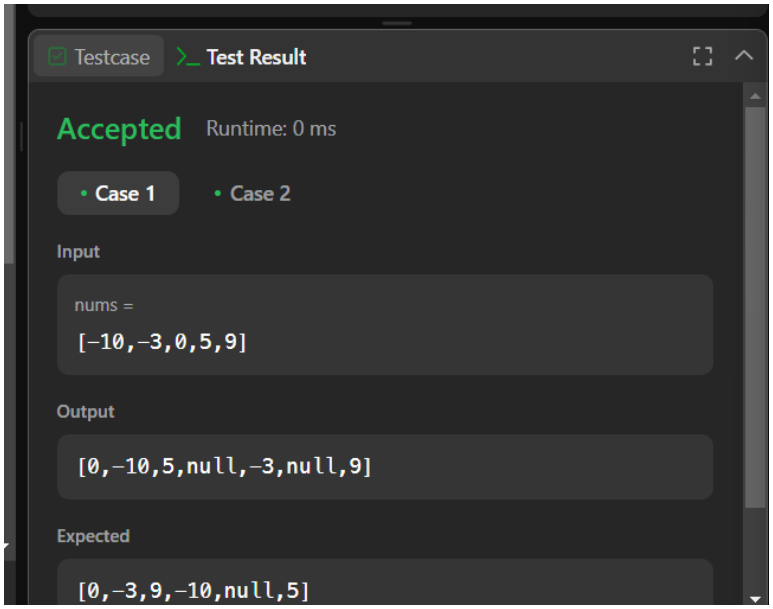
NAME: LAKSHAY BASKOTRA UID : 22BCS15016

Q1: **Convert Sorted Array to Binary Search Tree**

Given an integer array nums where the elements are sorted in **ascending order**, convert *it to a **height-balanced** binary search tree*.

```
/**
 * Definition for a binary tree node.
 * 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 createBST(int nums[] , int x , int y){
        if(x>y){
            return null;
        }
        int mid = (x+y)/2;
        TreeNode root = new TreeNode(nums[mid]);
        root.left = createBST(nums,x,mid-1);
        root.right = createBST(nums,mid+1,y);
```

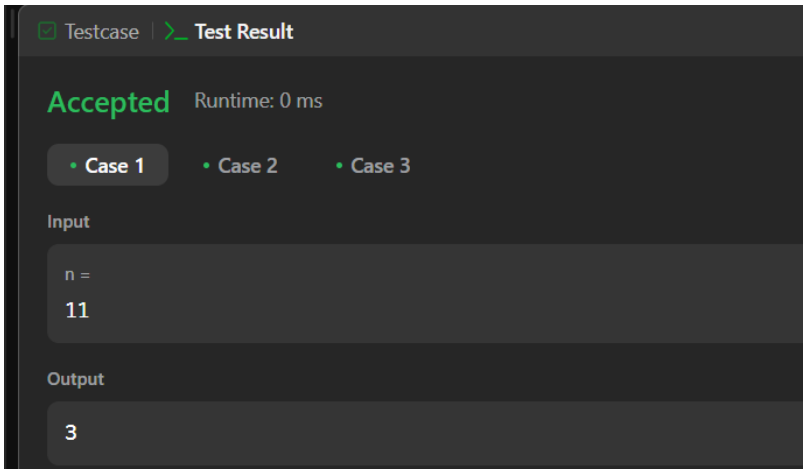
```
        return root;
    }
    public TreeNode sortedArrayToBST(int[] nums) {
        return createBST(nums , 0 , nums.length-1);
    }
}
```



Q2: **Number of 1 Bits**

Given a positive integer n, write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

```
public class Solution {
    public int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            count += (n & 1);
            n >>>= 1;
        }
        return count;
    }
}
```



Q3: [Sort an Array]

Given an array of integers nums, sort the array in ascending order and return it.

You must solve the problem **without using any built-in** functions in O(nlog(n)) time complexity and with the smallest space complexity possible.

```
class Solution {

    public void merge (int nums[] , int st , int m, int en )
    {
int n1 = m-st+1;
int n2 = en-m;
int ar1[] = new int [n1];
int ar2[] = new int [n2];

for(int i = 0 ; i<n1;i++)
{

    ar1[i] = nums[st+i];

}
```

```
for(int i = 0 ; i<n2;i++)
{

   ar2[i] = nums[m+1+i];

}
int i = 0 ;
int j = 0 ;
int k = st;
while(i<n1 && j<n2)
{
   if(ar1[i]<=ar2[j])
   {
      nums[k] = ar1[i];
      i++;
   }

   else{
      nums[k] = ar2[j];
      j++;
   }
   k++;
}
while(i<n1)
{
    nums[k] = ar1[i];
      i++;
      k++;

}
```

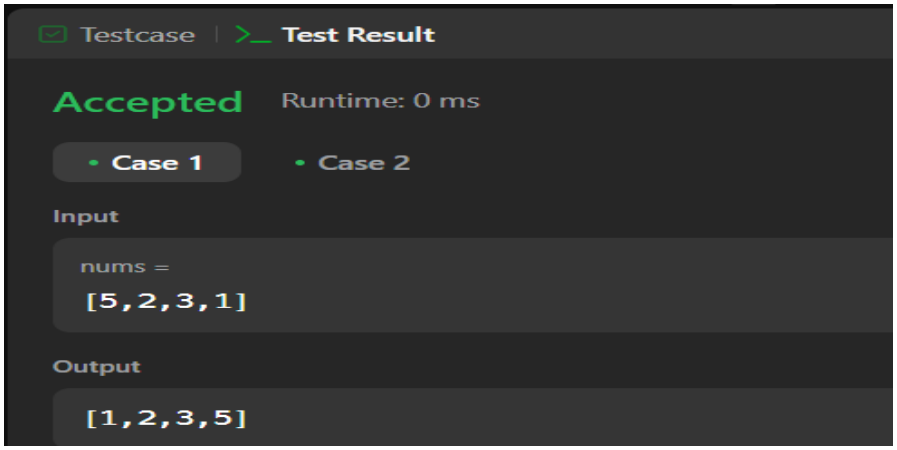
```
while(j<n2)
{
     nums[k] = ar2[j];
       j++;
       k++;
}



}
    public void mergesort(int nums[], int st , int en)
    {
       if(st<en)
       {
          int m = st +(en-st)/2;
          mergesort(nums,st,m);
          mergesort(nums,m+1, en);
          merge(nums,st,m,en);
       }

    }
    public int[] sortArray(int[] nums) {
       mergesort(nums,0,nums.length-1);
       return nums;


    }
}
```



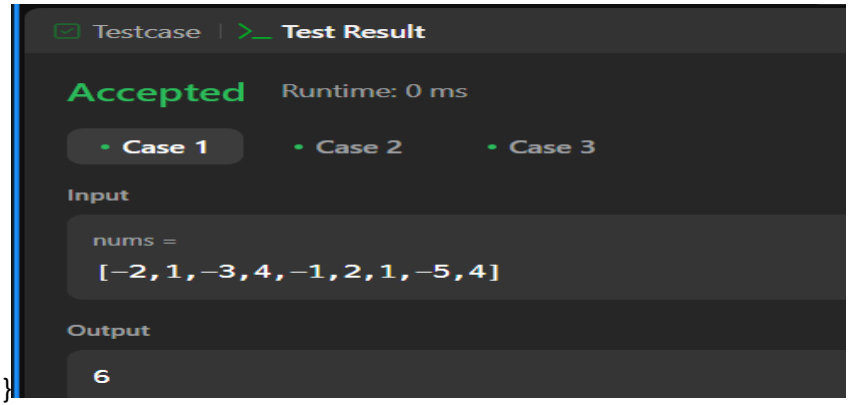
Q4: **Maximum Subarray**

Given an integer array nums, find the subarray with the largest sum, and return *its sum*.

```
public class Solution {
    public int maxSubArray(int[] nums) {
        int maxSum = nums[0];
        int currentSum = nums[0];

        for (int i = 1; i < nums.length; i++) {
            currentSum = Math.max(nums[i], currentSum + nums[i]);
            maxSum = Math.max(maxSum, currentSum);
        }

        return maxSum;
    }
```



}

Q5: **Beautiful Array**

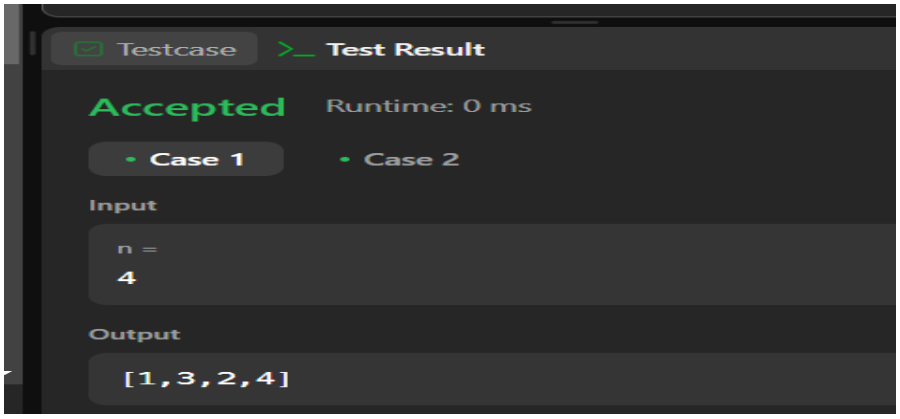
An array nums of length n is **beautiful** if:

- nums is a permutation of the integers in the range [1, n].
- For every 0 <= i < j < n, there is no index k with i < k < j where 2 * nums[k] == nums[i] + nums[j].

Given the integer n, return *any **beautiful** array* nums *of length* n. There will be at least one valid answer for the given n.

```
class Solution {
    public int[] beautifulArray(int N) {
        int[] res = new int[N];
        if (N == 1)
        {
            return new int[] {1};
        }
        else if (N == 2)
        {
            return new int[] {1, 2};
        }
        else
        {
            int[] odds = beautifulArray((N + 1) / 2);
            int[] even = beautifulArray(N / 2);
            for (int i = 0; i < odds.length; i ++)
            {
                res[i] = odds[i] * 2 - 1;
            }
            for (int j = 0; j < even.length; j ++)
            {
                res[odds.length + j] = even[j] * 2;
            }
        }
```

```
        return res;
    }
}
```



Q6: **Super Pow**

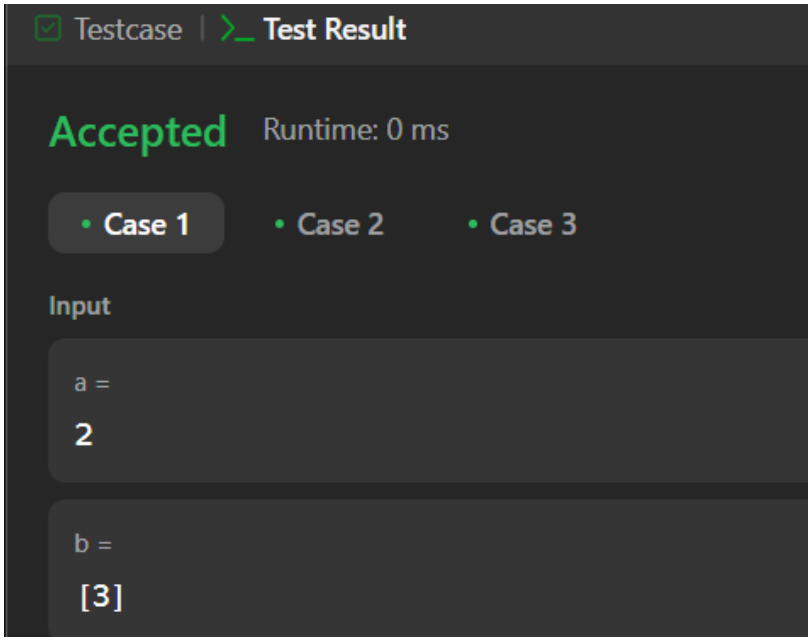
Your task is to calculate a^b mod 1337 where a is a positive integer and b is an extremely large positive integer given in the form of an array.

```
class Solution {

    public int superPow(int a, int[] b) {
        if (a % 1337 == 0) return 0;
        int result = 1;
        for (int digit : b) {
            result = modPow(result, 10) * modPow(a, digit) % 1337;
        }
        return result;
    }

    private int modPow(int base, int exponent) {
        base %= 1337;
        int result = 1;
        for (int i = 0; i < exponent; i++) {
```

```
            result = (result * base) % 1337;
        }
        return result;
    }
}
```



Q7: **The Skyline Problem**

A city's **skyline** is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Given the locations and heights of all the buildings,
return *the **skyline** formed by these buildings collectively*.

The geometric information of each building is given in the array buildings where buildings[i] = [$left_i$, $right_i$, $height_i$]:

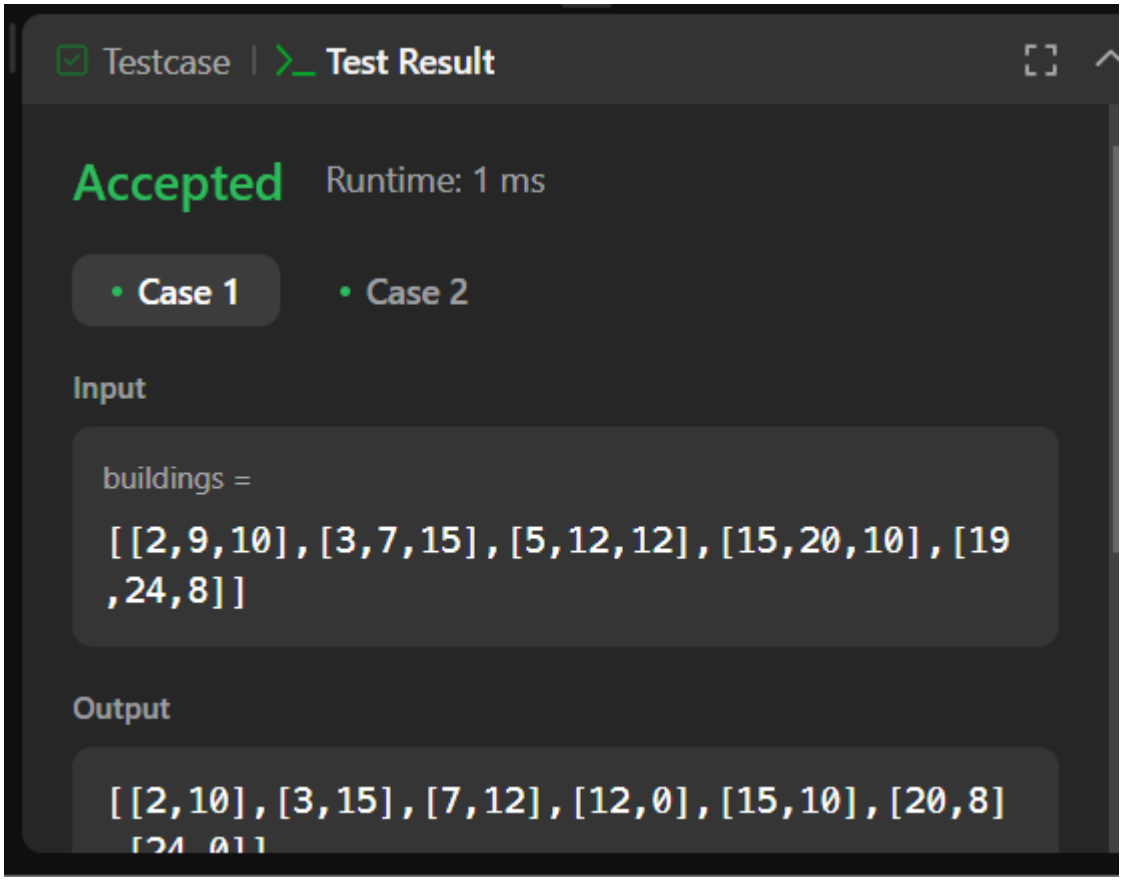
- $left_i$ is the x coordinate of the left edge of the i^{th} building.
- $right_i$ is the x coordinate of the right edge of the i^{th} building.
- $height_i$ is the height of the i^{th} building.

You may assume all buildings are perfect rectangles grounded on an absolutely flat surface at height 0.

The **skyline** should be represented as a list of "key points" **sorted by their x-coordinate** in the form [[x_1,y_1],[x_2,y_2],...]. Each key point is the left endpoint of some horizontal segment in the skyline except the last point in the list, which always has a y-coordinate 0 and is used to mark the skyline's

termination where the rightmost building ends. Any ground between the leftmost and rightmost buildings should be part of the skyline's contour.

```
class Solution {
    public List<List<Integer>> getSkyline(int[][] B) {
        int[][] H = new int[2 * B.length][2];
        for (int i = 0; i < B.length; i++) {
            H[i * 2] = new int[]{B[i][0], -B[i][2]};
            H[i * 2 + 1] = new int[]{B[i][1], B[i][2]};
        }
        Arrays.sort(H, (a, b) -> a[0] != b[0] ? a[0] - b[0] : a[1] - b[1]);
        var map = new TreeMap<Integer, Integer>(Comparator.reverseOrder());
        map.put(0, 1);
        List<List<Integer>> res = new ArrayList<>();
        int prev = 0;
        for (int[] h : H) {
            if (h[1] < 0) map.put(-h[1], map.getOrDefault(-h[1], 0) + 1);
            else {
                map.put(h[1], map.get(h[1]) - 1);
                if (map.get(h[1]) == 0) map.remove(h[1]);
            }
            if (map.firstKey() != prev) {
                prev = map.firstKey();
                res.add(List.of(h[0], prev));
            }
        }
        return res;
    }
}
```


Testcase
Test Result
Accepted
Runtime: 1 ms
Case 1
Case 2
Input
buildings =
[[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]
Output
[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8]