

## ASSIGNMENT 6

**STUDENT NAME:** Amandeep Sharma

**UID:** 22BCS16228

**BRANCH:** CSE

**SECTION:** 22BCS\_FL\_IOT\_601A

**SEMESTER:** 6

**DATE OF SUBMISSION:** 16/3/25

**SUBJECT NAME:** AP LAB -2

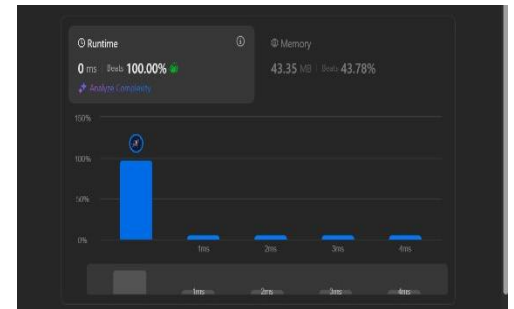
**SUBJECT CODE:** 22CSP-351

### LEET CODE QUESTIONS :

#### 108.CONVERT SORTED ARRAY TO BINARY SEARCH TREE class

Solution {

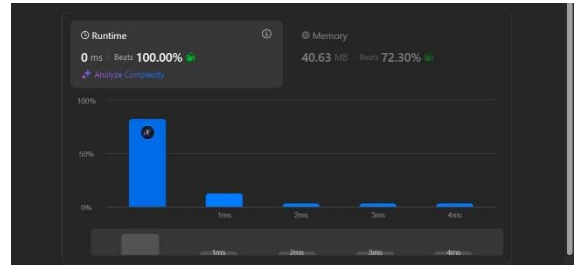
```
public TreeNode sortedArrayToBST(int[] nums) {  
    return helper(nums, 0, nums.length - 1);  
}  
private TreeNode helper(int[] nums, int left, int right) {  
    if (left > right) return null;  
    int mid = left + (right - left) / 2;  
    TreeNode node = new TreeNode(nums[mid]);  
    node.left = helper(nums, left, mid - 1);  
    node.right = helper(nums, mid + 1, right);  
    return node;  
}  
}
```



#### 191.NUMBER OF 1 BITS

class Solution {

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

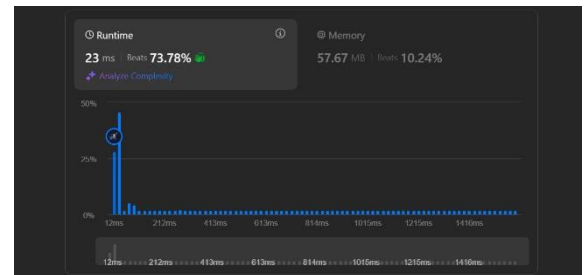


## 912. [SORT AN ARRAY](#)

```
class Solution {
    public int[] sortArray(int[] nums) {
        mergeSort(nums, 0, nums.length - 1);
        return nums;
    }

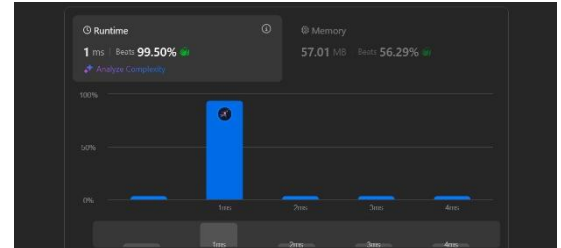
    private void mergeSort(int[] nums, int left, int right)
    {
        if (left >= right) return;
        int mid = left + (right - left) / 2;
        mergeSort(nums, left, mid);
        mergeSort(nums, mid + 1, right);
        merge(nums, left, mid, right);
    }

    private void merge(int[] nums, int left, int mid, int right) {
        int[] temp = new int[right - left + 1];
        int i = left, j = mid + 1, k = 0;
        while (i <= mid && j <= right) {
            if (nums[i] <= nums[j]) temp[k++] = nums[i++];
            else temp[k++] = nums[j++];
        }
        while (i <= mid) temp[k++] = nums[i++];
        while (j <= right) temp[k++] = nums[j++];
        System.arraycopy(temp, 0, nums, left, temp.length);
    }
}
```



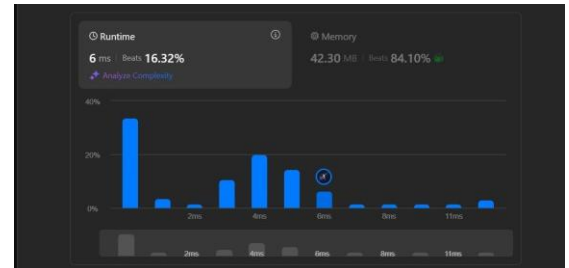
### 53. MAXIMUM SUBARRAY

```
class Solution {
    public int maxSubArray(int[] nums)
    {
        int maxSum = nums[0], currSum = nums[0];
        for (int i = 1; i < nums.length; i++) {
            currSum = Math.max(nums[i], currSum + nums[i]);
            maxSum = Math.max(maxSum, currSum);
        }
        return maxSum;
    }
}
```



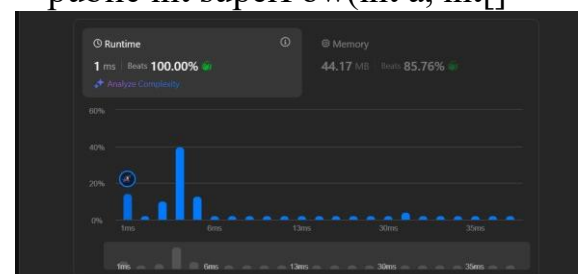
### 932. BEAUTIFUL ARRAY

```
class Solution {
    public int[] beautifulArray(int n) {
        List<Integer> res = new ArrayList<>();
        res.add(1);
        while (res.size() < n) {
            List<Integer> temp = new ArrayList<>();
            for (int num : res) if (num * 2 - 1 <= n) temp.add(num * 2 - 1);
            for (int num : res) if (num * 2 <= n) temp.add(num * 2);
            res = temp;
        }
        return res.stream().mapToInt(i -> i).toArray();
    }
}
```



### 372. SUPER POW

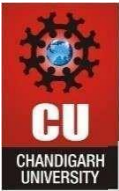
```
class Solution {
    private static final int MOD = 1337;
    public int superPow(int a, int[] b) {
        return modPow(a, arrayToInt(b), MOD);
    }
    private int arrayToInt(int[] b) {
        int num = 0;
        for (int digit : b) num = (num * 10 + digit) % 1140;
        return num == 0 ? 1140 : num;
    }
}
```



```
    }  
    private int modPow(int a, int b, int mod) {  
a %= mod;    int res = 1;    while (b >  
0) {  
    if ((b & 1) == 1) res = (res * a) % mod;  
a = (a * a) % mod;  
    b >>= 1;  
    }  
return res;  
    }  
}
```

## **218. THE SKYLINE PROBLEM**

```
class Solution {  
    public List<List<Integer>> getSkyline(int[][] buildings) {  
        List<int[]> events = new ArrayList<>();  
for (int[] b : buildings) {  
events.add(new int[] {b[0], -b[2]});  
        events.add(new int[] {b[1], b[2]});  
    }  
    Collections.sort(events, (a, b) -> a[0] == b[0] ? Integer.compare(a[1], b[1]) :  
Integer.compare(a[0], b[0]));  
    List<List<Integer>> res = new ArrayList<>();  
    TreeMap<Integer, Integer> heightMap = new  
TreeMap<>(Collections.reverseOrder());  
    heightMap.put(0, 1);  
int prevHeight = 0;  
for (int[] e : events) {  
    if (e[1] < 0) heightMap.put(-e[1], heightMap.getOrDefault(-e[1], 0) + 1);  
else {  
        if (heightMap.get(e[1]) == 1) heightMap.remove(e[1]);  
else heightMap.put(e[1], heightMap.get(e[1]) - 1);  
    }  
    int currHeight = heightMap.firstKey();  
if (currHeight != prevHeight) {
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
res.add(Arrays.asList(e[0], currHeight));  
prevHeight = currHeight;  
    }  
    return res;  
  
}  
}
```

