

return -1;

}

public static void main(String[] args) {

List<Integer> arr1 = Arrays.asList(4, 5, 6, 7, 0, 1, 2);

int key1 = 0;

int result1 = pivotedSearch(arr1, key1);

System.out.println(result1);

List<Integer> arr2 = Arrays.asList(4, 5, 6, 7, 0, 1, 2);

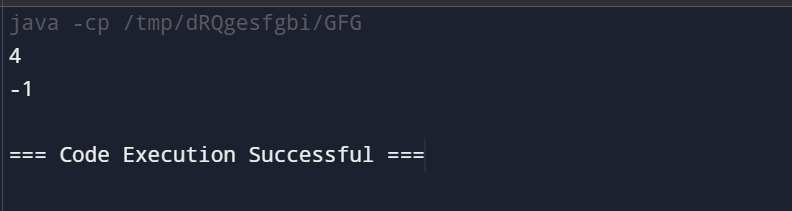
int key2 = 3;

int result2 = pivotedSearch(arr2, key2);

System.out.println(result2);

}

}



Time complexity: O(log n)

1. container-with-most-water:

import java.util.\*;

class GfG {

static int maxArea(int[] arr) {

int n = arr.length;

int area = 0;

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

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

area = Math.*max*(area, Math.*min*(arr[j], arr[i]) \* (j - i));

}

}

return area;

}

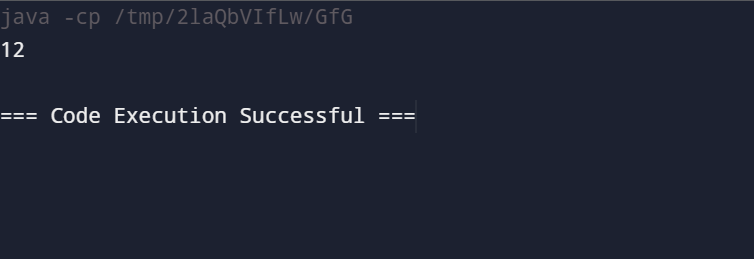
public static void main(String[] args){

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

System.***out***.println(*maxArea*(a));

}

}



Hidden Test Case:

Input: {3, -1,3,5,6,7}

Output: 15

Time Complexity: O(n)

1. Find the Factorial of a large number:

class GFG {

static void factorial(int n)

{

int res[] = new int[500];

res[0] = 1;

int res\_size = 1;

for (int x = 2; x <= n; x++)

res\_size = multiply(x, res, res\_size);

System.out.println("Factorial of given number is ");

for (int i = res\_size - 1; i >= 0; i--)

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

}

static int multiply(int x, int res[], int res\_size)

{

int carry = 0;

for (int i = 0; i < res\_size; i++) {

int prod = res[i] \* x + carry;

res[i] = prod % 10;

carry = prod / 10;

}

while (carry != 0) {

res[res\_size] = carry % 10;

carry = carry / 10;

res\_size++;

}

return res\_size;

}

public static void main(String args[])

{

factorial(100);

}

}

Test Cases:

Input: 100

Output: Factorial of given number is

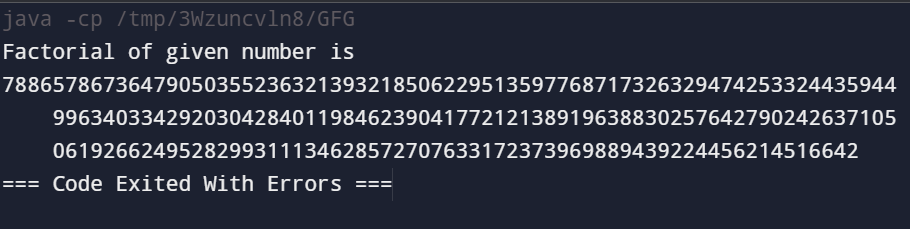
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000

Hidden TestCases:

Input: 200

Output: Factorial of given number is

788657867364790503552363213932185062295135977687173263294742533244359449963403342920304284011984623904177212138919638830257642790242637105061926624952829931113462857270763317237396988943922445621451664240254033291864131227428294853277524242407573903240321257405579568660226031904170324062351700858796178922222789623703897374720000000000000000000000000000000000000000000000000



Time Complexity: O(N^2.logn)

1. Trapping Rainwater Problem

import java.util.\*;

class GfG {

static int findWater(int[] arr) {

int n = arr.length;

int[] left = new int[n];

int[] right = new int[n];

int res = 0;

left[0] = arr[0];

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

left[i] = Math.max(left[i - 1], arr[i]);

right[n - 1] = arr[n - 1];

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

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

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

int minOf2 = Math.min(left[i - 1], right[i + 1]);

if (minOf2 > arr[i]) {

res += minOf2 - arr[i];

}

}

return res;

}

public static void main(String[] args) {

int[] arr = { 0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1 };

System.out.println(findWater(arr));

}

}

Test Cases:

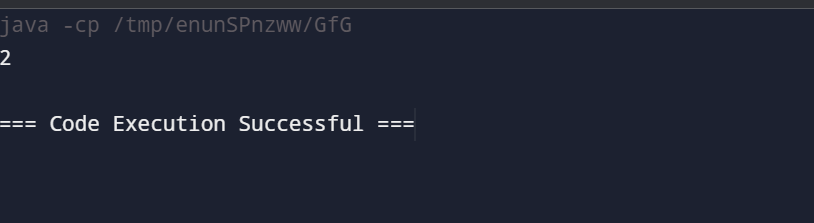
Input: { 0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1 }

Output: 6

Hidden Test cases:

Input: { 2,3,5,3,6,1 }

Output: 2



Time complexity:O(n)

1. Chocolate Distribution Problem

import java.util.Arrays;

class GfG {

static int findMinDiff(int[] arr, int m) {

int n = arr.length;

Arrays.sort(arr);

int minDiff = Integer.MAX\_VALUE;

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

int diff = arr[i + m - 1] - arr[i];

if (diff < minDiff)

minDiff = diff;

}

return minDiff;

}

public static void main(String[] args) {

int[] arr = {7, 3, 2, 4, 9, 12, 56};

int m = 3;

System.out.println(findMinDiff(arr, m));

}

}

Test Case:

Input: {7, 3, 2, 4, 9, 12, 56} ,m=3

Output=3

Hidden Test cases:

Input: {7, 3, 2, 4, 9, 12, 56}, m=4

Output: 3

Time complexity: O(n log n)

1. Merge Overlapping Intervals

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

class GfG {

static List<int[]> mergeOverlap(int[][] arr) {

Arrays.sort(arr, (a, b) -> Integer.compare(a[0], b[0]));

List<int[]> res = new ArrayList<>();

res.add(new int[]{arr[0][0], arr[0][1]});

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

int[] last = res.get(res.size() - 1);

int[] curr = arr[i];

if (curr[0] <= last[1])

last[1] = Math.max(last[1], curr[1]);

else

res.add(new int[]{curr[0], curr[1]});

}

return res;

}

public static void main(String[] args) {

int[][] arr = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};

List<int[]> res = mergeOverlap(arr);

for (int[] interval : res)

System.out.println(interval[0] + " " + interval[1]);

}

}

Test Cases:

Input: arr = [[1, 3], [2, 4], [6, 8], [9, 10]]

Output: [[1, 4], [6, 8], [9, 10]]

Hidden test cases:

Input: arr = {{2, 3}, {1, 1}, {2, 3}, {2, 6}}

Output: [[1,1],[2,6]]

Time Complexity : O(nlogn)

1. A Boolean Matrix Question

import java.util.\*;

public class BooleanMatrix {

public static void modify(int[][] mat) {

int M = mat.length;

int N = mat[0].length;

boolean[] rowFlag = new boolean[M];

boolean[] colFlag = new boolean[N];

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

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

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

rowFlag[i] = true;

colFlag[j] = true;

}

}

}

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

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

if (rowFlag[i] || colFlag[j]) {

mat[i][j] = 1;

}

}

}

}

public static void print(int[][] mat) {

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

for (int j = 0; j < mat[i].length; j++) {

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

}

System.out.println();

}

}

public static void main(String[] args) {

int[][] mat1 = {{1, 0}, {0, 0}};

System.out.println("Modified Matrix 1:");

modify(mat1);

print(mat1);

int[][] mat2 = {{0, 0, 0}, {0, 0, 1}};

System.out.println("Modified Matrix 2:");

modify(mat2);

print(mat2);

int[][] mat3 = {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}};

System.out.println("Modified Matrix 3:");

modify(mat3);

print(mat3);

}

}

Time Complexity: O(M\*N)

Test cases:

Input: {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}}

Output: {{1, 1, 1, 1}, {1, 1, 1, 1}, {1, 0, 1, 1}}

Hidden test cases:

Input: {{ 2, 1, 0, 2 }, { 3, 1, 3, 0 },{ 5, 2, 0, 5 }}

Output: {{ 1 1 1 1 },{1 1 1 1 },{5 1 0 5}}

1. Print a given matrix in spiral form:

import java.util.\*;

public class SpiralOrderMatrix {

public static List<Integer> spiralOrder(int[][] matrix) {

int m = matrix.length;

int n = matrix[0].length;

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

if (m == 0)

return result;

boolean[][] seen = new boolean[m][n];

int[] dr = {0, 1, 0, -1};

int[] dc = {1, 0, -1, 0};

int r = 0, c = 0;

int di = 0;

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

result.add(matrix[r][c]);

seen[r][c] = true;

int newR = r + dr[di];

int newC = c + dc[di];

if (0 <= newR && newR < m && 0 <= newC && newC < n

&& !seen[newR][newC]) {

r = newR;

c = newC;

} else {

di = (di + 1) % 4;

r += dr[di];

c += dc[di];

}

}

return result;

}

public static void main(String[] args) {

int[][] matrix = {

{ 1, 2, 3, 4 },

{ 5, 6, 7, 8 },

{ 9, 10, 11, 12 },

{ 13, 14, 15, 16 }

};

List<Integer> result = spiralOrder(matrix);

for (int num : result) {

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

}

}

}

**Time Complexity**: O(m×n)

Test Cases:

Input: matrix = {

{ 1, 2, 3, 4 },

{ 5, 6, 7, 8 },

{ 9, 10, 11, 12 },

{ 13, 14, 15, 16 }

Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

Hidden Test cases:

Input : matrix = {

{ 1, 2, 3},

{ 11, 6, 7},

{ 2, 10, 1},

{ 3, 2, 5 }

Output: 1 2 3 7 1 5 2 3 2 11 6 10

13) Check if given Parentheses expression is balanced or not .

import java.util.\*;

public class Paranthesis {

static String check(String s) {

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

for(char i:s.toCharArray()) {

if(i=='(') {

st.push(i);

}

else if(i==')'){

if(!st.isEmpty()) {

st.pop();

}

else {

return "not Balanced";

}

}}

return st.isEmpty()?"Balanced":"not Balanced";

}

public static void main(String[] args) {

String res=check("(())");

System.out.println(res);

String res1=check("“((()))()()");

System.out.println(res1);

String res2=check("())((())”");

System.out.println(res2);

}

}

Test cases:

Input: str = “((()))()()”

Output: Balanced

Input: str = “())((())”

Output: Not Balanced

Hidden Test cases: s = "{([]"

Input: s = "{([]"

Output: Not balanced

14) Check if two Strings are Anagrams of each other:

import java.util.\*;

public class ValidAnagram {

public static boolean valid(String s1, String s2) {

char[] lst1=s1.toCharArray();

char[] lst2=s2.toCharArray();

if(lst1.length!=lst2.length) {

return false;

}

Arrays.sort(lst1);

Arrays.sort(lst2);

return Arrays.equals(lst1,lst2);

}

public static void main(String[] args) {

String s = "geeks";

String t = "skeeg";

System.out.println(valid(s,t));

String s1 = "allergy";

String t1 = "allergic";

System.out.println(valid(s1,t1));

String s2 = "g";

String t2 = "g";

System.out.println(valid(s2,t2));

}

}

Time Complexity: O(n log n)

Test Cases:

Input: s1 = “geeks” s2 = “kseeg”

Output: true

Hidden Test Cases:

Input: s1 = “hello” s2 = “hai”

Output: false

15) . Longest Palindromic Substring:

import java.util.\*;

public class longPalindrome {

public static String longestPalindrome(String str) {

if (str == null || str.length() == 0) {

return "";

}

int n = str.length();

int start = 0;

int maxLength = 1;

boolean[][] dp = new boolean[n][n];

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

dp[i][i] = true;

}

for (int length = 2; length <= n; length++) {

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

int j = i + length - 1;

if (str.charAt(i) == str.charAt(j)) {

if (length == 2 || dp[i + 1][j - 1]) {

dp[i][j] = true;

if (length > maxLength) {

maxLength = length;

start = i;

}

}

}

}

}

return str.substring(start, start + maxLength);

}

public static void main(String[] args) {

String str1 = "forgeeksskeegfor";

System.out.println("Longest Palindromic Substring: " + longestPalindrome(str1));

String str2 = "Geeks";

System.out.println("Longest Palindromic Substring: " + longestPalindrome(str2));

String str3 = "abc";

System.out.println("Longest Palindromic Substring: " + longestPalindrome(str3));

String str4 = "";

System.out.println("Longest Palindromic Substring: " + longestPalindrome(str4));

}

}

Time Complexity: O(n^2)

Test cases:

Input: str = “Geeks”

Output: “ee”

Input: str = “abc”

Output: “a”

16) Longest Common Prefix using Sorting

import java.util.\*;

public class CommonPrefix {

static String Common(String[] arr) {

Arrays.sort(arr);

String f=arr[0];

String l=arr[arr.length-1];

int n=Math.min(f.length(),l.length());

String res="";

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

if(f.charAt(i)==l.charAt(i)){

res+=f.charAt(i);

}

else{

break;

}

}

return res.isEmpty()? "-1":res;

}

public static void main(String[] args) {

String[] arr1= {"geeksforgeeks", "geeks", "geek", "geezer"};

System.out.println(Common(arr1));

String[] arr2= {"hello","world"};

System.out.println(Common(arr2));

}

}

Test case:

Input: arr[] = [“geeksforgeeks”, “geeks”, “geek”, “geezer”]

Output: gee

Hidden test case:

Input: arr = { "hello", "hemsiphere", "hero" }

Output: he

Time Complexity: O(n)

17) Delete middle element of a stack

import java.util.Stack;

import java.util.Vector;

public class Main {

public static void main(String[] args) {

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

st.push('1');

st.push('2');

st.push('3');

st.push('4');

st.push('5');

st.push('6');

st.push('7');

Vector<Character> v = new Vector<Character>();

while (!st.empty()) {

v.add(st.pop());

}

int n = v.size();

if (n % 2 == 0) {

int target = (n / 2);

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

if (i == target) continue;

st.push(v.get(i));

}

} else {

int target = (int) Math.ceil(n / 2);

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

if (i == target) continue;

st.push(v.get(i));

}

}

System.out.print("Printing stack after deletion of middle: ");

while (!st.empty()) {

char p = st.pop();

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

}

}

}

Test case:

Input : Stack[] = [1, 2, 3, 4, 5]

Output : Stack[] = [1, 2, 4, 5]

Hidden test case:

Input: Stack[]=[ 7 ,2 ,4 ,5,8, 0, 4]

Output: [7, 2, 4, 8, 0, 4]

**Time complexity**: O(n).

18) . Next Greater Element (NGE) for every element in given Array

import java.util.\*;

public class NextGreat {

static void Greater(int[] arr) {

int n = arr.length;

int[] nge = new int[n];

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

Arrays.fill(nge, -1);

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

while (!stack.isEmpty() && stack.peek() <= arr[i]) {

stack.pop();

}

if (!stack.isEmpty()) {

nge[i] = stack.peek();

}

stack.push(arr[i]);

}

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

System.out.println(arr[i] + " --> " + nge[i]);

}

}

public static void main(String[] args) {

int[] arr1 = {4, 5, 2, 25};

System.out.println("Next Greater Elements for the array " + Arrays.toString(arr1) + ":");

Greater(arr1);

System.out.println();

int[] arr2 = {13, 7, 6, 12};

System.out.println("Next Greater Elements for the array " + Arrays.toString(arr2) + ":");

Greater(arr2);

}

}

Time Complexity: O(n)

Test case:

Input: arr[] = { 11, 13, 21, 3 }

Output:

11 -- 13

13 -- 21

21 -- -1

3 -- -1

Hidden test case:

Input: arr[] = { 1, 3, 2, 3 }

Output:

1 -- 3

3 -- -1

2 -- 3

3 -- -1

Time Complexity: O(n)

19) Print Right View of a Binary Tree

import java.util.\*;

class Node {

int val;

Node left, right;

Node(int val) {

this.val = val;

left = right = null;

}

}

public class BinaryTreeRight {

public static List<Integer> rightView(Node root) {

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

if (root == null) {

return result;

}

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

queue.add(root);

while (!queue.isEmpty()) {

int cap = queue.size();

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

Node curr = queue.poll();

if (i == cap - 1) {

result.add(curr.val);

}

if (curr.left != null) {

queue.add(curr.left);

}

if (curr.right != null) {

queue.add(curr.right);

}

}

}

return result;

}

public static void main(String[] args) {

Node root = new Node(1);

root.left = new Node(2);

root.right = new Node(3);

root.left.right = new Node(5);

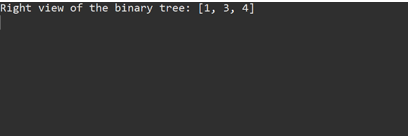
root.right.right = new Node(4);

List<Integer> rightView = rightView(root);

System.out.println("Right view of the binary tree: " + rightView);

}

}



Time complexity: O(n)

20)Maximum height or depth of the binary tree

import java.util.\*;

class Node {

int val;

Node left, right;

Node(int val) {

this.val = val;

this.left = this.right = null;

}

}

public class TreeHeight {

static int maxDepth(Node root) {

if (root == null) {

return 0;

}

int leftDepth = maxDepth(root.left);

int rightDepth = maxDepth(root.right);

return Math.max(leftDepth, rightDepth) + 1;

}

public static void main(String[] args) {

Node root = new Node(1);

root.left = new Node(2);

root.right = new Node(3);

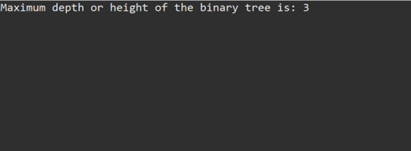
root.left.left = new Node(4);

root.left.right = new Node(5);

System.out.println("Maximum depth or height of the binary tree is: " + maxDepth(root));

}

}



Time Complexity: O(n)