Problems: Sriram N  
 22AD139

1. Max Subarray Sum:

import java.io.\*;

import java.util.\*;

public class msas{

public static void main(String msas[]){

Scanner sc = new Scanner(System.in);

int n= sc.nextInt();

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

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

nums[i]=sc.nextInt();

}

int c=nums[0];

int m=nums[0];

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

m=Math.max(m+nums[j],nums[j]);

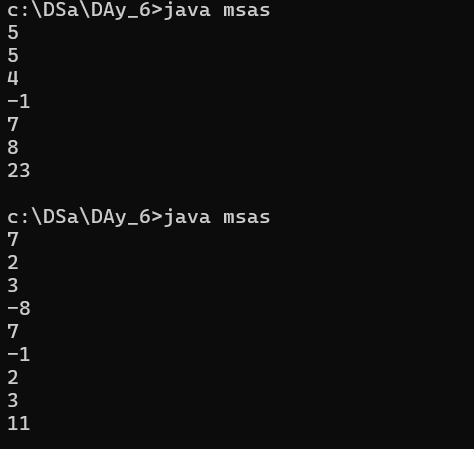
c=Math.max(m,c);

}

System.out.println(c);

}}

O/P:



Time Complexity – o(n) Space Complexity – o(1)

1. Max Product Array

import java.io.\*;

import java.util.\*;

public class msap{

public static void main(String msap[]){

Scanner sc = new Scanner(System.in);

int n= sc.nextInt();

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

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

nums[i]=sc.nextInt();

}

int m = nums[0];

int k = nums[0];

int c = nums[0];

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

if ( nums[i] < 0) {

int temp = m;

m = k;

k = temp;

}

m = Math.max( nums[i], m \* nums[i]);

k = Math.min( nums[i], k \* nums[i]);

c = Math.max(c, m);

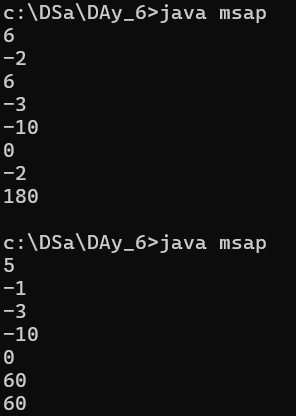
}

System.out.println(c);

}

}

O/P:



Time Complexity – o(n) Space Complexity – o(1)

1. Search in RSA:

import java.util.\*;

import java.lang.\*;

import java.io.\*;

public class sirsa

{

public static void main (String sirsa[]) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] nums = new int[n];

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

nums[i] = sc.nextInt();

}

int t = sc.nextInt();

int result = -1;

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

if (nums[i] == t) {

result = i;

break;

}

}

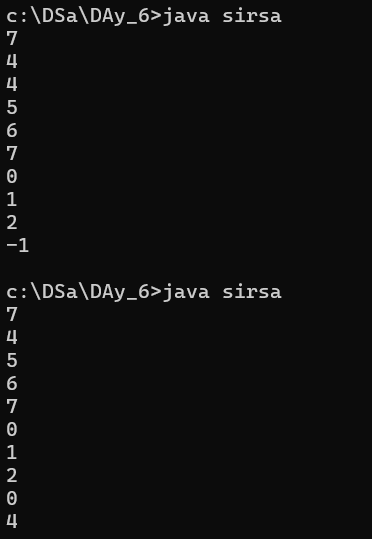
System.out.println(result);

sc.close();

}

}

O/P:



Time Complexity – o(n) Space Complexity – o(n)

1. Container With most Water:

import java.util.\*;

import java.lang.\*;

import java.io.\*;

public class cwmw

{

public static void main (String cwmw[]) throws java.lang.Exception

{

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] h = new int[n];

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

h[i] = sc.nextInt();

}

int i = 0, j = h.length - 1;

int m = 0;

while (i <= j) {

int w = j - i;

int l = Math.min(h[i], h[j]);

m = Math.max(m, l \* w);

if (h[i] > h[j]) {

j--;

} else {

i++;

}

}

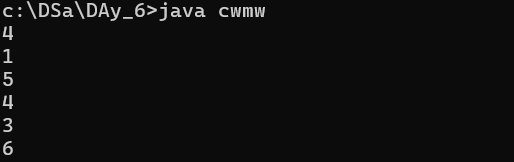
System.out.println(m);

sc.close();

}

}

O/P:



Time Complexity – o(n) Space Complexity – o(1)

1. Factorial of large number:  
   import java.io.\*;

import java.util.\*;

import java.math.BigInteger;

public class fact{

public static void main(String fact[]){

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

BigInteger k = BigInteger.ONE;

while (n > 1) {

k = k.multiply(BigInteger.valueOf(n));

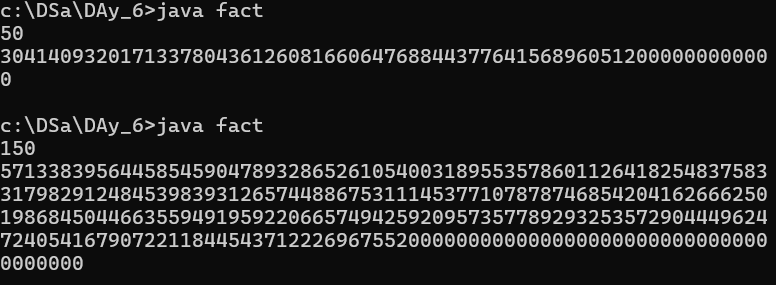
n--;

}

System.out.println(k);

sc.close();

}}

O/P:  


Time Complexity – o(n) Space Complexity – o(1)

1. Trapping Rainwater:

import java.io.\*;

import java.util.\*;

import java.lang.\*;

public class trap{

public static void main(String trap[]){

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

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

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

h[i]=sc.nextInt();

}

int l=0,r=h.length-1;

int a=h[l],b=h[r];

int c=0;

while(l<=r){

if (a<=b){

a=Math.max(a,h[l]);

c+=a-h[l];

l++;

}

else{

b=Math.max(b,h[r]);

c+=b-h[r];

r--;

}

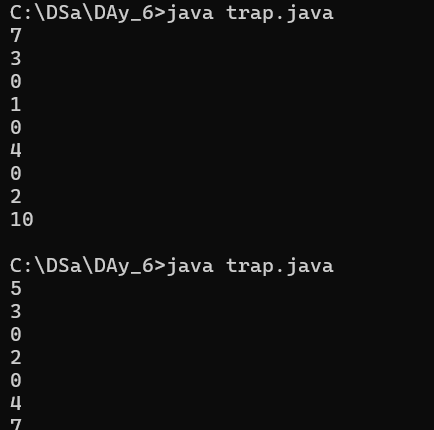
}

System.out.println(c);

}

}

O/P:



Time Complexity: o(n) Soace Complexity: o(1)

1. Choco Distribution:

import java.io.\*;

import java.util.\*;

public class chocodis {

public static void main(String chocodis[]) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt(), m = sc.nextInt();

int[] a = new int[n];

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

Arrays.sort(a);

int c = Integer.MAX\_VALUE;

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

c = Math.min(c, a[i + m - 1] - a[i]);

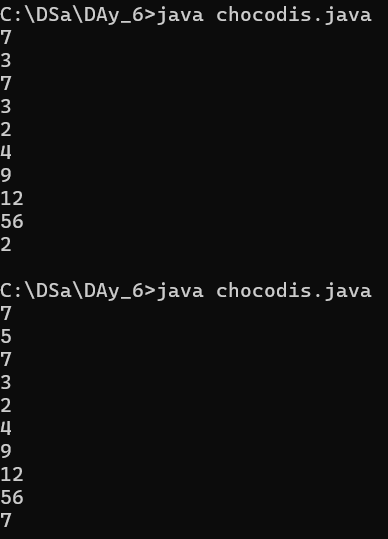
}

System.out.println(c);

}

}

O/P:



Time Complexity: o(nlogn) Space Comlexity : o(n)

1. Merge Intervals:

import java.util.\*;

public class mergeinter {

public static void main(String mi[]) {

Scanner s = new Scanner(System.in);

int n = s.nextInt();

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

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

intervalsList.add(new int[]{s.nextInt(), s.nextInt()});

}

int[][] intervals = intervalsList.toArray(new int[intervalsList.size()][]);

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

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

m.add(intervals[0]);

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

int[] p = m.get(m.size() - 1);

if (intervals[i][0] <= p[1]) {

p[1] = Math.max(p[1], intervals[i][1]);

} else {

m.add(intervals[i]);

}

}

for (int[] i : m) {

System.out.println(Arrays.toString(i));

}

}

}

O/P:

Time Complexity: o(nlogn) Space Comlexity : o(n)

1. Bool Matrix:

import java.io.\*;

import java.util.\*;

public class BooleanMatrix {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int r = sc.nextInt(), c = sc.nextInt();

int[][] mat = new int[r][c];

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

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

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

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

mat[i][j] = sc.nextInt();

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

rows.add(i);

cols.add(j);

}

}

}

for (int i : rows) Arrays.fill(mat[i], 1);

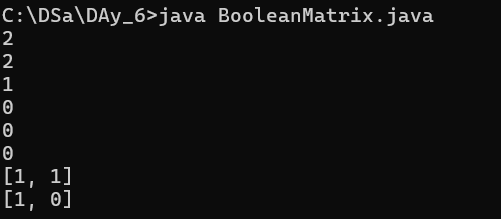
for (int j : cols) for (int i = 0; i < r; i++) mat[i][j] = 1;

for (int[] row : mat) System.out.println(Arrays.toString(row));

}

}

O/P:



Time Complexity: O(r \* c) Space Complexity: O(r \* c)

1. Spiral matrix:

import java.io.\*;

import java.util.\*;

public class SpiralMatrix {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int r = sc.nextInt(), c = sc.nextInt();

int[][] mat = new int[r][c];

for (int i = 0; i < r; i++) for (int j = 0; j < c; j++) mat[i][j] = sc.nextInt();

int top = 0, bottom = r - 1, left = 0, right = c - 1;

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

while (top <= bottom && left <= right) {

for (int i = left; i <= right; i++) res.add(mat[top][i]);

top++;

for (int i = top; i <= bottom; i++) res.add(mat[i][right]);

right--;

if (top <= bottom) for (int i = right; i >= left; i--) res.add(mat[bottom][i]);

bottom--;

if (left <= right) for (int i = bottom; i >= top; i--) res.add(mat[i][left]);

left++;

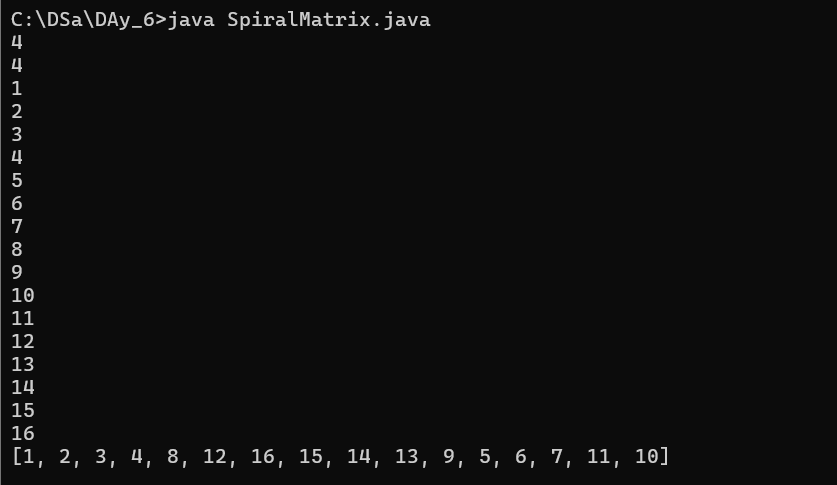
}

System.out.println(res);

}

}

O/P:



Time Complexity: O(n) Space Complexity: O(n)

1. Balaance:

import java.io.\*;

import java.util.\*;

public class BalPar {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String s = sc.next();

int c = 0;

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

c += (ch == '(') ? 1 : -1;

if (c < 0) {

System.out.println("Not Balanced");

return;

}

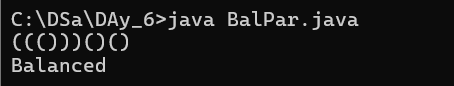
}

System.out.println(c == 0 ? "Balanced" : "Not Balanced");

}

}

O/P:



Time Complexity: o(n) Soace Complexity: o(1)

1. Anagrams:  
   import java.io.\*;

import java.util.\*;

public class anag {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String s1 = sc.next(), s2 = sc.next();

char[] a = s1.toCharArray(), b = s2.toCharArray();

Arrays.sort(a);

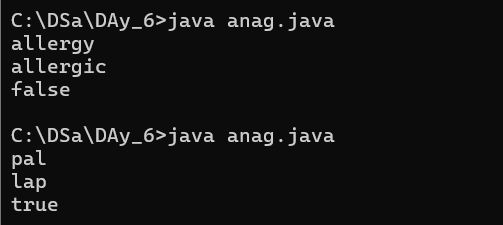
Arrays.sort(b);

System.out.println(Arrays.equals(a, b));

}

}

O/P:



Time Complexity: o(nlogn) Soace Complexity: o(n)

1. Palindrome:

import java.io.\*;

import java.util.\*;

public class longpals {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String s = sc.next();

String res = "";

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

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

String sub = s.substring(i, j + 1);

if (sub.equals(new StringBuilder(sub).reverse().toString()) && sub.length() > res.length()) {

res = sub;

}

}

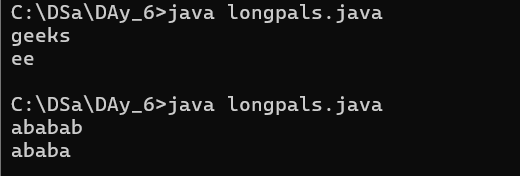
}

System.out.println(res);

}

}

O/P:



Time Complexity: o(n\*\*3) Soace Complexity: o(n)

1. Longest Common Prefix using Sorting :

import java.io.\*;

import java.util.\*;

public class lcaps {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

String[] a = new String[n];

for (int i = 0; i < n; i++) a[i] = sc.next();

Arrays.sort(a);

String s1 = a[0], s2 = a[n - 1];

int i = 0;

while (i < s1.length() && s1.charAt(i) == s2.charAt(i)) i++;

System.out.println(i > 0 ? s1.substring(0, i) : "-1");

}

}



TC: O(n log n + m), SC: O(1)

1. Delete Middle Stack

import java.io.\*;

import java.util.\*;

public class dms {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

for (int i = 0; i < n; i++) st.push(sc.nextInt());

deleteMiddle(st, n / 2);

System.out.println(st);

}

public static void deleteMiddle(Stack<Integer> st, int mid) {

if (mid == 0) {

st.pop();

return;

}

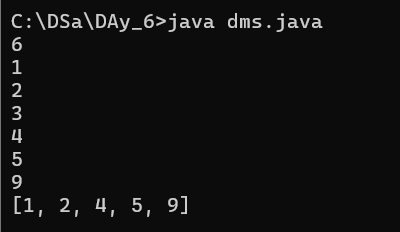
int top = st.pop();

deleteMiddle(st, mid - 1);

st.push(top);

}

}

****

**TC: O(n), SC: O(n)**

1. Next greatest element:

import java.io.\*;

import java.util.\*;

public class nge {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] a = new int[n], res = new int[n];

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

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

a[i] = sc.nextInt();

while (!st.isEmpty() && st.peek() <= a[i]) st.pop();

res[i] = st.isEmpty() ? -1 : st.peek();

st.push(a[i]);

}

System.out.println(Arrays.toString(res));

}

}



1. Right view:

import java.util.ArrayList;

import java.util.List;

class TreeNode {

int val;

TreeNode left;

TreeNode right;

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

}

class Solution {

int maxlevel = 0;

public List<Integer> rightSideView(TreeNode root) {

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

right(root, 1, list);

return list;

}

void right(TreeNode root, int level, List<Integer> list) {

if (root == null) {

return;

}

if (maxlevel < level) {

list.add(root.val);

maxlevel = level;

}

right(root.right, level + 1, list);

right(root.left, level + 1, list);

}

public static void main(String[] args) {

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

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

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

Solution solution = new Solution();

List<Integer> result = solution.rightSideView(root);

System.out.println(result);

}

}

O/P:



TC: O(n)

1. Depth of bst

class TreeNode {

int val;

TreeNode left;

TreeNode right;

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

}

class Solution {

public int maxDepth(TreeNode root) {

if (root == null) return 0;

int left = maxDepth(root.left);

int right = maxDepth(root.right);

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

}

public static void main(String[] args) {

TreeNode root = new TreeNode(3);

root.left = new TreeNode(9);

root.right = new TreeNode(20);

root.right.left = new TreeNode(15);

root.right.right = new TreeNode(7);

Solution solution = new Solution();

int result = solution.maxDepth(root);

System.out.println(result);

}

}

O/P:



TC : O(n)