# Assignment 6

## Student Name: Manish Kumar UID:22BCS10249

## Branch: BE-CSE (General) Section/Group: FL\_IOT-602 A

## Semester:6th Date of Performance: 18-03-25

## Subject Name: Advanced Programming Lab-2 Subject Code: 22CSP-351

# Aim: 108. Convert Sorted Array to Binary Search Tree

# Implementation/ Code:

# public class TreeNode {

# int val;

# TreeNode left;

# TreeNode right;

# TreeNode() {}

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

# TreeNode(int val, TreeNode left, TreeNode right) {

# this.val = val;

# this.left = left;

# this.right = right;

# }

# }

# class Solution {

# public TreeNode sortedArrayToBST(int[] nums) {

# return helper(nums, 0, nums.length - 1);

# }

# private TreeNode helper(int[] nums, int left, int right) {

# if (left > right) return null;

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

# TreeNode root = new TreeNode(nums[mid]);

# root.left = helper(nums, left, mid - 1);

# root.right = helper(nums, mid + 1, right);

# return root;

# }

# }

# Output:

# 

# Aim: 191. Number of 1 Bits

# Implementation/ Code:

# //976. Largest Perimeter Triangle

# class Solution {

# public int hammingWeight(int n) {

# return Integer.bitCount(n ^ 0);

# }

# }

# Output:

# 

# Aim: 912. Sort an Array

# Implementation/ Code:

# class Solution {

# public int[] sortArray(int[] nums) {

# PriorityQueue<Integer> pq = new PriorityQueue<>();

# for(int i : nums){

# pq.add(i);

# }

# int i =0;

# while(!pq.isEmpty()){

# nums[i++] = pq.poll();

# }

# return nums;

# }

# }

# Output:

# 

# 4. 53. Maximum Subarray

# Implementation/ Code:

# class Solution {

# public int maxSubArray(int[] nums) {

# int res = nums[0];

# int total = 0;

# for (int n : nums) {

# if (total < 0) {

# total = 0;

# }

# total += n;

# res = Math.max(res, total);

# }

# return res;

# }

# }

# Output:

# 

# 5.Aim: 932. Beautiful Array

# Implementation/ Code:

// 932. Beautiful Array

class Solution {

public int[] beautifulArray(int n) {

int[] ans = new int[n];

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

ans[i] = i+1;

}

recursion(ans, 0, n-1);

return ans;

}

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

if(left >= right)

return;

ArrayList<Integer> l = new ArrayList<>();

ArrayList<Integer> r = new ArrayList<>();

boolean alt = true;// Not worry about whether the factor of the interval is even or odd too much, they can be grouped by

// just picking one and skip one

for(int i = left; i <= right; i++){ // picking the elements and put them into the two groups

if(alt)

l.add(arr[i]);

else

r.add(arr[i]);

alt = !alt;

}

for(int i = left; i <= right; i++){ // merging them into the final array

if(!l.isEmpty())

arr[i] = l.remove(0);

else

arr[i] = r.remove(0);

}

recursion(arr, left, (right+left)/2);

recursion(arr, (left+right)/2+1, right);

}

# Output:

# 

# 6.Aim: 372. Super Pow

# Implementation/ Code:

# class Solution {

# private final int MOD = 1337;

# public int superPow(int a, int[] b) {

# return helper(a, b);

# }

# private int helper(int a, int[] b) {

# int res = 1;

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

# res = (int) (res \* pow(a, b[i])) % MOD;

# a = (int) pow(a, 10);

# }

# return res;

# }

# private int pow(int a, int b) {

# int res = 1;

# a %= MOD;

# for (int i = 0; i < b; i++) {

# res = (res \* a) % MOD;

# }

# return res;

# }

# }

# Output:

# 

# 7. 218. The Skyline Problem

# class TopNode {

# int x;

# int h;

# TopNode next;

# TopNode() {

# }

# TopNode(int x, int h) {

# this.x = x;

# this.h = h;

# }

# void insert(TopNode n) {

# n.next = next;

# next = n;

# }

# }

# class Solution {

# static final int LEFT=0, RIGHT=1, HEIGHT=2;

# public List<List<Integer>> getSkyline(int[][] buildings) {

# TopNode head = new TopNode(0,0);

# head.insert(new TopNode(Integer.MAX\_VALUE, 0));

# TopNode start = head;

# 

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

# int[] b = buildings[i];

# int bL = buildings[i][LEFT];

# int bR = buildings[i][RIGHT];

# int bH = buildings[i][HEIGHT];

# //System.out.println(Arrays.toString(buildings[i]));

# while (bL >= start.next.x) { start = start.next; }

# //System.out.println(start.toString());

# for (TopNode t = start ; bR > t.x; t = t.next) {

# //System.out.println(head.toString());

# if (bH <= t.h) {

# continue;

# }

# TopNode stop = t;

# while (stop.next != null && stop.next.x < bR && stop.next.h <= bH ) {

# stop = stop.next;

# }

# 

# if (bL <= t.x) {

# if (bR >= stop.next.x) {

# t.next = stop.next;

# t.h = bH;

# }

# else if (t == stop) {

# t.insert(new TopNode(bR,t.h));

# t.h = bH;

# break;

# }

# else {

# stop.x = bR;

# t.h = bH;

# t.next = stop;

# break;

# }

# }

# else {

# if (bR >= stop.next.x) {

# if (t == stop) {

# t.insert(new TopNode(bL, bH));

# }

# else {

# t.next = stop;

# stop.x = bL;

# stop.h = bH;

# }

# break;

# }

# else if (t == stop) {

# t.insert(new TopNode(bL,bH));

# t.next.insert(new TopNode(bR,t.h));

# break;

# }

# else {

# t.next = stop;

# t.insert(new TopNode(bL,bH));

# stop.x = bR;

# break;

# }

# }

# t = stop;

# }

# }

# List<List<Integer>> skyline = new ArrayList<>();

# if (head.h == 0)

# head = head.next;

# while (head != null) {

# int height = head.h;

# skyline.add(List.of(head.x, height));

# while ( (head = head.next) != null && head.h == height) {}

# }

# return skyline;

# }

# }

# Output

# 