First name: Hima Sai Kiran
Last name: Prudhivi
NetID: HXP220011
Section: 001

First name: Anthea
Last name: Abreo
NetID: AXA210122
Section: 001

# CS 5343: Assignment 2

---

## Question 1: Majority Element

Given an array nums of size n, return the majority element.

The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Example 1:

Input: nums = [3,2,3]
Output: 3

Example 2:
Input: nums = [2,2,1,1,1,2,2]
Output: 2

Constraints:
n == nums.length
1 <= n <= 5 * 104
-109 <= nums[i] <= 109

## Code

```
/* Topic: Arrays
 * Question: Majority Element * Given an array nums of size n, return the majority element. * The majority element is the element that appears more than ⌊n / 2⌋ times. * You may assume that the majority element always exists in the array. */
#include <iostream>
#include<vector>

using namespace std;
```

```cpp
int majorityElement(vector<int>& nums) {

    int ME;
    int me_count = 1;
    int n = nums.size();

    ME = nums[0];


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

        if(me_count == 0){
            ME = nums[i];
            me_count = 1;
        }
        else{
            if(nums[i] == ME){
                me_count++;
            }
            else{
                me_count--;
            }
        }

    }

    return ME;
}

int main() {
    vector<int> arr = { 1, 1, 2, 1, 3, 5, 1 };

    // Function calling
    cout<<"The majority element is "<<majorityElement(arr);
    return 0;
}
```

## Output

---

## Question 2: Longest Consecutive Sequence

Given an unsorted array of integers nums, return the length of the longest consecutive elements sequence.

You must write an algorithm that runs in O(n) time.

Example 1:

Input: nums = [100,4,200,1,3,2]
Output: 4

Explanation:
The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its length is 4.

# Code

```
/* Topic: Hashing
 * Question: Longest Consecutive Sequence * Given an unsorted array of integers
nums, return the length of the longest consecutive elements sequence. * You must
write an algorithm that runs in O(n) time. */
#include <iostream>
#include <vector>
#include <unordered_set>

using namespace std;

int longestConsecutive(vector<int>& nums) {

    int maxcount = 0;
    int ele;

    if(nums.size() == 0){
        return 0;
    }

    int count = 0;
    unordered_set<int> hset;

    //Insert elements in hash set
    for(int i = 0; i < nums.size(); i++){
        hset.insert(nums[i]);
    }

    //Check for consecutive sequence
    for(int i = 0; i < nums.size(); i++){

        ele = nums[i];

        if(hset.find(ele - 1) == hset.end()){
```

```
            count = 1;

            while(hset.find(ele + 1) != hset.end()){
                count++;
                ele++;
            }

        }

        maxcount = max(maxcount, count);
    }

    return maxcount;
}

int main()
{
    vector<int> arr = { 1, 9, 3, 10, 4, 20, 2, 5, 11 };
    cout << "Length of the Longest contiguous subsequence "
            "is "         << longestConsecutive(arr);
    return 0;
}
```

## Output

---

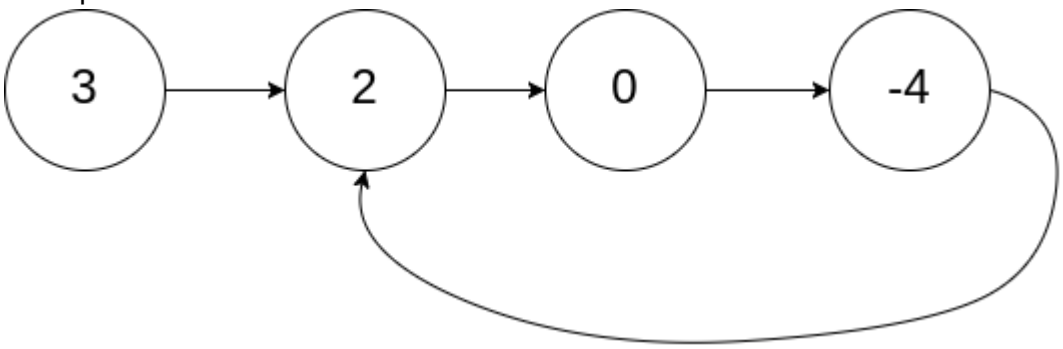
## Question 3: Linked List Cycle

Given head, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. Note that pos is not passed as a parameter.

Return true if there is a cycle in the linked list. Otherwise, return false.

Example:




Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

## Code

```
/* Topic: Linked Lists
 * Question: Linked List Cycle * Given head, the head of a linked list, determine if the linked list has a cycle in it. * Return true if there is a cycle in the linked list. Otherwise, return false. */#include <iostream>
#include <vector>

using namespace std;

struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(NULL) {}
};

void push(struct ListNode** head_ref, int new_data)
{
    /* allocate node */
    struct ListNode* new_node = new ListNode(new_data);

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

bool hasCycle(ListNode *head) {

    ListNode* fast;
```

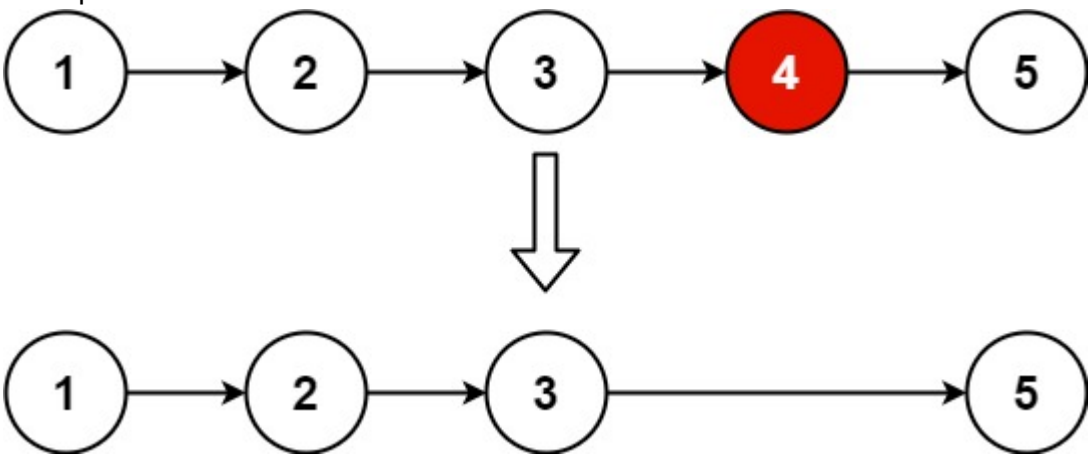
```
    ListNode* slow;

    fast = head;
    slow = head;

    while(fast != NULL && fast -> next != NULL){
        slow = slow -> next; //Move slow pointer by 1
        fast = fast -> next -> next; //Move fast pointer by 2

        if(slow == fast){
            return true;
        }

    }

    return false;
}

int main(){
    ListNode* head = new ListNode(4);

    push(&head, 20);
    push(&head, 5);
    push(&head, 15);
    push(&head, 10);

    head->next->next->next->next = head;


    if(hasCycle(head))
        cout<<"Loop found"<<endl;

    else
        cout<<"No Loop"<<endl;

    return 0;

}
```

## Output

---

## Question 4: Remove Nth Node From End of List

Given the head of a linked list, remove the nth node from the end of the list and return its head.
Example:




Input: head = [1,2,3,4,5], n = 2
Output: [1,2,3,5]

## Code

```cpp
/* Topic: Linked Lists
 * Question: Remove Nth Node From End of List * Given the head of a linked list, *
remove the nth node from the end of the list and return its head. */
#include <iostream>
#include <vector>

using namespace std;

struct ListNode {
    int val;
    ListNode *next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode *next) : val(x), next(next) {}
};

ListNode* removeNthFromEnd(ListNode* head, int n) {

    ListNode* slow = head;
    ListNode* fast = head;
    ListNode* dummy = new ListNode(0, head);

    if(head -> next == NULL){
        return NULL;
    }
```

```
    while(n > 0 && fast != NULL){
        fast = fast -> next;
        n--;
    }

    //If we reached null, it means n = number of nodes in the list, so delete
//first node
    if(fast == NULL){
        head = head -> next;
        return head;
    }

    //move the two pointers till we reach the end of the list
    while(fast->next != NULL){
        fast = fast -> next;
        slow = slow -> next;
    }

    slow -> next = slow -> next -> next;

    return head;

}

int main(){

    return 0;
}
```

## Output

---

## Question 5: Kth Largest Element in a Stream

Design a class to find the kth largest element in a stream. Note that it is the kth largest element in the sorted order, not the kth distinct element.

Implement KthLargest class:

KthLargest(int k, int[] nums) Initializes the object with the integer k and the stream of integers nums.
int add(int val) Appends the integer val to the stream and returns the element representing the kth largest element in the stream.

["KthLargest", "add", "add", "add", "add", "add"]
[3, [4, 5, 8, 2], [3], [5], [10], [9], [4]]
Output
[null, 4, 5, 5, 8, 8]

# Code

```cpp
/* Topic: Heaps/Priority Queues
 * Question: Kth Largest Element in a Stream * Design a class to find the kth
largest element in a stream. * Note that it is the kth largest element in the
sorted order, not the kth distinct element. */
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

class KthLargest {
public:
    KthLargest(int k, vector<int>& nums) {
        this -> k = k;

        for(int i = 0; i < nums.size(); i++){
            pq.push(nums[i]);

        }
        while(pq.size() > this -> k){
            pq.pop();
        }
    }

    int add(int val) {
        pq.push(val);

        if(pq.size() > k){
            pq.pop();
        }
        return pq.top();
    }
private:
    int k;
    priority_queue<int, vector<int>, greater<int>> pq;
};

int main(){
```

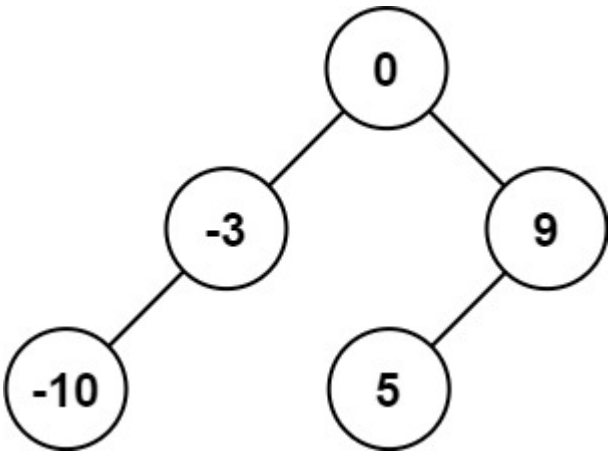
```
    int k = 3;
    vector<int> nums = {4, 5, 8, 2};
    KthLargest* obj = new KthLargest(k, nums);

    int param_1 = obj->add(10);
    cout<<"Kth Largest Value after adding 10: "<<param_1<<endl;
    int param_2 = obj->add(15);
    cout<<"Kth Largest Value after adding 15: "<<param_2<<endl;

    return 0;

}
```

## Output

---

## Question 6: Convert Sorted Array into 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.




Input: nums = [-10,-3,0,5,9]
Output: [0,-3,9,-10,null,5]
Explanation: [0,-10,5,null,-3,null,9] is also accepted:

## Code

```
/* Topic: Search Trees
 * Question: Convert Sorted Array into Binary Search Tree * Design a class to find the kth largest element in a stream. * Note that it is the kth largest element in the sorted order, not the kth distinct element. */
#include <iostream>
```

```cpp
#include <vector>

using namespace std;

struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
};

TreeNode* buildBST(vector<int>& A, int l, int r){

    if(l > r){
        return NULL;
    }
    int mid = r -(r - l)/2;

    TreeNode* root = new TreeNode(A[mid]);

    root->left = buildBST(A, l, mid-1);
    root->right = buildBST(A, mid + 1, r);

    return root;

}

TreeNode* sortedArrayToBST(vector<int>& nums) {

    return buildBST(nums, 0, nums.size()-1);

}

void printPreorder(TreeNode* root){

    if(root == NULL){
        cout<<" null";
        return;
    }

    cout<<" "<<root -> val;

    printPreorder(root->left);
```

```
    printPreorder(root->right);

}

int main() {
    vector<int> nums = {-10, -3, 0, 5, 9};

    TreeNode *mytree = new TreeNode();
    mytree = sortedArrayToBST(nums);
    cout<<"Preorder Traversal:"<<endl;
    printPreorder(mytree);

    return 0;
}
```

## Output

---

## Question 7: Rotate Array

Given an array, rotate the array to the right by k steps, where k is non-negative.

Input: nums = [1,2,3,4,5,6,7], k = 3
Output: [5,6,7,1,2,3,4]
Explanation:
rotate 1 steps to the right: [7,1,2,3,4,5,6]
rotate 2 steps to the right: [6,7,1,2,3,4,5]
rotate 3 steps to the right: [5,6,7,1,2,3,4]

## Code

```
/* Topic: Arrays
 * Question: Rotate Array * Given an array, rotate the array to the right by k
steps, where k is non-negative. */
#include <iostream>
#include <vector>

using namespace std;

void reverse(vector<int>&A, int start, int end){
    int n = A.size();

    int i = start;
    int j = end;
```

```
    while(i <= j){
        swap(A[i], A[j]);
        i++;
        j--;
    }
}

void rotate(vector<int>& nums, int k) {

    int N = nums.size();

    if(k == N){
        return;
    }
    if(k > N){
        k = k % N;
    }
    //Reverse whole array
    reverse(nums, 0, N-1);

    //Reverse first k elements
    reverse(nums, 0, k-1);

    //Reverse remaining
    reverse(nums, k, N-1);

}

void printArr(vector<int> &A){

    for(int i = 0; i < A.size(); i++){
        cout<<A[i]<<" ";
    }
    cout<<endl;
}
int main(){

    vector<int> arr1 = {1,2,3,4,5,6,7};
    vector<int> arr2 = {-1,-100,3,99};

    printArr(arr1);
    rotate(arr1, 3);
    cout<<"{1,2,3,4,5,6,7} rotated by 3: ";
    printArr(arr1);
```

```
    printArr(arr2);
    rotate(arr2, 2);
    cout<<"{-1,-100,3,99} rotated by 2: ";
    printArr(arr2);

    return 0;
}
```

## Output

---

## Question 8: Two Sum

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Example 1:

Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

## Code

```
/* Topic: Hashing
 * Question: Two Sum * Given an array of integers nums and an integer target, *
return indices of the two numbers such that they add up to target. */
#include <iostream>
#include <vector>
#include <unordered_map>

using namespace std;

vector<int> twoSum(vector<int>& nums, int target) {

    unordered_map<int,int> hmap;
    vector<int> result;

    for(int i = 0; i < nums.size(); i++){
```

```
        int look = target - nums[i];

        if(hmap.find(look) != hmap.end()){//if target-nums[i] exists in hashmap
            result.push_back(hmap[look]);
            result.push_back(i);
            break;
        }

        hmap[nums[i]] = i;
    }
    return result;

}

void printArr(vector<int> &A){

    for(int i = 0; i < A.size(); i++){
        cout<<A[i]<<" ";
    }
    cout<<endl;
}

int main() {

    vector<int> arr1 = {2,7,11,15};
    vector<int> arr2 = {3,2,4,5};
    vector<int> res1, res2;

    printArr(arr1);
    cout<<"Two Sum: ";
    res1 = twoSum(arr1,9);
    printArr(res1);

    printArr(arr2);
    cout<<"Two Sum: ";
    res2 =  twoSum(arr2, 6);
    printArr(res2);


    return 0;
}
```

## Output

# Question 9: Insert element in a Sorted List

Given a sorted linked list and a value to insert, write a function to insert the value in a sorted way

# Code

```
/* Topic: Linked Lists
 * Question: Insert in a Sorted List * Given a linked list sorted in ascending order and an integer called data, * insert data in the linked list such that the list remains sorted. */
#include <iostream>
#include <vector>

using namespace std;

class Node {
public:
    int data;
    Node* next;
};

void sortedInsert(Node** head_ref,
                  Node* new_node)
{
    Node* current;
    /* Special case for the head end */
    if (*head_ref == NULL
        || (*head_ref)->data
           >= new_node->data) {
        new_node->next = *head_ref;
        *head_ref = new_node;
    }
    else {
        /* Locate the node before the
 point of insertion */        current = *head_ref;
        while (current->next != NULL
               && current->next->data
                  < new_node->data) {
            current = current->next;
        }
        new_node->next = current->next;
        current->next = new_node;
    }
}
```

```
Node* newNode(int new_data)
{
    /* allocate node */
    Node* new_node = new Node();

    /* put in the data */
    new_node->data = new_data;
    new_node->next = NULL;

    return new_node;
}

void printList(Node* head)
{
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
}

int main()
{
    /* Start with the empty list */
    Node* head = NULL;
    Node* new_node = newNode(5);
    sortedInsert(&head, new_node);
    new_node = newNode(10);
    sortedInsert(&head, new_node);
    new_node = newNode(7);
    sortedInsert(&head, new_node);
    new_node = newNode(3);
    sortedInsert(&head, new_node);
    new_node = newNode(1);
    sortedInsert(&head, new_node);
    new_node = newNode(9);
    sortedInsert(&head, new_node);
    cout << "Created Linked List\n";
    printList(head);

    return 0;
}
```

## Output

## Question 10: Remove Duplicates from Sorted Array

Given an integer array nums sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the first part of the array nums. More formally, if there are k elements after removing the duplicates, then the first k elements of nums should hold the final result. It does not matter what you leave beyond the first k elements.

Return k after placing the final result in the first k slots of nums.

Example 1:

Input: nums = [1,1,2]
Output: 2, nums = [1,2,_]
Explanation: Your function should return k = 2, with the first two elements of nums being 1 and 2 respectively.
It does not matter what you leave beyond the returned k (hence they are underscores).

## Code

```cpp
/* Topic: Arrays
 * Question: Remove Duplicates from Sorted Array * Given an integer array nums sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. * The relative order of the elements should be kept the same. */
#include <iostream>
#include <vector>

using namespace std;

int removeDuplicates(vector<int>& nums) {

    if(nums.size() == 1){
        return 1;
    }
    int i = 0;
    int j = 1;
    int k = 1;
    int n = nums.size();

```

```
    while(i < n && j < n){
        if(nums[i] == nums[j]){
            j++;
        }
        else{
            nums[i+1] = nums[j];
            i++;
            j++;
            k++;
        }
    }

    return k;


}

int main() {
    vector<int> arr = { 1, 1, 2, 2, 3, 5, 5, 6 };

    // Function calling
    cout<<"Number of elements after removing duplicates: "<<removeDuplicates(arr);
    return 0;
}
```

## Output

---

## Question 11: Contains Duplicate

Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

Example 1:

Input: nums = [1,2,3,1]
Output: true

## Code

```
/* Topic: Hashing
 * Question: Contains Duplicate * Given an integer array nums, * return true if any value appears at least twice in the array, * and return false if every element is distinct. */
```

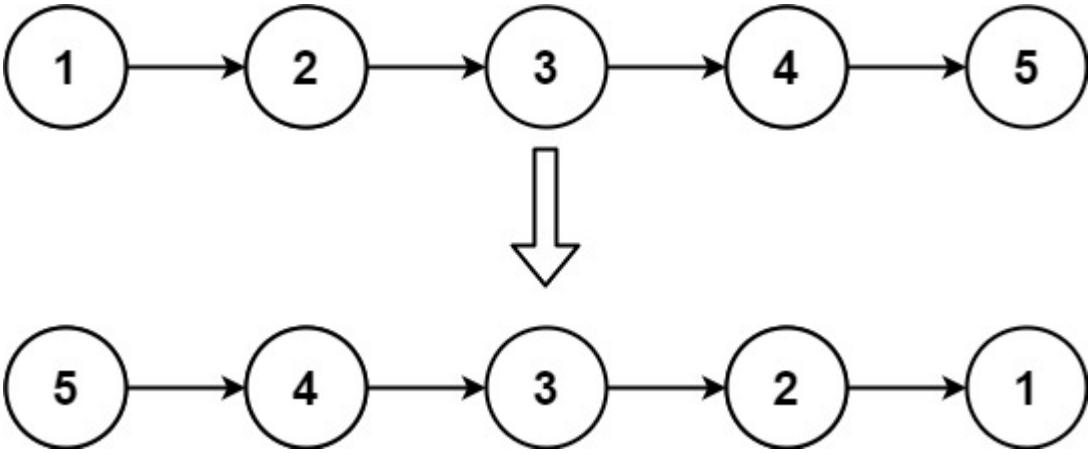
```cpp
#include <iostream>
#include <vector>
#include <unordered_map>

using namespace std;

bool containsDuplicate(vector<int>& nums) {

    unordered_map<int,int> mymap;

    for(int i = 0; i < nums.size(); i++){

        if(mymap.find(nums[i]) != mymap.end()){
            return true;
        }

        mymap[nums[i]]++;

    }

    return false;
}

int main() {
    vector<int> arr = { 1, 1, 2, 2, 3, 5, 5, 6 };

    // Function calling
    (containsDuplicate(arr)? cout<<"Yes" : cout<<"No");
    return 0;
}
```

## Output

---

## Question 12: Reverse a Linked List

Given the head of a singly linked list, reverse the list, and return the reversed list.




## Code

```cpp
/* Topic: Linked List
 * Question: Reverse a Linked List * Given the head of a singly linked list,
reverse the list, and return the reversed list. */
#include <iostream>
#include <vector>

using namespace std;

struct ListNode {
int val;
ListNode *next;
ListNode() : val(0), next(nullptr) {}
ListNode(int x) : val(x), next(nullptr) {}
ListNode(int x, ListNode *next) : val(x), next(next) {}
};

ListNode* reverseList(ListNode* head) {

    if(head == NULL || head->next == NULL){
        return head;
    }

    ListNode* prev = NULL;
    ListNode* temp = head;
    ListNode* curr;

    while(temp != NULL){
        curr = temp;
        temp = curr->next;
        curr->next = prev;
        prev = curr;
    }
```

```
    head = curr;

    return head;
}

void printLL(ListNode* head){
    if(head == NULL){
        cout<<"NULL";
        return;
    }
    ListNode* newnode = head;

    while(newnode!= NULL){
        cout<<newnode->val<<" ";
        newnode = newnode->next;
    }
}

int main(){

    ListNode* node1 = new ListNode(1);
    ListNode* node2 = new ListNode(2);
    ListNode* node3 = new ListNode(3);
    ListNode* node4 = new ListNode(4);
    ListNode* node5 = new ListNode(5);
    ListNode* head = new ListNode(0);

    node5->next = NULL;
    node4->next = node5;
    node3->next = node4;
    node2->next = node3;
    node1->next = node2;
    head->next = node1;

    cout<<"LL before reversal:\n";
    printLL(head);
    ListNode* newhead = reverseList(head);
    cout<<"\nLL after reversal:\n";
    printLL(newhead);

    return 0;

}
```

## Output

## Question 13: Reverse words in a string

Given an input string s, reverse the order of the words.

A word is defined as a sequence of non-space characters. The words in s will be separated by at least one space.

Return a string of the words in reverse order concatenated by a single space.

Input: s = "the sky is blue"
Output: "blue is sky the"

## Code

```
/* Topic: Strings
 * Question: Reverse words in a string * Write a function that reverses a string. *
The input string is given as an array of characters. * You must do this by
modifying the input array in-place with O(1) extra memory. */
#include <iostream>
#include <string>

using namespace std;

string reverseWords(string s) {
    string word = "";
    string res = "";
    for (char i: s) {
        if (i == ' ') {
            res = word + " " + res;
            word = "";
        } else {
            word += i;
        }
    }
    res = word + " " + res;
    return res.substr(0, res.size() - 1);
}

int main(){
    string s1 = "the sky is blue";

    string res1 = reverseWords(s1);

    cout<<"Reverse of "<<s1<<":"<<res1;
```

```
    return 0;
}
```

## Output

---

## Question 14: Valid Anagram

Given two strings s and t, return true if t is an anagram of s, and false otherwise.

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Input: s = "anagram", t = "nagaram"
Output: true

## Code

```
/* Topic: Strings
 * Question: Valid Anagram * Given two strings s and t, return true if t is an anagram of s, and false otherwise. */#include <iostream>
#include <string>
#include <vector>

using namespace std;

bool isAnagram(string s, string t) {
    if (s.size() != t.size()) {
        return false;
    }

    vector<int> count(26);

    for (int i = 0; i < s.size(); i++) {
        count[s[i] - 'a']++;
    }

    for (int j = 0; j < t.size(); j++) {
        count[t[j] - 'a']--;
        if (count[t[j] - 'a'] < 0) {
            return false;
        }
    }
```

```
    return true;

}

int main(){
    string s1 = "nagaram";
    string s2 = "anagram";



    (isAnagram(s1,s2))? cout<<"Yes" : cout<<"No";

    return 0;
}
```

## Output

---

## Question 15: Remove Duplicate Letters

Given a string s, remove duplicate letters so that every letter appears once and only once. You must make sure your result is the smallest in lexicographical order among all possible results.

Example 1:

Input: s = "bcabc"
Output: "abc"
Example 2:

Input: s = "cbacdcbc"
Output: "acdb"

## Code

```
/* Topic: Stacks
 * Question: Remove Duplicate Letters * Given a string s, remove duplicate letters
so that every letter appears once and only once. * You must make sure your result
is the smallest in lexicographical order among all possible results */#include
<iostream>
#include <string>
#include <vector>
#include <stack>

using namespace std;
```

```
string removeDuplicateLetters(string s) {

    vector<int> lastidx(26,0);
    vector<bool> charseen(26,false);
    stack<char> st;
    string res = "";

    for(int i = 0; i < s.length(); i++){
        lastidx[s[i] - 'a'] = i;
    }

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

        int curr = s[i] - 'a';

        if(charseen[curr]){
            continue;
        }

        while(st.size()!=0 && st.top() > s[i] && i < lastidx[st.top() - 'a']){
            charseen[st.top() - 'a'] = false;
            st.pop();
        }

        st.push(s[i]);
        charseen[curr] = true;

    }

    while(!st.empty()){
        res += st.top();
        st.pop();
    }

    reverse(res.begin(), res.end());

    return res;


}

int main(){
    string s1 = "cbacdcbc";

    cout<<removeDuplicateLetters(s1);
```

```
    return 0;
}
```

## Output

---

## Question 16: Pow(x, n)

Implement pow(x, n), which calculates x raised to the power n (i.e., xn).

Example 1:

Input: x = 2.00000, n = 10
Output: 1024.00000
Example 2:

Input: x = 2.10000, n = 3
Output: 9.26100

## Code

```
/* Topic: Recursion
 * Question: Pow(x, n) * Implement pow(x, n), which calculates x raised to the
power n (i.e., xn). */
#include <iostream>
#include <cmath>

using namespace std;

double myPow(double x, int n) {

    if(n < 0) {
        x = 1 / x;
    }

    long num = labs(n);

    double pow = 1;

    while(num){ // equivalent to while(num != 0)
        if(num & 1) { // equivalent to if((num & 1) != 0)
            pow *= x;
        }
```

```
        x *= x;
        num >>= 1;
    }

    return pow;
}

int main(){
    int x = 2.00000;
    int n = 10;

    cout<<myPow(x, n);
}
```

## Output

---

## Question 17: Implement Stack Using Queue

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).

## Code

```
/* Topic: Queues
 * Question: Implement Stack Using Queues * Implement a last-in-first-out (LIFO) stack using only two queues. * The implemented stack should support all the functions of a normal stack (push, top, pop, and empty). */
#include <iostream>
#include <queue>

using namespace std;

class Stack {
    queue<int> q;
public:
    void push(int data);
    void pop();
    int top();
    int size();
    bool empty();
};
```

```
void Stack::push(int data)
{
    //  Get previous size of queue
    int s = q.size();

    // Push the current element
    q.push(data);

    // Pop all the previous elements and put them after
    // current element
    for (int i = 0; i < s; i++) {
        // Add the front element again
        q.push(q.front());

        // Delete front element
        q.pop();
    }
}

void Stack::pop()
{
    if (q.empty())
        cout << "No elements\n";
    else
        q.pop();
}

// Returns top of stack
int Stack::top() {
    return (q.empty()) ? -1 : q.front();
}

// Returns true if Stack is empty else false
bool Stack::empty() {
    return (q.empty());
}

int Stack::size(){
    return q.size();
}

int main()
{
    Stack st;
    st.push(1);
    st.push(2);
```

```
    st.push(3);
    cout << "current size: " << st.size() << "\n";
    cout << "Pop "<<st.top() << "\n";
    st.pop();
    cout << "Pop "<<st.top() << "\n";
    st.pop();
    cout << "Pop "<<st.top() << "\n";
    cout << "current size: " << st.size();
    return 0;
}
```

## Output

---

## Question 18: Insertion Sort List

Given the head of a singly linked list, sort the list using insertion sort, and return the sorted list's head.

## Code

```
/* Topic: Sorting Techniques
 * Question: Insertion Sort List * Given the head of a singly linked list, * sort
the list using insertion sort, and return the sorted list's head. */
#include <iostream>
#include <vector>

using namespace std;

struct Node {
    int val;
    struct Node* next;
    Node(int x)
    {
        val = x;
        next = NULL;
    }
};

class LinkedlistIS {

public:
    Node* head;
    Node* sorted;
```

```
void push(int val)
{

    Node* newnode = new Node(val);
    newnode->next = head;
    head = newnode;
}

void insertionSort(Node* headref)
{
    sorted = NULL;
    Node* current = headref;

    while (current != NULL) {

        Node* next = current->next;
        sortedInsert(current);
        current = next;
    }
    head = sorted;
}

void sortedInsert(Node* newnode)
{
    if (sorted == NULL || sorted->val >= newnode->val) {
        newnode->next = sorted;
        sorted = newnode;
    }
    else {
        Node* current = sorted;
        while (current->next != NULL
               && current->next->val < newnode->val) {
            current = current->next;
        }
        newnode->next = current->next;
        current->next = newnode;
    }
}
void printlist(Node* head)
{
    while (head != NULL) {
        cout << head->val << " ";
        head = head->next;
    }
}
```

```
};

int main()
{
    LinkedlistIS list;
    list.head = NULL;
    list.push(5);
    list.push(20);
    list.push(4);
    list.push(3);
    list.push(30);
    cout << "Before Sorting:" << endl;
    list.printlist(list.head);
    cout << endl;
    list.insertionSort(list.head);
    cout << "After Sorting:" << endl;
    list.printlist(list.head);

    return 0;
}
```

## Output

---

## Question 19: 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.

## Code

```
/* Topic: Heap/Priority Queues
 * Question: 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. */
#include <iostream>
#include <vector>

using namespace std;

void heapify(vector<int>&nums,int n,int i){
    int l=2*i+1;
```

```
    int r=2*i+2;
    int lar=i;

    if(l<n && nums[l]>nums[lar])
        lar=l;
    if(r<n && nums[r]>nums[lar])
        lar=r;

    if(lar!=i){
        swap(nums[lar],nums[i]);
        heapify(nums,n,lar);
    }
}

void buildheap(vector<int>&nums,int n){
    for(int i=(n-2)/2;i>=0;i--){
        heapify(nums,n,i);
    }
}

void heapsort(vector<int>&nums,int n){
    buildheap(nums,n);

    for(int i=n-1;i>=0;i--){
        swap(nums[i],nums[0]);
        heapify(nums,i,0);
    }
}

vector<int> sortArray(vector<int>& nums) {
    heapsort(nums,nums.size());
    return nums;
}

void printarr(vector<int>&A){
    for(int i = 0; i < A.size(); i++){
        cout<<A[i]<<" ";
    }
    cout<<endl;
}
int main(){
    vector<int> nums = {5,1,1,2,0,0,7};
    vector<int> res;

    cout<<"Before:";
    printarr(nums);
```

```
    res = sortArray(nums);
    cout<<"After:";
    printarr(res);

    return 0;
}
```

## Output

---

## Question 20:Find if Path Exists in Graph

There is a bi-directional graph with n vertices, where each vertex is labeled from 0 to n - 1 (inclusive). The edges in the graph are represented as a 2D integer array edges, where each edges[i] = [ui, vi] denotes a bi-directional edge between vertex ui and vertex vi. Every vertex pair is connected by at most one edge, and no vertex has an edge to itself.

You want to determine if there is a valid path that exists from vertex source to vertex destination.

Given edges and the integers n, source, and destination, return true if there is a valid path from source to destination, or false otherwise.

## Code

```
/* Topic: Graphs
 * Question: Find if Path Exists in Graph * determine if there is a valid path that exists from vertex source to vertex destination. */
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

bool validPath(int n, vector<vector<int>>& edges, int start, int end) {
    vector<vector<int>> graph(n);
    // Build the graph
    for(int i=0; i<edges.size(); i++) {
        graph[edges[i][0]].push_back(edges[i][1]);
        graph[edges[i][1]].push_back(edges[i][0]);
    }

    // BFS
    queue<int> q;
```

```
    vector<int> visited(n, 0);
    q.push(start);
    visited[start] = 1;
    while(!q.empty()){
        int top = q.front();
        q.pop();
        if(top == end)
            return true;

        for(int i=0; i<graph[top].size(); i++){
            if(visited[graph[top][i]] == 0){
                q.push(graph[top][i]);
                visited[graph[top][i]] = 1;
            }
        }
    }
    return false;
}

int main(){
    return 0;
}
```

## Output

---

## Question 21: Merge Sorted Array

You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

Merge nums1 and nums2 into a single array sorted in non-decreasing order.

The final sorted array should not be returned by the function, but instead be stored inside the array nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

Example 1:

Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
Output: [1,2,2,3,5,6]
Explanation: The arrays we are merging are [1,2,3] and [2,5,6].
The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

# Code

```cpp
/* Topic: Sorting
 * Question: Merge Sorted Array * You are given two integer arrays nums1 and nums2,
sorted in non-decreasing order, * and two integers m and n, representing the number
of elements in nums1 and nums2 respectively. * Merge nums1 and nums2 into a single
array sorted in non-decreasing order. * The final sorted array should not be
returned by the function, * but instead be stored inside the array nums1. * To
accommodate this, nums1 has a length of m + n, * where the first m elements denote
the elements that should be merged, * and the last n elements are set to 0 and
should be ignored. nums2 has a length of n. */
#include <iostream>
#include <vector>

using namespace std;

void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
    int i=m-1,j=n-1,k=m+n-1;
    while(i>=0&&j>=0)
    {
        if(nums1[i]>nums2[j])
        {
            nums1[k--]=nums1[i--];

        }
        else
        {
            nums1[k--]=nums2[j--];

        }
    }

    while(j>=0){
        nums1[k--]=nums2[j--];
    }
}

void printarr(vector<int>&A){
    for(int i = 0; i < A.size(); i++){
        cout<<A[i]<<" ";
    }
    cout<<endl;
}

int main(){
    vector<int>nums1 = {1,2,3,0,0,0};
```

```
    vector<int>nums2 = {2,5,6};

    int m = nums1.size() - nums2.size();
    int n = nums2.size();
    printarr(nums1);
    printarr(nums2);
    merge(nums1, m, nums2, n);
    printarr(nums1);

    return 0;

}
```

## Output

---

## Question 22: Search in Rotated Sorted Array

There is an integer array nums sorted in ascending order (with distinct values).

Prior to being passed to your function, nums is possibly rotated at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (0-indexed). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].

Given the array nums after the possible rotation and an integer target, return the index of target if it is in nums, or -1 if it is not in nums.

You must write an algorithm with O(log n) runtime complexity.

Example 1:

Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4

## Code

```
/* Topic: Searching
 * Question: Search in Rotated Sorted ArrayThere is an integer array nums sorted in
ascending order (with distinct values).
Prior to being passed to your function,
nums is possibly rotated at an unknown pivot index k (1 <= k < nums.length)
such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0],
nums[1], ..., nums[k-1]] (0-indexed).
```

```
For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become
[4,5,6,7,0,1,2].
Given the array nums after the possible rotation and an integer target,
return the index of target if it is in nums, or -1 if it is not in nums.
*/

#include <iostream>
#include <vector>

using namespace std;

int binarySearch(vector<int>&A, int l, int r, int target){
    int mid;

    while(l <= r){
        mid = l + (r-l)/2;

        if(A[mid] == target){
            return mid;
        }
        else if(A[mid] < target){
            l = mid + 1;
        }
        else{
            r = mid - 1;
        }
    }

    return -1;
}

int search(vector<int>& nums, int target) {

    int left = 0;
    int right = nums.size()-1;
    int mid;
    int boundary = -1;

    //Search for the boundary index
    while(left <= right){
        mid = left + (right - left)/2;

        if(nums[mid] > nums[mid + 1]){
            if(mid == 0 || nums[mid] > nums[mid - 1]){
                boundary = mid;
                break;
```

```
            }
            else if(mid !=0 && nums[mid] < nums[mid - 1]){
                boundary = mid - 1;
                break;
            }

        }
        else if((nums[mid] < nums[mid + 1]) && (nums[mid] > nums[mid - 1])){
            if(nums[mid] < nums[right]){
                right = mid - 1;
            }
            else{
                left = mid + 1;
            }
        }
    }

    //Search in the two different arrays
    int res_l = binarySearch(nums, 0, boundary, target);
    int res_r = binarySearch(nums, boundary + 1, nums.size() - 1, target);


    if(res_l == -1)return res_r;
    if(res_r == -1)return res_l;

    return -1;

}

int main(){

    vector<int> nums = {4,5,6,7,0,1,2};
    int target = 0;

    cout<<"Found at index "<<search(nums, target);

    return 0;

}
```

## Output

---

## Question 23: Maximum Depth of Binary Tree

Given the root of a binary tree, return its 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.

## Code

```
/* Topic: Tree
Given the root of a binary tree, return its 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.
 */

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 Solution23 {
    private int depth;

    private void calculateDepth(TreeNode root, int d) {
        if (root == null) {
            return;
        }
        if (d > depth) {
            depth = d;
        }
        calculateDepth(root.left, d + 1);
        calculateDepth(root.right, d + 1);
    }
```

```
    public int maxDepth(TreeNode root) {
        depth = 0;
        calculateDepth(root, 1);
        return depth;
    }
}

public class Q23 {
    public static void main(String[] args) {
        Solution23 solution = new Solution23();
        TreeNode root = new TreeNode(
                3,
                new TreeNode(9),
                new TreeNode(
                        20,
                        new TreeNode(15),
                        new TreeNode(7)
                )
        );
        int depth = solution.maxDepth(root);
        System.out.println("Max depth: " + depth);
    }
}
```

## Output

```
Max depth: 3

Process finished with exit code 0
```

---

## Question 24: Four Sum

Given an array nums of n integers, return an array of all the unique quadruplets [nums[a], nums[b], nums[c], nums[d]] such that:

0 <= a, b, c, d < n
a, b, c, and d are distinct.
nums[a] + nums[b] + nums[c] + nums[d] == target
You may return the answer in any order.

## Code

```java
/* Topic: Arrays
Given an array nums of n integers, return an array of all the unique quadruplets
[nums[a], nums[b], nums[c], nums[d]] such that:

0 <= a, b, c, d < n
a, b, c, and d are distinct.
nums[a] + nums[b] + nums[c] + nums[d] == target
You may return the answer in any order.
 */

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

class Solution24 {
    public List<List<Integer>> fourSum(int[] nums, int target) {
        List<List<Integer>> ans = new ArrayList<>();

        Arrays.sort(nums);

        int len = nums.length;
        for (int j = 0; j < len - 2; j++) {
            if (j == 0 || nums[j] != nums[j - 1]) {
                for (int i = j + 1; i < len - 2; i++) {
                    if (i == j + 1 || nums[i] != nums[i - 1]) {
                        int l = i + 1;
                        int r = len - 1;
                        int sum = -nums[i] - nums[j] + target;
                        while (l < r) {
                            if (nums[l] + nums[r] == sum) {
                                ans.add(Arrays.asList(nums[j], nums[i], nums[l],
nums[r]));
                                while (l < r && nums[l] == nums[l + 1]) {
                                    l += 1;
                                }
                                while (l < r && nums[r] == nums[r - 1]) {
                                    r -= 1;
                                }
                                l += 1;
                                r -= 1;
                            } else if (nums[l] + nums[r] < sum) {
                                l += 1;
                            } else {
                                r -= 1;
                            }
                        }
```

```
                    }
                }
            }
        }

        return ans;
    }
}

public class Q24 {
    public static void main(String[] args) {
        Solution24 solution = new Solution24();
        List<List<Integer>> ans = solution.fourSum(new int[]{1, 0, -1, 0, -2, 2},
0);
        System.out.println(ans);
    }
}
```

## Output

```
Four numbers with sum as target:
[[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]

Process finished with exit code 0
```

---

## Question 25: Longest Substring Without Repeating Characters

Given a string s, find the length of the longest substring without repeating characters.

## Code

```
/* Topic: Hash tables, Strings
Given a string s, find the length of the longest substring without repeating
characters.
 */

class Solution25 {
    public int lengthOfLongestSubstring(String s) {
        int[] lastIndex = new int[256];
        for (int i = 0; i < 256; i++) {
            lastIndex[i] = -1;
```

```
        }

        int i = 0, res = 0;
        for (int j = 0; j < s.length(); j++) {
            i = Math.max(i, lastIndex[s.charAt(j)] + 1);
            res = Math.max(res, j - i + 1);
            lastIndex[s.charAt(j)] = j;
        }

        return res;
    }
}

public class Q25 {
    public static void main(String[] args) {
        Solution25 solution = new Solution25();
        int len = solution.lengthOfLongestSubstring("abcabcbb");
        System.out.println("Length: " + len);
    }
}
```

## Output

```
Length of Longest Substring Without Repeating Characters: 3

Process finished with exit code 0
```

---

## Question 26: Replace Elements with Greatest Element on Right Side

Given an array arr, replace every element in that array with the greatest element among the elements to its right,
and replace the last element with -1.
After doing so, return the array.

## Code

```
/* Topic: Arrays
Given an array arr, replace every element in that array with the greatest element
among the elements to its right,
and replace the last element with -1.
```

```java
After doing so, return the array.
 */

import java.util.Arrays;

class Solution26 {
    public int[] replaceElements(int[] arr) {
        int length = arr.length;
        if (length == 0) {
            return arr;
        } else if (length == 1) {
            arr[0] = -1;
            return arr;
        } else {
            int max = arr[length - 1];
            arr[length - 1] = -1;
            for (int i = length - 2; i >= 0; i--) {
                int a = arr[i];
                arr[i] = max;
                if (a > max) {
                    max = a;
                }
            }
            return arr;
        }
    }
}

public class Q26 {
    public static void main(String[] args) {
        Solution26 solution = new Solution26();
        int[] ans = solution.replaceElements(new int[]{17, 18, 5, 4, 6, 1});
        System.out.println(Arrays.toString(ans));
    }
}
```

## Output

```
Replaced array:
[18, 6, 6, 6, 1, -1]

Process finished with exit code 0
```

---

# Question 27: Longest Palindromic Substring

Given a string s, return the longest palindromic substring in s.

## Code

```java
/* Topic: Dynamic Programming, Strings
Given a string s, return the longest palindromic substring in s.
 */

class Solution27 {
    public String longestPalindrome(String s) {
        int length = s.length();
        if (length == 0 || length == 1) {
            return s;
        }
        int[][] dp = new int[length][length];
        int max = -1, a = 0, b = 0;
        for (int i = 0; i < length; i++) {
            for (int j = 0; j + i < length; j++) {
                int start = j;
                int end = j + i;
                int len = end - start + 1;
                if (len == 1) {
                    dp[start][end] = 1;
                } else {
                    final boolean condition = s.charAt(start) == s.charAt(end);
                    if (len == 2) {
                        dp[start][end] = condition ? 1 : 0;
                    } else {
                        dp[start][end] = condition && dp[start + 1][end - 1] != 0 ?
1 : 0;
                    }
                }

                if (dp[start][end] != 0) {
                    if (len > max) {
                        max = len;
                        a = start;
                        b = end;
                    }
                }
            }
        }
```

```
        return s.substring(a, b + 1);
    }
}

public class Q27 {
    public static void main(String[] args) {
        Solution27 solution = new Solution27();
        String ans = solution.longestPalindrome("babad");
        System.out.println(ans);
    }
}
```

## Output

```
Longest Palindromic Substring: bab

Process finished with exit code 0
```

---

## Question 28: Smallest Subtree with all the Deepest Nodes

Given the root of a binary tree, the depth of each node is the shortest distance to the root.
Return the smallest subtree such that it contains all the deepest nodes in the original tree.
A node is called the deepest if it has the largest depth possible among any node in the entire tree.
The subtree of a node is a tree consisting of that node, plus the set of all descendants of that node.

## Code

```
/* Topic: Tree
Given the root of a binary tree, the depth of each node is the shortest distance to
the root.
Return the smallest subtree such that it contains all the deepest nodes in the
original tree.
A node is called the deepest if it has the largest depth possible among any node in
the entire tree.
The subtree of a node is a tree consisting of that node, plus the set of all
descendants of that node.
 */

class TreeNode28 {
```

```
    int val;
    TreeNode28 left;
    TreeNode28 right;

    TreeNode28() {
    }

    TreeNode28(int val) {
        this.val = val;
    }

    TreeNode28(int val, TreeNode28 left, TreeNode28 right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
}

class Solution28 {
    private int height(TreeNode28 root) {
        if (root == null) {
            return 0;
        }
        return 1 + Math.max(height(root.left), height(root.right));
    }

    private TreeNode28 traverse(TreeNode28 root, int height, int maxHeight) {
        if (height == maxHeight) {
            return root;
        }

        if (root.left != null && root.right != null) {
            TreeNode28 left = traverse(root.left, height + 1, maxHeight);
            TreeNode28 right = traverse(root.right, height + 1, maxHeight);
            if (left != null && right != null) {
                return root;
            } else {
                return left != null ? left : right;
            }
        } else if (root.left != null) {
            return traverse(root.left, height + 1, maxHeight);
        } else if (root.right != null) {
            return traverse(root.right, height + 1, maxHeight);
        } else {
            return null;
        }
```

```
    }

    public TreeNode28 subtreeWithAllDeepest(TreeNode28 root) {
        int height = height(root);
        return traverse(root, 1, height);
    }
}

public class Q28 {
    public static void main(String[] args) {
        Solution28 solution = new Solution28();
        TreeNode28 root = new TreeNode28(
                3,
                new TreeNode28(
                        5,
                        new TreeNode28(6),
                        new TreeNode28(
                                2,
                                new TreeNode28(7),
                                new TreeNode28(4)
                        )
                ),
                new TreeNode28(
                        1,
                        new TreeNode28(0),
                        new TreeNode28(8)
                )
        );
        TreeNode28 ans = solution.subtreeWithAllDeepest(root);
        System.out.println("Smallest Subtree with all the Deepest Nodes has node
value: " + ans.val);
    }
}
```

## Output

```
Smallest Subtree with all the Deepest Nodes has node value: 2

Process finished with exit code 0
```

---

## Question 29: Lowest Common Ancestor of Deepest Leaves

Given the root of a binary tree, return the lowest common ancestor of its deepest leaves.
Recall that:
The node of a binary tree is a leaf if and only if it has no children
The depth of the root of the tree is 0. if the depth of a node is d, the depth of each of its children is d + 1.
The lowest common ancestor of a set S of nodes, is the node A with the largest depth such that every node in S is in the subtree with root A.

## Code

```
/* Topic: Tree
Given the root of a binary tree, return the lowest common ancestor of its deepest
leaves.
Recall that:
The node of a binary tree is a leaf if and only if it has no children
The depth of the root of the tree is 0. if the depth of a node is d, the depth of
each of its children is d + 1.
The lowest common ancestor of a set S of nodes, is the node A with the largest
depth such that every node in S is in the subtree with root A.
 */

class TreeNode29 {
    int val;
    TreeNode29 left;
    TreeNode29 right;

    TreeNode29() {
    }

    TreeNode29(int val) {
        this.val = val;
    }

    TreeNode29(int val, TreeNode29 left, TreeNode29 right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
}

class Solution29 {
    private int height(TreeNode29 root) {
        if (root == null) {
            return 0;
        }
```

```
        return 1 + Math.max(height(root.left), height(root.right));
    }

    private TreeNode29 traverse(TreeNode29 root, int height, int maxHeight) {
        if (height == maxHeight) {
            return root;
        }

        if (root.left != null && root.right != null) {
            TreeNode29 left = traverse(root.left, height + 1, maxHeight);
            TreeNode29 right = traverse(root.right, height + 1, maxHeight);
            if (left != null && right != null) {
                return root;
            } else {
                return left != null ? left : right;
            }
        } else if (root.left != null) {
            return traverse(root.left, height + 1, maxHeight);
        } else if (root.right != null) {
            return traverse(root.right, height + 1, maxHeight);
        } else {
            return null;
        }
    }

    public TreeNode29 lcaDeepestLeaves(TreeNode29 root) {
        int height = height(root);
        return traverse(root, 1, height);
    }
}

public class Q29 {
    public static void main(String[] args) {
        Solution29 solution = new Solution29();
        TreeNode29 root = new TreeNode29(
                3,
                new TreeNode29(
                        5,
                        new TreeNode29(6),
                        new TreeNode29(
                                2,
                                new TreeNode29(7),
                                new TreeNode29(4)
                        )
                ),
                new TreeNode29(
```

```
                        1,
                        new TreeNode29(0),
                        new TreeNode29(8)
                )
        );
        TreeNode29 ans = solution.lcaDeepestLeaves(root);
        System.out.println("Lowest Common Ancestor of Deepest Leaves has node
value: " + ans.val);
    }
}
```

## Output

```
Lowest Common Ancestor of Deepest Leaves has node value: 2

Process finished with exit code 0
```

---

## Question 30: Palindrome Partitioning

Given a string s, partition s such that every substring of the partition is a palindrome. Return all possible palindrome partitioning of s.

## Code

```
/* Topic: Dynamic Programming, String
Given a string s, partition s such that every substring of the partition is a
palindrome. Return all possible palindrome partitioning of s.
 */

import java.util.ArrayList;
import java.util.List;

class Solution30 {
    private ArrayList<ArrayList<Integer>> ansList;
    private List<List<String>> ans;

    private void calc(int index, ArrayList<String> strList, String s) {
        if (index > ansList.size() - 1) {
            ans.add(strList);
            return;
        }
        ArrayList<Integer> a = ansList.get(index);
```

```
        for (Integer right : a) {
            ArrayList<String> sl = new ArrayList<>(strList);
            sl.add(s.substring(index, Math.min(right + 1, ansList.size())));
            calc(right + 1, sl, s);
        }
    }

    public List<List<String>> partition(String s) {
        int len = s.length();
        int[][] dp = new int[len][len];

        ans = new ArrayList<>();
        ansList = new ArrayList<>();
        for (int i = 0; i < len; i++) {
            ansList.add(i, new ArrayList<>());
        }

        for (int i = 0; i < len; i++) {
            for (int j = 0; j + i < len; j++) {
                int left = j;
                int right = j + i;

                if (right - left == 0) {
                    dp[left][right] = 1;
                } else if (right - left == 1) {
                    dp[left][right] = s.charAt(left) == s.charAt(right) ? 1 : 0;
                } else {
                    dp[left][right] = dp[left + 1][right - 1] == 1 ?
(s.charAt(left) == s.charAt(right) ? 1 : 0) : 0;
                }

                if (dp[left][right] == 1) {
                    ansList.get(left).add(right);
                }
            }
        }
        calc(0, new ArrayList<>(), s);
        return ans;
    }
}

public class Q30 {
    public static void main(String[] args) {
        Solution30 solution = new Solution30();
        List<List<String>> ans = solution.partition("aab");
        System.out.println(ans);
```

```
    }
}
```

## Output

```
After Palindrome Partitioning:
[[a, a, b], [aa, b]]

Process finished with exit code 0
```

---