1. Median of Data Streams

Brute:

Sort and find median

Optimal:

➔ Use two heaps (Min and Max)

➔ Size diff should not be more than one

```
➔ void addNum(int num) {
➔         if(maxHeap.size()>0){
➔             if(maxHeap.top()>num)
➔                 maxHeap.push(num);
➔             else
➔                 minHeap.push(num);
➔         }
➔         else{
➔             maxHeap.push(num);
➔         }
➔ 
➔         if(maxHeap.size()>minHeap.size()+1){
➔             minHeap.push(maxHeap.top());
➔             maxHeap.pop();
➔         }
➔         else if(minHeap.size()>maxHeap.size()){
➔             maxHeap.push(minHeap.top());
➔             minHeap.pop();
➔         }
➔     }
```

2. Topological Sort

U->V where U must comes before V

DFS and then push node into stack

```
void toposort(int i,vector<int> &vis,vector<int> adj[],stack<int> &st){
        vis[i] = 1;

        for(int it : adj[i]){
            if(!vis[it])
                toposort(it,vis,adj,st);
        }
        st.push(i);
}
```

3. Least Common Ancestor

- Check base case if (root==null or p==root or r==root) return root;
- Traverse left and right
- If(left is null) return right, right is null return left else return root

```
{
        if(root==NULL) return NULL;
        //2. case
        if(p==root or q==root) return root;

        TreeNode* leftTree = lowestCommonAncestor(root->left,p,q);
        TreeNode* rightTree = lowestCommonAncestor(root->right,p,q);
        // leftTree (val) 5
        // RightTree (val) 1
        if(leftTree==NULL) return rightTree;
        if(rightTree==NULL) return leftTree;

        return root;
}
```

4. Buy and Sell stock maximum

- Buy = day0,
- Check for minimum buying opt. and maximum selling price

```
int maxProfit(vector<int>& prices) {
        int price = prices[0];
        int maxProfit = 0;
        for(int i=1;i<prices.size();i++){
            int profit = prices[i] - price;
            maxProfit = max(maxProfit,profit);
            price = min(price,prices[i]);
        }
        return maxProfit;
    }
```

5. Two Sum

- If sorted, you 2 pointers i=0 and j=n-1, if match return true
- Else if sum>target j—else i++
- If Not sorted, use Maps

```
 vector<int> twoSum(vector<int>& nums, int target) {
        /* for(int i=0;i<nums.size()-1;i++){
                for(int j=i+1;j<nums.size();j++){
                    if(nums[i]+nums[j]==target) return {i,j};
                }
            }
            return {};
            */
            map<int,int> mp;
            for(int i=0;i<nums.size();i++){
                if(mp.find(target-nums[i])!=mp.end()){
                    return {i,mp[target-nums[i]]};
                }
                mp[nums[i]]=i;
            }
            return {};
        }
```

6. Maximum Subarray Sum

Brute: Make every possible subarray and calculate Max.

Best: (Kadane’s Algorithm)

- Add elements to your sum variable and calculate max = max(max,sum)
- If sum<0 then sum=0;

```
    int maxSubArray(vector<int>& nums) {
            int maxSum=INT_MIN,sum=0;
            int n = nums.size();
            for(int i=0;i<n;i++){
                sum+=nums[i];
                maxSum = max(maxSum,sum);
                if(sum<0){
                    sum=0;
                }
            }
            return maxSum;
        }
```

7. LRU Cache

Brute: Use of Map:

1. If size is not full and key not present we insert key
2. If size is not full and key is present, we will remove previous added key and add this key
3. If size is full then, then remove the element which was inserted Last, and we’ll add this element.

Optimize: Use of Doubly Linked List

1. If size is not full and key does not present we insert key at front
2. If size is not full and key is present, we will remove previous added key and add this key to front
3. If size is full then, then remove the element from back

```
  class Node{
        public:
        int key;
        int val;
        Node* next= NULL;
        Node* prev= NULL;
        Node(int _key, int _val){
            key= _key;
            val= _val;
        }
    };

    int maxCap= 0;
    unordered_map<int, Node*> keybucket;
    Node *head= new Node(-1, -1);
    Node *tail= new Node(-1, -1);

    LRUCache(int capacity) {
        maxCap = capacity;
        head->next=tail;
        tail->prev=head;
    }

    void addFrontNode(Node* node){//check me
        Node* prevfront= head->next;
        head->next= node;
        node->prev= head;
        node->next= prevfront;
        prevfront->prev= node;
    }//add into list

    void removeNode(Node* node){//check me
        Node* nodeprev= node->prev;
        Node* nodenext= node->next;
        nodeprev->next= nodenext;
        nodenext->prev= nodeprev;
        node->prev= NULL;
        node->next= NULL;
    }//remove into list

    int get(int key) {
        int val;
        if(keybucket.find(key)!= keybucket.end()){
            //remove node
            removeNode(keybucket[key]);
            //add to front
            addFrontNode(keybucket[key]);
            //store into bucket
            // keybucket[key]=
            val= keybucket[key]->val;
        }
        else{
            return -1;
        }
        return val;
    }

    void put(int key, int value) {
        if(keybucket.find(key)!= keybucket.end()){
            //remove node
            removeNode(keybucket[key]);
            //make new node
            keybucket[key]->val= value;
            //add to front
            addFrontNode(keybucket[key]);
            //store into bucket
            // addFrontNode(keybucket[key]);
        }
        else if(maxCap==keybucket.size()){
            //remove keybucket entry of this key
            keybucket.erase(tail->prev->key);
            //remove last node from list
            removeNode(tail->prev);
            //make new node
            Node* newentry= new Node(key, value);
            //add new node to front
            addFrontNode(newentry);
            //store into bucket
            keybucket[key]= newentry;
        }
        else{
            Node* newentry= new Node(key, value);
            //add new node to front
            addFrontNode(newentry);
            //store into bucket
            keybucket[key]= newentry;
        }
    }
```

8. Find the missing number from 1 to N-1 in the array

Brute:

- ➔ Store the frequency and find whose frequency is 0.
- ➔ Or Sort the array and find the missing number.

Best:

- ➔ Calculate sum of N numbers and calculate sum of array and Return the difference.

```
int missingNumber(vector<int>& nums) {

        int n = nums.size();
        int sumOfn_Numbers = n*(n+1)/2; // 6
        int sumCurrArray=0;
        for(int i=0;i<n;i++){
            sumCurrArray += nums[i];
        }
        return sumOfn_Numbers - sumCurrArray;
    }
};
```

9. Merge K sorted Arrays.

Brute:

1. Store all values of matrix into an array and sort it.

Better:

1. Merge 2 Arrays => return a single array, then merge this array with another array and so on.

Best:

1. Use of Min heap, (value, row, index).
2. Push first elements of all arrays to it.

3. While min heap is not empty:
    a. If index<K
    b. From which row we’ve popped our element, will push element from same row

```
 vector<int> mergeKArrays(vector<vector<int>> arr, int K)
    {
        //code here
        priority_queue<pair<int,pair<int,int>>, 
  vector<pair<int,pair<int,int>>>, greater<pair<int,pair<int,int>>>> pq;
        for(int i=0;i<K;i++){
            pq.push({arr[i][0],{i,0}});
        }
        
        vector<int> ans;
        while(!pq.empty()){
            auto it = pq.top();
            pq.pop();
            ans.push_back(it.first); // value
            int vec = it.second.first;
            int ind = it.second.second;
            ind++;
            if(ind<K){
                pq.push({arr[vec][ind],{vec,ind}});
            }
        }
        return ans;
    }
```

10. WAP to find, all nodes are at K distance of a Binary Tree from node N

- Create a parent pointer for every node.
- Use BFS traversal, and will go in left, right and Up direction only if they are not visited.
- For each traversal we’ll decrement K
- If k==0 break

```
void markParent(TreeNode* root,map<TreeNode*,TreeNode*> &par){
        queue<TreeNode*> q;
        q.push(root);
        while(!q.empty()){
            auto node = q.front();
            q.pop();

```

```cpp
            if(node->left){
                par[node->left] = node;
                q.push(node->left);
            }
            if(node->right){
                par[node->right] = node;
                q.push(node->right);
            }
        }
    }
    vector<int> distanceK(TreeNode* root, TreeNode* target, int k) {
        map<TreeNode*,TreeNode*> par;
        map<TreeNode*,bool> vis;
        markParent(root,par);
        queue<TreeNode*> q;
        q.push(target);
        vis[target]=true;
        while(!q.empty()){
            if(k==0) break;
            k--;
            int n = q.size();
            for(int i=0;i<n;i++){
                auto node = q.front();
                q.pop();
                if(node->left!=NULL and !vis[node->left]){
                    q.push(node->left);
                    vis[node->left]=true;
                }
                 if(node->right!=NULL and !vis[node->right]){
                    q.push(node->right);
                    vis[node->right]=true;
                }
                if( par[node]!=NULL and!vis[par[node]]){
                    q.push(par[node]);
                    vis[par[node]]=true;
                }
            }
        }
        vector<int> ans;
        while(!q.empty()){
            ans.push_back(q.front()->val);
            q.pop();
        }
        return ans;
    }
```

11. Copy Linked List with Random pointer

Step1:

➔ Make copy of each node and link them together side-by-side in a single list.

```
➔  Node* copyRandomList(Node* head) {
➔         Node *iter = head;
➔           Node *front = head;
➔
➔           // First round: make copy of each node,
➔           // and link them together side-by-side in a single list.
➔           while (iter != NULL) {
➔             front = iter->next;
➔
➔             Node *copy = new Node(iter->val);
➔             iter->next = copy;
➔             copy->next = front;
➔
➔             iter = front;
➔           }
```

Step2:

assign random pointers for the copy nodes.

```
    iter = head;
    while (iter != NULL) {
      if (iter->random != NULL) {
        iter->next->random = iter->random->next;
      }
      iter = iter->next->next;
    }
```

Step3: restore the original list, and extract the copy list.

```
   iter = head;
    Node *pseudoHead = new Node(0);
    Node *copy = pseudoHead;

    while (iter != NULL) {
      front = iter->next->next;
      // extract the copy
      copy->next = iter->next;

      // restore the original list
      iter->next = front;

      copy = copy -> next;
      iter = front;
    }

    return pseudoHead->next;
```

12. Print All anagrams pair in the given array

Brute:

One simple idea to find whether all anagram pairs is to run two nested loops. The outer loop picks all strings one by one. The inner loop checks whether remaining strings are anagram of the string picked by outer loop.

Optimize:

Optimizations:

We can optimize the above solution using following approaches.

1) **Using sorting:** We can sort array of strings so that all anagrams come together. Then print all anagrams by linearly traversing the sorted array. The time complexity of this solution is O(mnLogn) (We would be doing O(nLogn) comparisons in sorting and a comparison would take O(m) time)

2) **Using Hashing:** We can build a hash function like XOR or sum of ASCII values of all characters for a string. Using such a hash function, we can build a hash table. While building the hash table, we can check if a value is already hashed. If yes, we can call areAnagrams() to check if two strings are actually anagrams

13.Find the distance between 2 Keys in a Binary Tree.

- Find LCA of both nodes.
- Int d1 = distance from lca to node1
- Int d2 = distance from lca to node2
- Return d1+d2

```
void distance(Node* root,int data,int &d,int dist){
        if(root==NULL) return;
        if(root->data==data){
            d=dist;
            return;
        }
        distance(root->left,data,d,dist+1);
        distance(root->right,data,d,dist+1);
    }
    int findDist(Node* root, int a, int b) {
        // Your code here
        root = lca(root,a,b);
        int d1,d2;
        distance(root,a,d1,0);
        distance(root,b,d2,0);
```

14. Print all start index number from where the anagrams of given pattern matches.

Use of Sliding window,

- Create freqTxt and freqPat , and map all frequency to it upto Pattern’s size.
- Interate from pattern’s size to txt’s size
    - If both freqTxt and freqPat is equall then -> {i-M} index
    - FreqTxt[i-m]- -
    - FreqTxt[i]+ +

```
bool compare(vector<int> f,vector<int> s){
        for(int i=0;i<26;i++){
            if(f[i]!=s[i]) return false;
        }
        return true;
    }
        int search(string pat, string txt) {
            // code here
            int m = pat.size();
            int n = txt.size();

            vector<int> patFreq(26,0),txtFreq(26,0);

➔           for(int i=0;i<m;i++){
➔               patFreq[pat[i]-97]++;
➔               txtFreq[txt[i]-97]++;
➔           }
➔           int count=0;
➔           for(int i=m;i<n;i++){
➔               if(compare(patFreq,txtFreq)) count++;
➔
➔               txtFreq[txt[i]-97]++;
➔               txtFreq[txt[i-m]-97]--;
➔           }
➔           if(compare(patFreq,txtFreq))
➔                   count++;
➔           return count;
➔       }
```

15. Reverse a stack using Recursion

```
  void reverse(stack<int> &st){

       if(st.empty()) return ;
        int x= st.top(); st.pop();
        reverse(st); st.push(x);}
```

16. Print Next Greater Element (NGE)

- Push -1 to answer vector, and push last element to stack as well
- Traverse from back to 0 in array

```
If( st.top() > arr[i])
   ans.push_back(top)
   st.push(arr[i])

   else while(st.empty() not)

   check for above condition or else POP
```

return reverse Array Ans.

```
for(int i=n-2;i>=0;i--){
            if(num2[i] < st.top()){
                nge.push_back(st.top());
                st.push(num2[i]);
            }
            else{
                while(!st.empty()){
                    if(num2[i] < st.top()){
                        nge.push_back(st.top());
                        st.push(num2[i]);
                        break;
                    }
                    else{
                        st.pop();
                    }
                }
                if(st.empty()){
                    nge.push_back(-1);
                    st.push(num2[i]);
                }
            }
        }
```
17. (Not Done)

18. Peak Element (Increasing and Decreasing) find the Max Value

Brute Force : Traverse the entire and find the max Value

Use of binary search.

```
    int n = nums.size();

     if(n==1){
         return 0;
     }
     else if(nums[0]>nums[1]) return 0;
     else if(nums[n-1] > nums[n-2]) return n-1;
     else{
         int left=0,right=n-1;
         while(left<=right){
         int mid = (left+right)/2;
         if(nums[mid] >= nums[mid+1] and nums[mid]>=nums[mid-1]){
             return mid;
         }
         else if(nums[mid+1]>nums[mid]) left=mid+1;
         else right = mid-1;
     }
    }
     return -1;
```

19. Celebrity Problem

1. Graph

        a. Calculate Incoming (sum of column)
        b. Calculate Outgoing (sum of row)
        c. If(incoming==n-1 and outgoing==0) it is celebrity
2. Use Stack
        a. Push all (0 to N-1) into stack
        b. Now for N-1 times, take top two values
        c. If one knows two then two might be celeb
        d. If two knows one then one might be celeb
        e. Push celeb again
        f. Return top

```
stack<int> st;
        for(int i=0;i<n;i++) st.push(i);

        int count=0;
        while(count<n-1){
            int a = st.top();
            st.pop();
            int b = st.top();
            st.pop();

            if(M[a][b]) st.push(b);
            else st.push(a);
            count++;
        }
```

20. Palindrome Pairs

Map reverse of string to index mp[string_rev]=i

```
for i=0 to n
      for j=0 to words.size()
                   suffix = substr(0,j)
                   prefix = substr(j)

                   if(**suffix is present in map** and **prefix is Pal**
and **suffix_index!=i**)
                         push i and mp[suffix]
                        if (**suffix is not empty** and **prefix is
present** and **suffix is Pal** and **prefix_ind!=i** )
                              push  mp[prefix],i
```

```
vector<vector<int>> palindromePairs(vector<string>& words) {
        unordered_map<string, int>mp;
        vector<vector<int>>ans;
        int m = words.size();
        for(int i = 0; i < m; i++) {
            string s = words[i];
            reverse(s.begin(), s.end());
            mp[s] = i;
        }
        if(mp.find("") != mp.end()) {
            for(int i = 0; i < words.size(); i++) {
                if(i == mp[""]) continue;
                if(isPalindrom(words[i])) ans.push_back({mp[""], i});
            }
        }
        for(int i = 0; i < m; i++) {
            int n = words[i].size();
            for(int j = 0; j < n; j++) {
                string left = words[i].substr(0, j);
                string right = words[i].substr(j, words[i].size() - j);
                if(mp.find(left) != mp.end() && isPalindrom(right) && mp[left] !=
i) ans.push_back({i, mp[left]});
                if(mp.find(right) != mp.end() && isPalindrom(left) && mp[right]
!= i) ans.push_back({mp[right], i});
            }
        }
        return ans;
    }
```

21. Count All possible path from top left to bottom right

```
if(i>=m or j>=n) return 0;
if(i==m-1 and j=n-1) return 1;
return dfs(i+1,j)  + dfs(i,j+1);
```

22. Maximum Path sum in a binary Tree

Use of concept of Height of Tree.

We go left and right

Calculate max maxi for => (left,right)+node->data and node->data

```
Calculte max top for left+right+node->data and maxi
Res = max(res,top)
Return maxi;
```

```
int maxi=0;
    int solve(TreeNode* root,int &res){
        if(root==NULL) return 0;

        int leftsum = solve(root->left,res);
        int rightsum = solve(root->right,res);

        int maxi = max(max(leftsum,rightsum)+root->val,root->val);

        int maxtop = max(maxi,leftsum+rightsum+root->val);

        res = max(res,maxtop);

        return maxi;
    }
```

23. (Not done yet)

24. --

25. Knight Walk Problem

1. Create visited array
2. Queue<int,int,int> field (x,y,dist)
3. Q.push(initX,initY,0);

While q is not empty traverse it and then visited all 8 direction

If at any point x==finX and y==finY return distance;

```
vector<vector<int>> vis(N,vector<int>(N,0));
            int x = KnightPos[0]-1;
            int y = KnightPos[1]-1;

            int tx = TargetPos[0]-1;
            int ty = TargetPos[1]-1;

            if(x==tx and y==ty) return 0;

            int dx[]={2,2,-2,-2,1,1,-1,-1};
            int dy[]={1,-1,1,-1,2,-2,2,-2};

            queue<pair<pair<int,int>,int>>q;
            q.push({{x,y},0});

            while(!q.empty()){
                auto top = q.front();
                q.pop();

                int nx = top.first.first;
                int ny = top.first.second;
                int dis = top.second;

                for(int i=0;i<8;i++){
                    x = nx+dx[i];
                    y = ny+dy[i];

                    if(x==tx and y==ty) return dis+1;

                    if(isSafe(x,y,vis,N)){
                        vis[x][y]=1;
                        q.push({{x,y},dis+1});
                    }
                }
            }
            return -1;
```

26. (Not done yet)

27. Bottom View of binary tree

Use concept of vertical traversal (Line)

Queue of pair Node*,int => node and line

Map of <int int> node->data and line

Print the last node of every line

```
map<int,int> mp;
        queue<pair<Node*,int>> q;
        q.push({root,0});

        while(!q.empty()){
            Node* p = q.front().first;
            int x = q.front().second;
            q.pop();
            mp[x]=p->data;
            if(p->left) q.push({p->left,x-1});
            if(p->right) q.push({p->right,x+1});
        }
```

28. Handshake Problem

a. (n-1) + fun(n-1)
b. nC2
c. n*(n-1)/2

29. Kth Largest number in BST

a. Do a reverse inorder traversal and keep a count
b. When count==k return our node

```
 void kthLargestUtil(Node *root, int k, int &c)

{
    if (root == NULL || c >= k)
        return;
    kthLargestUtil(root->right, k, c);
    c++;
    if (c == k) return root->data;
    kthLargestUtil(root->left, k, c);
}
```

30. Check cycle in Directed graph or not

a. Created 2 visited arrays vis and dfsVis
b. Mark both as true
c. Now call dfs for neighbour nodes.
d. If vis and dfs both true return TRUE
e. Else call dfs()

Mark Dfsvis=false at end

```
vis[i]=true;
        dfsvis[i]=true;

        for(auto node : adj[i]){
            if(vis[node] and dfsvis[node]) return true;
            if(!vis[node]){
                if(checkForCycle(node,vis,dfsvis,adj)) return true;
            }
        }
      dfsvis[i]=false;
      return false;
```

31. (Not Done)

32. Kth Largest Number in an Array

1. Use sorting and return arr[k-1]
2. Use of Heap
3. Use of Quick Select

```
// If k is smaller than number of elements in array
    if (K > 0 && K <= r - l + 1) {

        // Partition the array around last element and get
        // position of pivot element in sorted array
        int pos = partition(arr, l, r);

        // If position is same as k
        if (pos - l == K - 1)
            return arr[pos];
        if (pos - l > K - 1) // If position is more, recur
                             // for left subarray
            return kthSmallest(arr, l, pos - 1, K);

        // Else recur for right subarray
        return kthSmallest(arr, pos + 1, r,
                           K - pos + l - 1);
    }

```

33. Print Duplicates elements in a string

a. Sort and print
b. Use of Frequency Array

34. Gold Mine Problem

a. Mark visited
b. Traverse left and right calculate maxLR
c. Traverse up and down calculate maxUD
d. Max = max(maxLR, maxUD)
e. Return gold[i][j]+max
f. Mark unvisited

```
if(row<0 or row>=m or col<0 or col>=n or grid[row][col]==0 or vis[row][col]){

            return 0;
        }
        vis[row][col]=1;
        int left = solve(row,col-1,grid,vis,m,n);
        int right = solve(row,col+1,grid,vis,m,n);
        int maxLeftRight = max(left,right);

        int up = solve(row-1,col,grid,vis,m,n);
        int down = solve(row+1,col,grid,vis,m,n);
        int maxUpDown = max(up,down);

        int finalMax = max(maxLeftRight,maxUpDown);
        finalMax+=grid[row][col];
        vis[row][col]=0;
        return finalMax;
```

35. (Not Done Yet)
36. Return the Nth node from the end of the linked List
    a. Init slow=head,fast=head
    b. for fast for N times
    c. while fast!=null
        i. slow=slow->next and fast=fast->next

    return slow->next;

```
ListNode* slow=dummy,*fast=dummy;
        for(int i=1;i<=n;i++) fast = fast->next;
        while(fast->next!=NULL){
            slow=slow->next;
            fast = fast->next;
        }
       return slow->next;
```

37. (Not done yet)
38. (Not done yet)
39. Find the longest Palindromic substring (only length is done)

```
If(i<0 or j<0) return 0;
If(s[i]==t[j]) return 1 + lps(i-1,j-1)
Else return lps(i-1,j-1)
```

40. Find the lowest possible sum of maxi and min values of a subarray

a. Sliding window (N*M)
b. Optimize:
   i. Calculate sum of adjacent elements only with max and min values

An **efficient solution** is based on the fact that adding any element to a subarray would not increase sum of maximum and minimum. Consider the array [a1, a2, a3, a4, a5….an] Each ai will be minimum of some subarray [al, ar] such that i lies between [l, r] and all elements in the subarray are greater than or equal to ai. The cost of such subarray would be ai + max(subarray). Since the max of an array will never decrease on adding elements to the array, It will only increase if we add larger elements so It is always optimal to consider only those subarrays having length 2.
In short consider all subarrays of length 2 and compare sum and take the minimum one which will reduce the time complexity by O(N) now we have to run the loop only once.

```
int maxSum(int arr[], int n)
{
    if (n < 2)
        return -1;
    int ans = arr[0] + arr[1];
    for (int i = 1; i + 1 < n; i++)
        ans = min(ans, (arr[i] + arr[i + 1]));
    return ans;
}
```

41. Pairwise swap in Linked List

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

        ListNode* temp = head->next;
        head->next = swapPairs(head->next->next);
        temp->next=head;

        return temp;
```

42. Cousins of all Nodes.

Marks every node parents

Do a level order traversal

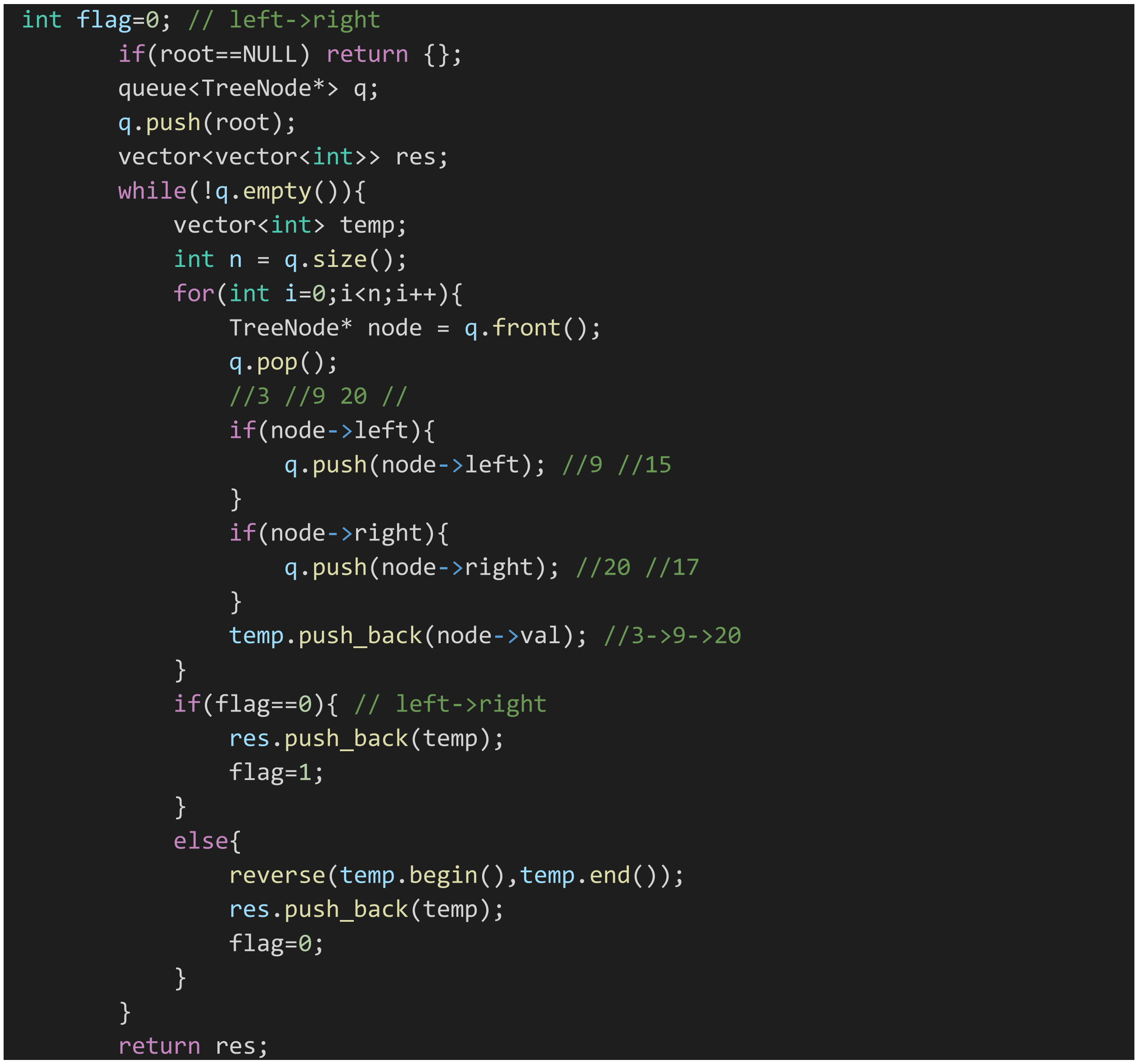
If on same level and different parents then they are cousins

```
queue<TreeNode*> q;
        if(root==NULL) return false;
        q.push(root);

        while(!q.empty()){
            int n = q.size();
            map<int,int> parChild;
            for(int i=0;i<n;i++){
                TreeNode* node = q.front();
                q.pop();
                if(node==root){
                    parChild[node->val]=0;
                }
                if(node->left){
                    parChild[node->left->val]=node->val;
                    q.push(node->left);
                }
                if(node->right){
                    parChild[node->right->val]=node->val;
                    q.push(node->right);
                }
            }
            if(parChild[x] and parChild[y]){
                if(parChild[x]!=parChild[y]) return true;
            }
        }
        return false;
```

43. Zig Zag Traversal of Binary Tree

```
Create a flag=0;
Do a level order traversal
If flag==0
      Add vector
      Flag=1
If flag==1
      Add reverse vector
      Flag=0;
```



```
int flag=0; // left->right
       if(root==NULL) return {};
       queue<TreeNode*> q;
       q.push(root);
       vector<vector<int>> res;
       while(!q.empty()){
           vector<int> temp;
           int n = q.size();
           for(int i=0;i<n;i++){
               TreeNode* node = q.front();
               q.pop();
               //3 //9 20 //
               if(node->left){
                   q.push(node->left); //9 //15
               }
               if(node->right){
                   q.push(node->right); //20 //17
               }
               temp.push_back(node->val); //3->9->20
           }
           if(flag==0){ // left->right
               res.push_back(temp);
               flag=1;
           }
           else{
               reverse(temp.begin(),temp.end());
               res.push_back(temp);
               flag=0;
           }
       }
       return res;
```