

DAY-5

Name:Hritik Ranjan Rai
Branch:BE-CSE

UID:22BCS13655
SEC-615-IOT

1. Median of Two Sorted Arrays.

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

The overall run time complexity should be $O(\log(m+n))$.

Example 1:

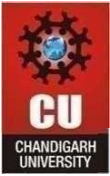
Input: nums1 = [1,3], nums2 = [2]

Output: 2.00000

Explanation: merged array = [1,2,3] and median is 2. Answer:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits> using
namespace std;

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
    if (nums1.size() > nums2.size()) {        return findMedianSortedArrays(nums2,
        nums1); // Ensure nums1 is smaller
    }
    int m = nums1.size(), n = nums2.size();    int low = 0, high = m;    while (low
    <= high) {        int partition1 = (low + high) / 2;        int partition2 = (m + n + 1)
    / 2 - partition1;        int maxLeft1 = (partition1 == 0) ? INT_MIN :
    nums1[partition1 - 1];        int minRight1 = (partition1 == m) ? INT_MAX :
    nums1[partition1];        int maxLeft2 = (partition2 == 0) ? INT_MIN :
    nums2[partition2 - 1];        int minRight2 = (partition2 == n) ? INT_MAX :
    nums2[partition2];        if (maxLeft1 <= minRight2 && maxLeft2 <= minRight1) {
    if ((m + n) % 2 == 0) {            return (max(maxLeft1, maxLeft2) +
    min(minRight1, minRight2)) / 2.0;
        } else {
```



Name: Vatsala Singh

1
UID:22BCS10028

return max(maxLeft1,

maxLeft2);

}

```

    } else if (maxLeft1 > minRight2) {    high
= partition1 - 1;

```

```

    } else {    low = partition1
+ 1;

```

}

}

```

    throw invalid_argument("Input arrays are not sorted");

```

```

} int main()

```

```

{    vector<int> nums1 = {1, 3};    vector<int> nums2 = {2};    cout
<<

```

```

"Median: " << findMedianSortedArrays(nums1, nums2) << endl;    return
0;

```

}

2.Kth Smallest Product of Two Sorted Arrays.

Given two sorted 0-indexed integer arrays `nums1` and `nums2` as well as an integer `k`, return the `k`th (1-based) smallest product of `nums1[i] * nums2[j]` where $0 \leq i < \text{nums1.length}$ and $0 \leq j < \text{nums2.length}$.

Example 1:

Input: `nums1 = [2,5]`, `nums2 = [3,4]`, `k = 2`

Output: 8

Answer:

```

#include <iostream>

```

```

#include <vector>

```

```

#include <queue>

```

```

using namespace std;

```

```

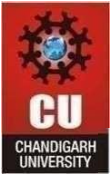
int kthSmallestProduct(vector<int>& nums1, vector<int>& nums2, int k) {
int m = nums1.size(), n = nums2.size();    priority_queue<long long> pq;

```

```

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

```



```

        = (long long)nums1[i] * nums2[j];
        pq.push(product);
        } else if (pq.top() > product) {
        pq.pop();          pq.push(product);
        }
    }
}

return pq.top();
}

int main() {
    vector<int> nums1 = {2, 5};    vector<int> nums2 = {3, 4};    int k = 2;    cout
<<
    "Kth smallest product: " << kthSmallestProduct(nums1, nums2, k) << endl;    return
    0;
}

```

3.Sorted GCD Pair Queries.

You are given an integer array nums of length n and an integer array queries.

Let gcdPairs denote an array obtained by calculating the GCD of all possible pairs (nums[i], nums[j]), where $0 \leq i < j < n$, and then sorting these values in ascending order.

For each query queries[i], you need to find the element at index queries[i] in gcdPairs.

Return an integer array answer, where answer[i] is the value at gcdPairs[queries[i]] for each query.

The term gcd(a, b) denotes the greatest common divisor of a and b.

Example 1:

Input: nums = [2,3,4], queries = [0,2,2] Output:

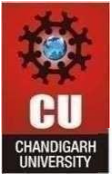
[1,2,2] Answer:

```

#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>

```

```
using namespace std;
```



```
vector<int> gcdPairsQuery(vector<int>& nums, vector<int>& queries) {  
    vector<int> gcdPairs;    int n = nums.size();    for (int i = 0; i < n; ++i) {  
        for (int j = i + 1; j < n; ++j) {            gcdPairs.push_back(gcd(nums[i],  
nums[j]));  
        }  
    }  
}
```

```
sort(gcdPairs.begin(), gcdPairs.end());
```

```
    vector<int> result;    for (int q :  
queries) {  
        result.push_back(gcdPairs[q]);  
    }
```

```
    return result;  
}
```

```
int main() {
```

```
    vector<int> nums = {2, 3, 4};  
    vector<int> queries = {0, 2, 2};  
    vector<int> result = gcdPairsQuery(nums, queries);
```

```
    cout << "Query results: ";  
    for (int res : result) {        cout  
<< res << " ";  
    }    cout <<  
endl;  
    return 0;  
}
```

4.Binary Tree - Find Maximum Depth

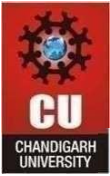
A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1:

Input: [3,9,20,null,null,15,7] Output:

3 Answer:

```
#include <iostream>
```



```
#include <queue>
```

```
using namespace std;
```

```
struct TreeNode {  
    int val;  
    TreeNode* left;  
    TreeNode* right;  
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}  
};
```

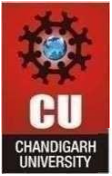
```
int maxDepth(TreeNode* root) {  
    if (!root) return 0;
```

```
    queue<TreeNode*> q;  
    q.push(root);  
    int depth = 0;
```

```
    while (!q.empty()) {        int  
        size = q.size();        for (int i = 0; i  
        < size; ++i) {            TreeNode*  
        node = q.front();        q.pop();  
            if (node->left) q.push(node->left);  
        if (node->right) q.push(node->right);  
        }  
        ++depth;  
    }
```

```
    return depth;  
}
```

```
int main() {  
    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);
```



```
cout << "Maximum Depth: " << maxDepth(root) << endl;
return 0;
}
```

5.Lowest Common Ancestor of a Binary Tree

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree. The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself).

Example 1:

Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1 Output:
3

Explanation: The LCA of nodes 5 and 1 is 3 Answer:

```
#include <iostream>
```

```
#include <unordered_map>
```

```
using namespace std;
```

```
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
```

```
TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
    if (!root || root == p || root == q) return root;
```

```
    TreeNode* left = lowestCommonAncestor(root->left, p, q);
    TreeNode* right = lowestCommonAncestor(root->right, p, q);
```

```
    if (left && right) return root;
    return left ? left : right;
}
```

```
int main() {
```

```
TreeNode* root = new TreeNode(3);  root-
>left = new TreeNode(5);  root->right =
new TreeNode(1);  root->left->left = new
TreeNode(6);  root->left->right = new
TreeNode(2);  root->right->left = new
TreeNode(0);

root->right->right = new TreeNode(8);
```

```
TreeNode* p = root->left; // Node 5
TreeNode* q = root->right; // Node 1
```

```
cout << "LCA: " << lowestCommonAncestor(root, p, q)->val << endl;
return 0;
}
```

6.Binary Tree Maximum Path Sum

A path in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence at most once. Note that the path does not need to pass through the root.

The path sum of a path is the sum of the node's values in the path.

Given the root of a binary tree, return the maximum path sum of any non-empty path.

Example 1: Input: root = [1,2,3] Output: 6 Answer:

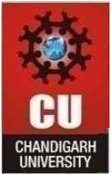
```
#include <iostream>
#include <algorithm>
```

```
using namespace std;
```

```
struct TreeNode {
    int val;
    TreeNode* left;

    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
```

```
int maxPathSum(TreeNode* root, int& result) {
    if (!root) return 0;
```



```

        int left = max(0, maxPathSum(root->
left, result));
        int right = max(0, maxPathSum(root->right, result));

        result = max(result, root->val + left + right);

        return root->val + max(left, right); }

int main() {
    TreeNode* root = new TreeNode(-10);
    root->left = new TreeNode(9);    root->right
= new TreeNode(20);    root->right->left =
new TreeNode(15);
    root->right->right = new TreeNode(7);

    int result = INT_MIN;
    maxPathSum(root, result);

    cout << "Maximum Path Sum: " << result << endl;
    return 0;
}

```

.Count Paths That Can Form a Palindrome in a Tree

You are given a tree (i.e. a connected, undirected graph that has no cycles) rooted at node 0 consisting of n nodes numbered from 0 to $n - 1$. The tree is represented by a 0-indexed array `parent` of size n , where `parent[i]` is the parent of node i . Since node 0 is the root, `parent[0] == -1`.

You are also given a string `s` of length n , where `s[i]` is the character assigned to the edge between i and `parent[i]`. `s[0]` can be ignored.

Return the number of pairs of nodes (u, v) such that $u < v$ and the characters assigned to edges on the path from u to v can be rearranged to form a palindrome.

A string is a palindrome when it reads the same backwards as forwards.

Example 1:

Input: `parent = [-1,0,0,1,1,2]`, `s = "acaabc"` Output:

8 Answer:

```

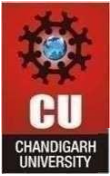
#include <iostream>
#include <vector>
#include <unordered_map>

```

```

using namespace std;

```

```
int countPalindromePaths(int n, vector<int>& parent, string& s) {
    vector<int> count(n, 0);    int res = 0;

    function<void(int)> dfs = [&](int node) {
        count[node] ^= (1 << (s[node] - 'a'));    for
        (int child : adj[node]) {        dfs(child);
    }
};

    for (int i = 1; i < n; ++i) {
        adj[parent[i]].push_back(i);
    }

    dfs(0);
    return res;
}

int main() {    vector<int> parent = {-1, 0, 0, 1, 1, 2};    string s =
"acaabc";    cout <<
"Palindrome Path Count: " << countPalindromePaths(6, parent, s) << endl;    return
0;
}
```