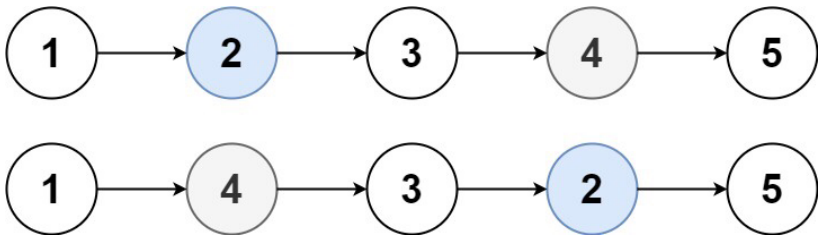
(12) Swapping Nodes in Linked List →

Eg:



k=2

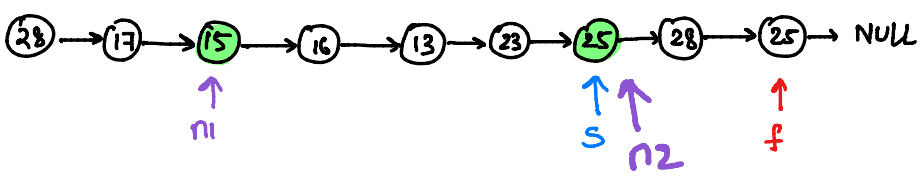
given a linkedlist swap k^{th} node from both ends.

Eg

* for k-1 times iterate f & mark n1, then iterate s & f till f is not NULL, once null mark s as n2.

(as it is 1 indexed)

swap (n1, n2)



(f → next == NULL, so s is n2)

swap (15, 25)

Result ⇒ 28 17 25 16 13 23 15 28 25 NULL

code →

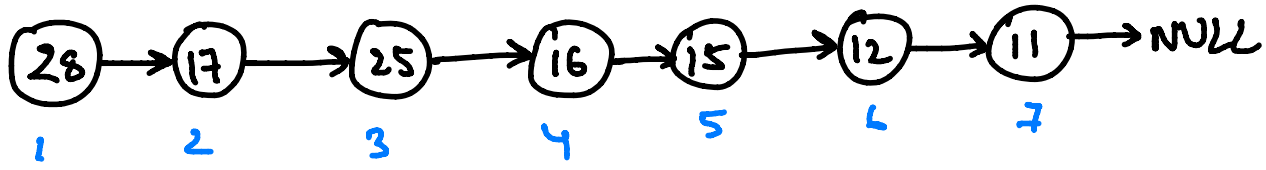
```cpp
class Solution {
public:
    ListNode* swapNodes(ListNode* head, int k) {
        ListNode *slow = head, *fast = head, *n1 = head;
        // finding n1
        for(int i=0; i<k-1; i++){
            fast = fast->next;
            n1 = fast;
        }
        // finding n2 (i.e slow)
        while(fast->next!=NULL){
            fast = fast->next;
            slow = slow->next;
        }
        // swapping
        int n1_val = n1->val;
        n1->val = slow->val;
        slow->val = n1_val;
        return head;
    }
};
```

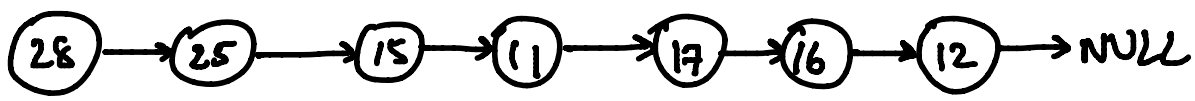
TC → O(n)
SC → O(1)

(13) Odd Even Linked List →

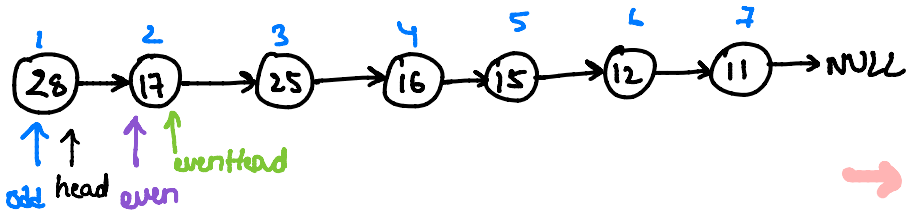
Given a linkedlist group all odd indices nodes followed by even nodes

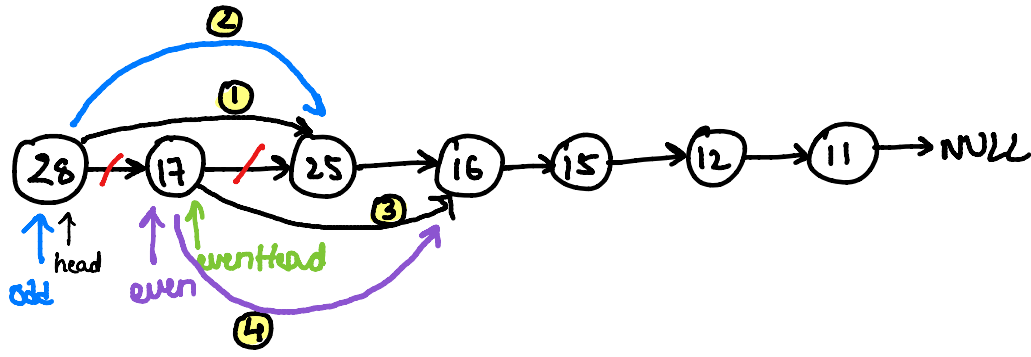
Eg

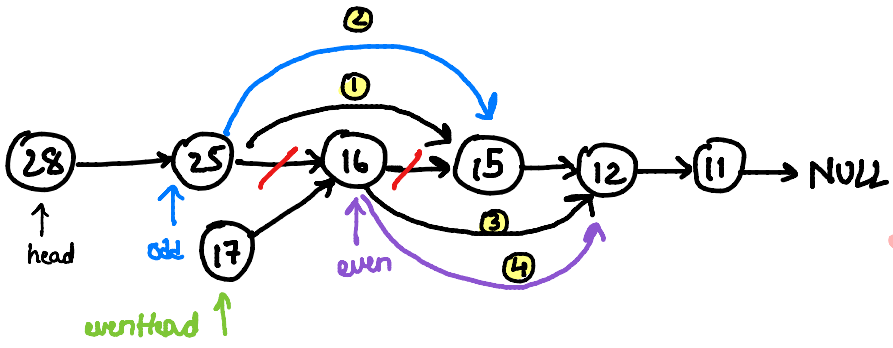


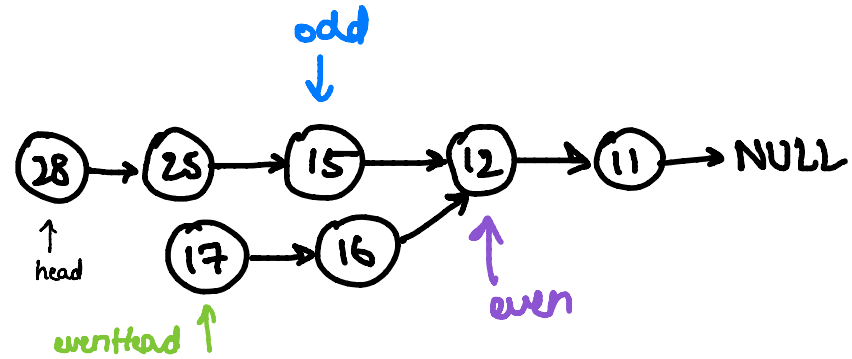


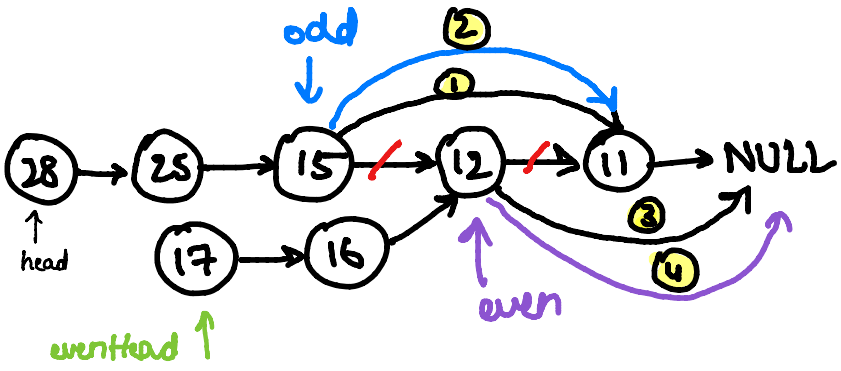
⇒

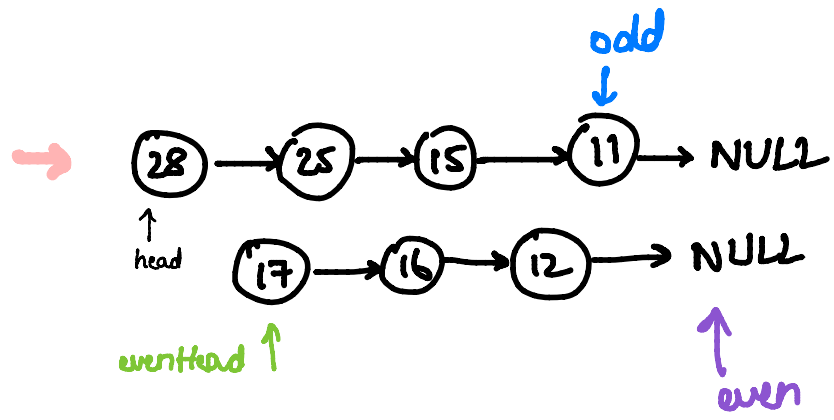


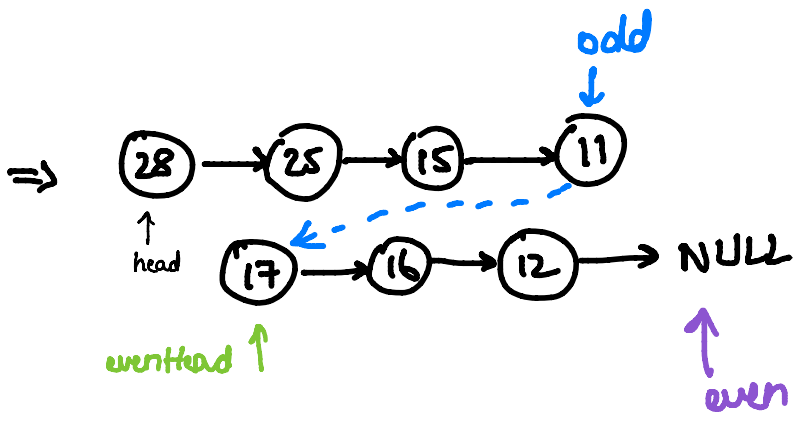


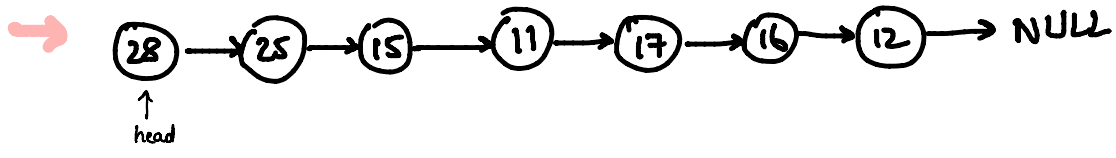












odd → next = eventhead

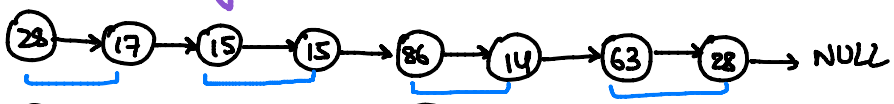
code →

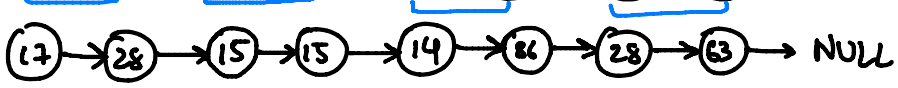
```cpp
class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {
        if(!head) return NULL;

        ListNode *even =  head->next;
        ListNode *odd = head;
        ListNode *evenHead = even;

        while(even && even->next){
            odd->next=even->next;
            odd=odd->next;
            even->next=odd->next;
            even=even->next;
        }

        // like odd and even lists
        odd->next = evenHead;
        return head;
    }
};
```

TC → O(n)
SC → O(1)

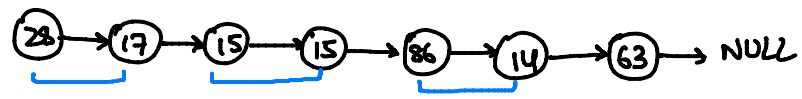
(14) Swap Nodes in Pairs → Given a linkedlist swap adjacent nodes.

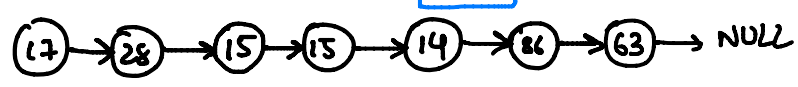
Even length

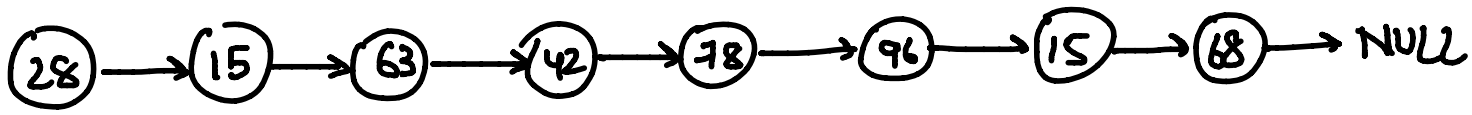
Eg 

Ans ⇒ 

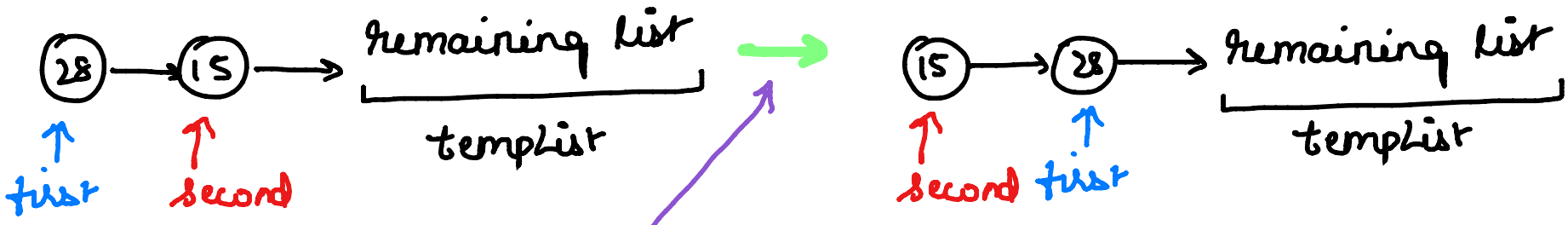
odd length

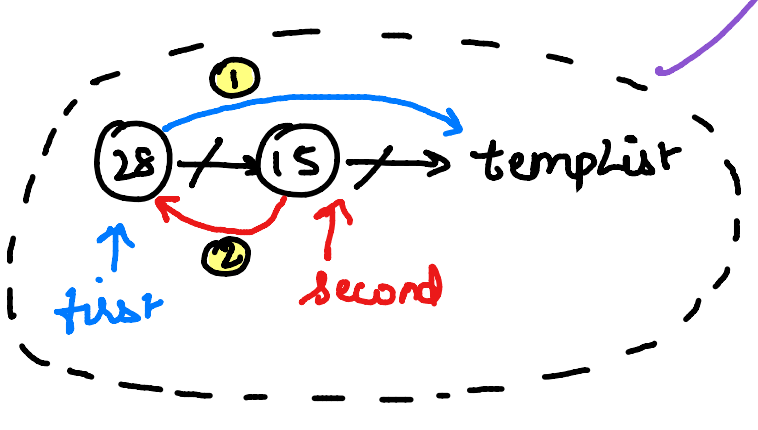


Ans ⇒ 

Eg → 

Consider for 1st pair,





templist = second → next

first → next = templist

second → next = first

} Logic

Solve recursively for all pairs.

Code →

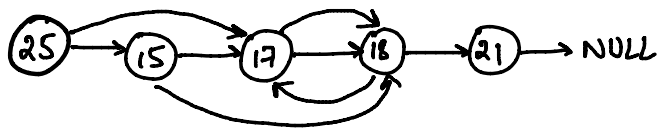
```java
class Solution {
public:
    ListNode* SwapAdjacentNodes(ListNode* head)
    {
        if(head==NULL || head->next==NULL)  return head;
        ListNode *first = head;
        ListNode *second = head->next;
        // start logic
        ListNode *tempList = SwapAdjacentNodes(second->next);
        first->next = temp;
        second->next = first;
        return second;
    }
    ListNode* swapPairs(ListNode* head) {
        return SwapAdjacentNodes(head);
    }
};
```

TC → O(N)

SC → O(1)

Recursive stack → O(N/2) ≃ O(N)

(15) Copy list with random pointer → given a list, clone & return.

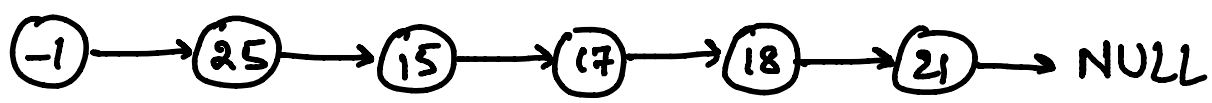
Eg 

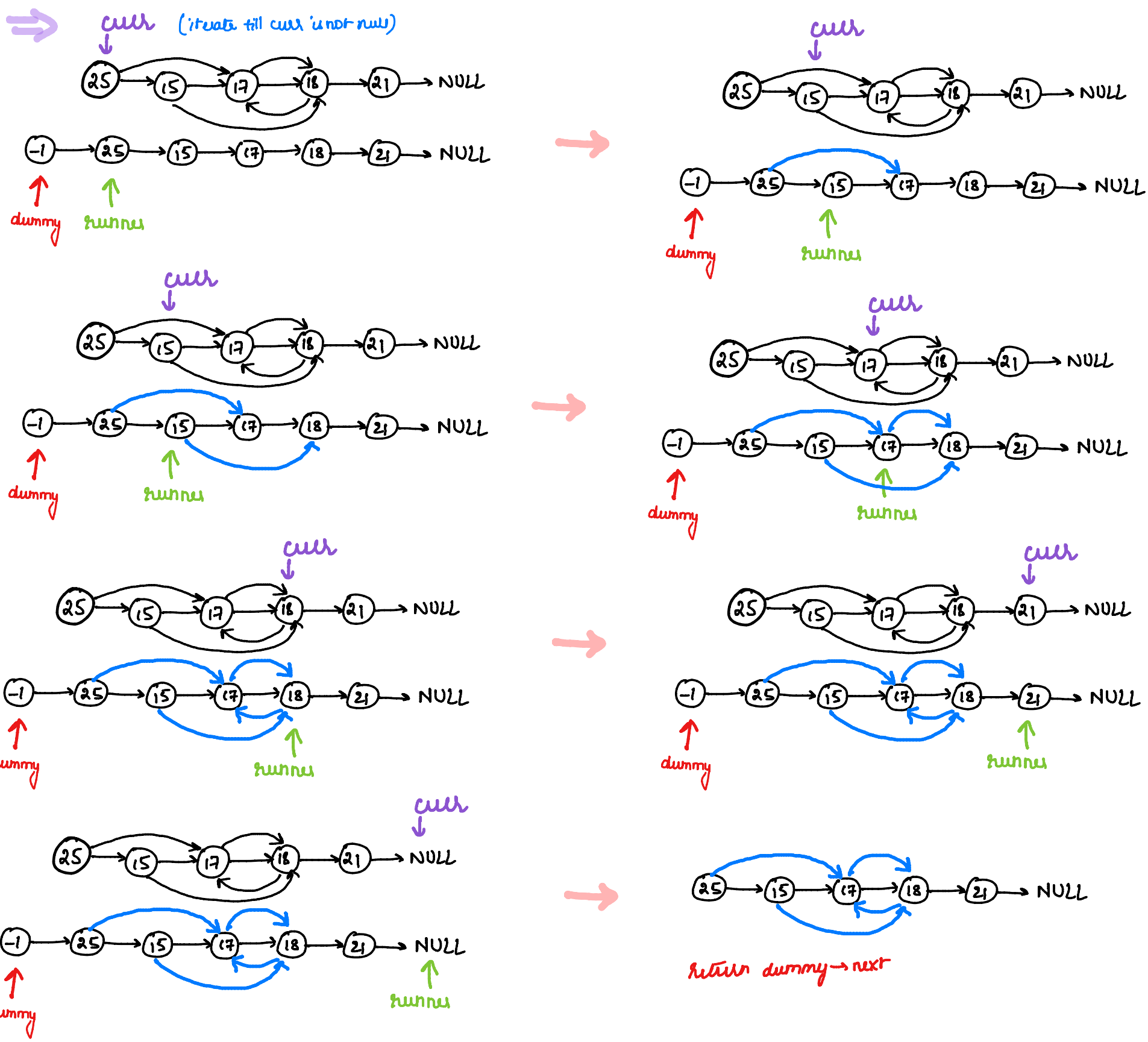
mp

25	25
15	15
17	17
18	18
21	21
NULL	NULL
Old	New

→ In 1st iteration create the list without random pointers & also maintain hashmap for mapping nodes pointed by random pointers

→ In 2nd iteration use map to link nodes pointed by random pointers

After 1st iteration → 

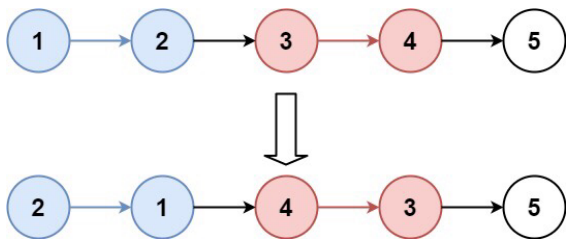


Code →

TC → O(n)
SC → O(n)

```cpp
class Solution {
public:
    Node* copyRandomList(Node* head) {

        unordered_map<Node*, Node*> mp;
        Node *dummy = new Node(100001);
        Node *runner = dummy, *curr = head;

        // initial iteration
        while(curr != NULL){
            Node *newNode = new Node(curr->val);
            runner->next = newNode;
            mp[curr] = newNode;
            curr = curr->next;
            runner = runner->next;
        }

        // setting starting points in both lists
        curr = head;
        runner = dummy->next;

        // setting the random pointers
        while(curr != NULL){
            if(curr->random != NULL)
                runner->random = mp[curr->random];
            runner = runner->next;
            curr = curr->next;
        }

        return dummy->next;
    }
};
```

(16) Reverse Nodes in K-Group



Given a linkedlist & K, return a list with reversed nodes by K-groups.

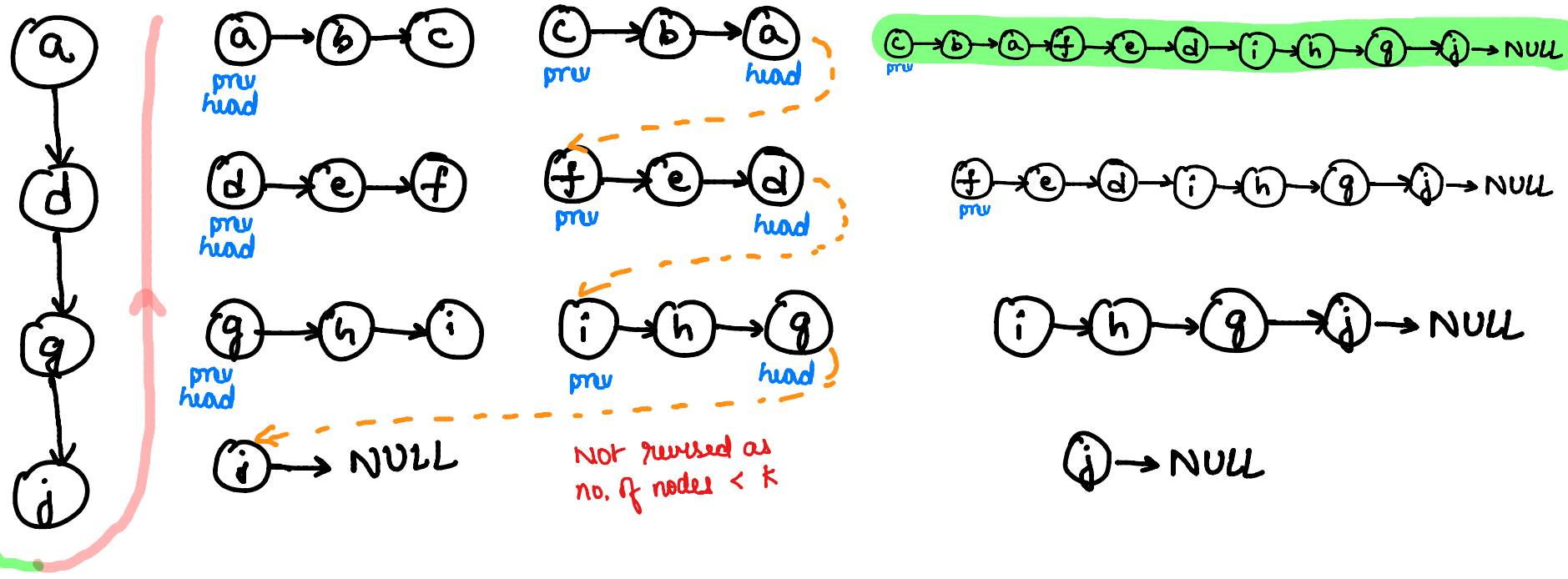
↳ K = 2

Eg. a → b → c → d → e → f → g → h → i → j → NULL

k = 3

Ans ⇒ c → b → a → f → e → d → i → h → g → j → NULL

Normal | Reversed | Returned list



* Consider the case of f → e → d (prev, head), then tempList = i → h → g

linking would happen as head → next = tempList & this list would become tempList to c → b → a

code →

TC → O(n)
SC → O(1)

```cpp
class Solution {
public:
    ListNode* reverseList(ListNode* head)
    {
        ListNode *prev = NULL, *curr = head, *temp;
        while (curr!=NULL)
        {
            temp = curr->next;
            curr->next = prev;
            prev = curr;
            curr = temp;
        }
        return prev;
    }

    ListNode* reverseInGroups(ListNode* head, int k)
    {
        ListNode *curr = head;
        int currlen = 1;
        if(head == NULL) return head;
        while(curr->next!=NULL && currlen<k ){
            curr=curr->next;
            currlen+=1;
        }
        if(currlen<k)   return head;
        ListNode *tempNode = curr->next;
        curr->next = NULL;

        // start linking
        ListNode *tempList = reverseInGroups(tempNode,k);
        ListNode *prev = reverseList(head);
        head->next = tempList;
        return prev;
    }

    ListNode* reverseKGroup(ListNode* head, int k) {
        return reverseInGroups(head,k);
    }
};
```

(17) Design Linked List → Implementation of Doubly Linked List

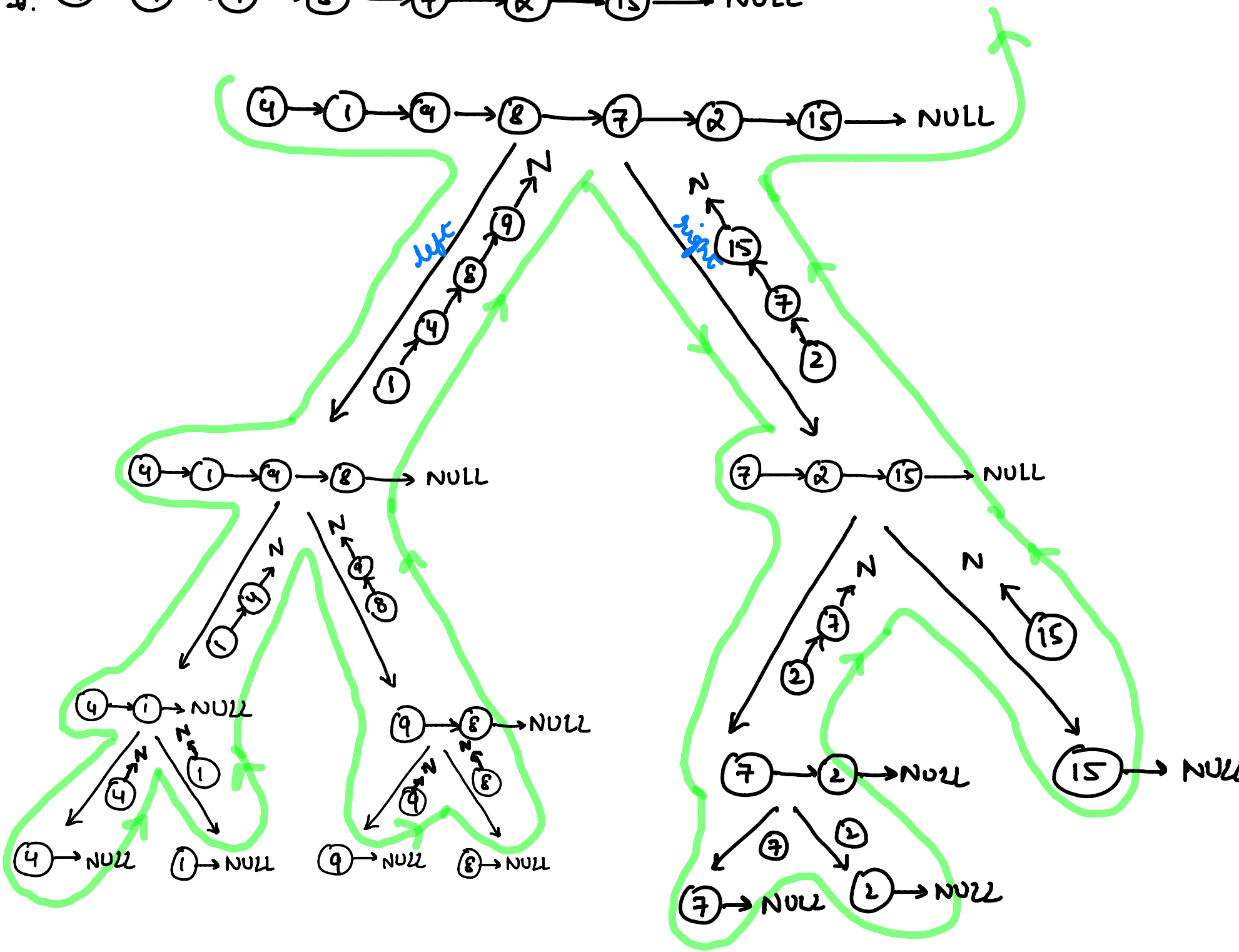
Code →

```
class Node{
    public:
    int val;
    Node* prev;
    Node* next;
    Node(int val){
        this->val=val;
        prev = nullptr;
        next = nullptr;
    }
};
class MyLinkedList {

public:
    Node *head;
    Node *tail;
    MyLinkedList(){
        head = nullptr;
        tail = nullptr;
    }

    int get(int index){
        if(head == NULL)    return -1;
        Node *temp = head;
        int count = 0;
        while(temp!=NULL){
            temp=temp->next;
            count++;
        }
        if(index>=count)    return -1;
        temp = head;
        while(temp != NULL && index>0){
            temp=temp->next;
            index--;
        }
        return temp->val;
    }

    void addAtHead(int val){
        Node *newNode = new Node(val);
        if(head == NULL){
            head = newNode;
            tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }

    void addAtTail(int val){
        Node *temp = head;
        if(head == NULL){
            Node *newNode = new Node(val);
            head = newNode;
            tail = newNode;
            return;
        }
        while(temp->next!=NULL){
            temp = temp->next;
        }
        Node *newNode = new Node(val);
        temp->next = newNode;
        newNode->prev = temp;
        tail = newNode;
    }
```

```
    void addAtIndex(int index, int val){
        Node *temp = head;
        int count = 0;
        while(temp != NULL){
            temp = temp->next;
            count++;
        }
        if(index>count) return ;
        if(index==0){
            addAtHead(val);
            return;
        } else if(count == index){
            addAtTail(val);
            return;
        } else {
            temp = head;
            while(temp != NULL && index>0){
                temp = temp->next;
                index--;
            }
            Node* newNode = new Node(val);
            Node* temp2 = temp->prev;
            temp->prev->next = newNode;
            temp->prev = newNode;
            newNode->prev = temp2;
            newNode->next = temp;
        }
    }

    void deleteAtIndex(int index) {
        Node* temp = head;
        int count = 0;
        while(temp != NULL){
            temp=temp->next;
            count++;
        }
        if(index>=count) return;
        if(count==1 && index==0){
            head = NULL;
            return;
        } else if(count-1 == index){
            tail = tail->prev;
            tail->next = NULL;
            return;
        } else {
            if(index==0){
                head->next->prev = NULL;
                head = head->next;
                return;
            }
            temp=head;
            while(temp!=NULL && index>0){
                temp = temp->next;
                index--;
            }
            Node* temp2 = temp->next;
            temp->prev->next = temp2;
            temp->next->prev = temp->prev;
        }
    }
};
```

(18) Sort List → By following Merge Sort.

Eg. 4 → 1 → 9 → 8 → 7 → 2 → 15 → NULL



In the last step while returning from both branches we have,

left = 1 → 4 → 8 → 9 → NULL & right = 2 → 7 → 15 → NULL

so create dummy node & merge, i.e -1 → 1 → 2 → 4 → 7 → 8 → 9 → 15 → NULL

return dummy→next, 1 → 2 → 4 → 7 → 8 → 9 → 15 → NULL

* The same happens at every intermediate merge

code →

Tc → O(m+n)

Sc → O(n)

```cpp
class Solution {
public:
    ListNode* merge(ListNode* l1, ListNode* l2) {
        ListNode *dummy = new ListNode(-1);
        ListNode *curr = dummy;
        while(l1 && l2){
            if(l1->val < l2->val){
                curr->next = l1;
                l1 = l1->next;
            } else {
                curr->next = l2;
                l2 = l2->next;
            }
            curr = curr->next;
        }
        if(l1) curr->next = l1;
        if(l2) curr->next = l2;

        return dummy->next;
    }

    ListNode* sortList(ListNode* head) {
        if(!head || !head->next) return head;

        ListNode *slow = head;
        ListNode *fast = head->next;
        while(fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        // dividing the lists into 2 parts
        fast = slow->next;
        slow->next = NULL;

        // sort & merge
        head = sortList(head);
        fast = sortList(fast)
        return merge(head, fast);
    }
};
```

Find the rest on
https://linktr.ee/KarunKarthik

Follow Karun Karthik For More Amazing Content !

https://www.linkedin.com/in/karun-karthik

Stacks & Queues

- Karun Karthik

Contents

Stack → Linear data structure

- follows LIFO, last in first out.
- operations → push : insert into top of stack
  pop : delete from top of stack.

Applications →

- by compilers to check for parenthesis
- to evaluate postfix expression
- to convert infix to postfix / prefix form.
- to store values during recursion & context during function call.
- to implement DFS of graph

Queue → Linear data structure

- follows FIFO, first in first out.
- operations → enqueue : insert element at end of queue
  dequeue : delete element at start of queue

Applications →

- Schedule jobs by CPU.
- to carry out FIFO basis like printing jobs.
- to implement BFS of graph

Types →

- Queue
- Circular Queue
- Doubly ended Queue
- Priority Queue.

① Implement a Stack using Linkedlist →

code →

```cpp
#include <bits/stdc++.h>
using namespace std;

struct Node{
    int data;
    Node* next;
};

Node* top;

void push(int data){
    Node* temp = new Node();
    if (!temp){
        cout << "\nStack Overflow";
        exit(1);
    }
    // add at the top and change top as new node
    temp->data = data;
    temp->next = top;
    top = temp;
}

int isEmpty(){
    // if top is null then empty
    return top == NULL;
}

int peek(){
    // if stack is not empty then return top node's data
    if (!isEmpty())
        return top->data;
    else
        exit(1);
}

void pop(){
    Node* temp;
    if(top == NULL){
        cout << "\nStack Underflow" << endl;
        exit(1);
    } else {
        temp = top;
        top = top->next;
        free(temp);
    }
}

```

② Implement a Queue using Linkedlist →

code →

```
class Node {
    int data;
    Node* next;
    Node(int d){
        data = d;
        next = NULL;
    }
};

class Queue {
    Node *front, *rear;

    Queue(){
        front = rear = NULL;
    }

    void enQueue(int x)
    {
        Node* temp = new Node(x);
        // if empty then node is both front and rear
        if (rear == NULL) {
            front = rear = temp;
            return;
        }
        // else add at end
        rear->next = temp;
        rear = temp;
    }

    void deQueue()
    {
        // if empty then return NULL
        if (front == NULL)
            return;
        // store front node
        Node* temp = front;
        front = front->next;

        // if front is NULL => no Nodes, change rear to NULL
        if (front == NULL)
            rear = NULL;
        // free node
        delete (temp);
    }
};
```

③ Implement a Stack using Queue →

If push, push into queue from rear end & pop & push all elements

If pop, pop from queue from front end.

Code →

```
class Stack {
  queue <int> q;

  public:

    // push operation
    void Push(int x) {
        int n = q.size();
        q.push(x);
        for (int i = 0; i < n; i++)
        {
            int value = q.front();
            q.pop();
            q.push(value);
        }
    }

    // pop operation
    int Pop() {
        int value = q.front();
        q.pop();
        return value;
    }

    // accessing top value
    int Top() {
        return q.front();
    }

    // finding size of stack
    int Size() {
        return q.size();
    }
};
```

④ Implement a Queue using Stack →

→ use 2 stacks.

→ while pop( ), shift all elements in 1 stack to another & return top value.

code →

```
class Queue {
  public:
    stack <int> in;
    stack <int> out;

    // push operation
    void Push(int x) {
        in.push(x);
    }

    // pop operation
    int Pop() {
        // shift in to out
        if (out.empty()){
            while (in.size()){
                out.push(in.top());
                in.pop();
            }
        }
        int x = out.top();
        out.pop();
        return x;
    }

    // peek operation
    int Top() {
        if (out.empty()){
            while (in.size()){
                out.push(in.top());
                in.pop();
            }
        }
        return out.top();
    }

    int Size() {
        return in.size()+out.size();
    }
};
```

(5) Valid parenthesis

s = "{}" → T s = "(){}" → T

s = "{[]}" → T s = ")[" → F

Eg s = "{[](){}()[]([])" → True.

→ if match found then pop, else push.

stack : [	]	s = "{[](){}()[]([])"
stack : [{	]	s = "{[](){}()[]([])"
stack : [{[	]	s = "{[](){}()[]([])"
stack : [{~~[~~	]	s = "{[]](){}()[]([])"
stack : [{(	]	s = "{[]()){}()[]([])"
stack : [{~~(~~	]	s = "{[]())}()[]([])"
stack : [~~{~~	]	s = "{[]())}()[]([])"
stack : [(	]	s = "{[]())}(()[]([])"
stack : [~~(~~	]	s = "{[]())}()[]([])"
stack : [[	]	s = "{[]())}()[][]([])"
stack : [~~[~~	]	s = "{[]())}()[]][]([])"
stack : [(	]	s = "{[]())}()[]([][])"
stack : [(~~[~~	]	s = "{[]())}()[]([]])"
stack : [~~(~~	]	s = "{[]())}()[]([])"

∴ As the stack is empty & string is completely traversed the string is valid ∴ return true.

Code →

```cpp
class Solution {
public:
    bool isValid(string s) {
        stack<char> st;
        for(auto i : s)
        {
            if (st.empty() || i == '(' || i == '{' || i == '[')
            {
                st.push(i);
            }
            else
            {
                if ((i == ')' && st.top() != '(') ||
                    (i == ']' && st.top() != '[') ||
                    (i == '}' && st.top() != '{')){
                    return false;
                }
                st.pop();
            }
        }
        return st.empty();
    }
};
```

TC → O(n)

SC → O(n)

⑥ Asteroid Collision →

only consider magnitude

+ve sign ⇒ right direction
-ve sign ⇒ left direction

if x & y collide then min(x,y) will be removed
if x == y then both will be removed.

Eg [5, 10, -5] 5,10 will not collide
10,-5 will collide & -5 will be removed

result = [5,10]

Eg [10, 6, -8, -8, 8, 9]

stack [ ] [10, 6, -8, -8, 8, 9]

stack [10,] [10, 6, -8, -8, 8, 9]

stack [10, 6] [10, 6, -8, -8, 8, 9] as 6 is +ve push

stack [10, 6] [10, 6, -8, -8, 8, 9] as 6 & 8 will collide (opp directions), 6 will be removed

stack [10] [10, 6, -8, -8, 8, 9] as 10 & 8 will collide (opp directions), 8 will be removed

stack [10] [10, 6, -8, -8, 8, 9] as 10 & 8 will collide (opp directions), 8 will be removed

stack [10, 8] [10, 6, -8, -8, 8, 9] as 8 is +ve push

stack [10, 8, 9] [10, 6, -8, -8, 8, 9] as 9 is +ve push

result = [10, 8, 9]

TC → O(2n) ≃ O(n) SC → O(n)
worst case

code →

```cpp
class Solution {
public:
    vector<int> asteroidCollision(vector<int>& asteroids) {

        vector<int> res;

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

            if(res.empty() || asteroids[i]>0)
                res.push_back(asteroids[i]);
            else {

                while(!res.empty() && res.back()>0 && res.back()<abs(asteroids[i])) {
                    res.pop_back();
                }

                if(!res.empty() && res.back()+asteroids[i]==0)
                    res.pop_back();
                else if(res.empty() || res.back()<0)
                    res.push_back(asteroids[i]);
            }
        }
        return res;
    }
};
```

⑦ Next greater element → [2, 4, 1, 3, 1, 6]

Eg [4, 5, 2, 25]

4 → 5 2 → 25
5 → 25 25 → -1

2 → 4 3 → 6
4 → 6 1 → 6
1 → 3 6 → -1

→ Iterate from last & compare its value with top of stack
→ If stack is greater then its the next greater element
→ else keep popping till the next greater element is found.

Eg [11, 13, 3, 10, 7, 21, 26]

stack = [ ]	[11, 13, 3, 10, 7, 21, 26] ←	
stack = [26]	[11, 13, 3, 10, 7, 21, 26]	26 → -1
stack = [26, 21]	[11, 13, 3, 10, 7, 21, 26]	21 → 26
stack = [26, 21, 7]	[11, 13, 3, 10, 7, 21, 26]	7 → 21
stack = [26, 21, ~~7~~, 10]	[11, 13, 3, 10, 7, 21, 26]	pop 7, push 10 10 → 21
stack = [26, 21, 10]	[11, 13, 3, 10, 7, 21, 26]	3 → 10
stack = [26, 21, ~~10~~, ~~3~~, 13]	[11, 13, 3, 10, 7, 21, 26]	pop 3,10 push 13 13 → 21
stack = [26, 21, 13]	[11, 13, 3, 10, 7, 21, 26]	11 → 13

res = [13, 21, 10, 21, 21, 26, -1]

Code →

```cpp
class Solution
{
    public:
    //Function to find the next greater element for each element of the array.
    vector<long long> nextLargerElement(vector<long long> arr, int n){

        stack<long long> st;
        vector<long long> res(n);

        for(int i=n-1; i>=0 ; i--){
            long long currVal = arr[i];

            while(!st.empty() && st.top()<=currVal)
                st.pop();

            res[i] = st.empty()?-1:st.top();
            st.push(currVal);
        }
        return res;
    }
};
```

Tc → O(n)

Sc → O(n)

(8) Next Smaller element →

→ entire approach is similar to next greater element except for comparision.

Tc → O(n)

Sc → O(n)

code →

```
1  vector<int> nextSmallerElement(vector<int> &arr, int n)
2  {
3      stack<int> st;
4      vector<int> res(n);
5      for(int i=n-1; i>=0 ; i--){
6
7          long long currVal = arr[i];
8
9          while(!st.empty() && st.top()>=currVal)
10             st.pop();
11
12         res[i] = st.empty()?-1:st.top();
13         st.push(currVal);
14     }
15     return res;
16 }
```

(9) **Stock Span Problem** → given price quotes of stock for n days. we need to find span of stock on any particular day.

→ max no. of consecutive days for which price <= curr day's price

Eg [100, 80, 60, 70, 60, 75, 85]
0 1 2 3 4 5 6

stack = [stores indexes]

span = [0 0 0 0 0 0 0]
0 1 2 3 4 5 6

if currentElement > stack.top
  pop stack
else:
  span = currentIndex - stack.top

→ push index into stack after processing →

0 1 2 3 4 5 6

[100, 80, 60, 70, 60, 75, 85] span of 1st element = 1 — stack [0] — span [1 0 0 0 0 0 0]

[100, 80, 60, 70, 60, 75, 85] 80 > 100 ⇒ false ∴ span = 1 − 0 = 1 — stack [0, 1] — span [1 1 0 0 0 0 0]

[100, 80, 60, 70, 60, 75, 85] 60 > 100 ⇒ false ∴ span = 2 − 1 = 1 — stack [0, 1, 2] — span [1 1 1 0 0 0 0]

[100, 80, 60, 70, 60, 75, 85] 70 > 60 ⇒ true ∴ pop; 70 > 80 ⇒ false ∴ span = 3 − 1 = 2 — stack [0, 1, 3] — span [1 1 1 2 0 0 0]

[100, 80, 60, 70, 60, 75, 85] 60 > 70 ⇒ false ∴ span = 4 − 3 = 1 — stack [0, 1, 3, 4] — span [1 1 1 2 1 0 0]

[100, 80, 60, 70, 60, 75, 85] 75 > 60 ⇒ true ∴ pop; 75 > 70 ⇒ true ∴ pop; 75 > 80 ⇒ false; span = 5 − 1 = 4 — stack [0, 1, 5] — span [1 1 1 2 1 4 0]

[100, 80, 60, 70, 60, 75, 85] 85 > 75 ⇒ true ∴ pop; 85 > 80 ⇒ true ∴ pop; 85 > 100 ⇒ false; span = 6 − 0 = 6 — stack [0, 6] — span [1 1 1 2 1 4 6]

span = [1 1 1 2 1 4 6]
0 1 2 3 4 5 6

Code →

Tc → O(n)

Sc → O(n)

```cpp
class Solution
{
    public:
    //Function to calculate the span of stocks price for all n days.
    vector <int> calculateSpan(int price[], int n)
    {
        vector<int> span(n);
        stack<int> st;

        st.push(0);
        span[0] = 1;

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

            int currPrice = price[i];

            while(!st.empty() && currPrice >= price[st.top()])
                st.pop();

            if(st.empty()){
                span[i] = i+1;
            } else {
                span[i] = i-st.top();
            }

            st.push(i);
        }
        return span;
    }
};
```

(10) Celebrity Problem →

A Celebrity is a person, who is known to everyone & knows none.

given a square matrix M & if i^{th} person knows j^{th} person then M[i][j] = 1, else 0.

Eg →

$$M = \begin{matrix} & 0 & 1 & 2 \\ 0 & [0, & 1, & 0], \\ 1 & [0, & 0, & 0], \\ 2 & [0, & 1, & 0] \end{matrix} \qquad n = 3.$$

1. create Stack & push values from 0 to n-1.
2. do the following till stack more than has 1 value.
   - pop 1st element & let it to A
   - pop again & let it to B
   - if A knows B then push B else A.

→ Stack [ ] ⇒ Stack [0, 1, 2]

⇒ Stack [0, ~~1~~, ~~2~~] A = 2, B = 1 & M[2][1] == 1 (true) ∴ push 1 ⇒ Stack [0, 1]

Stack [~~0~~, ~~1~~] A = 1, B = 0 & M[1][0] == 1 (false) ∴ push 1 ⇒ Stack [1]

∴ as stack has only 1 element, stop.

⇒ Now pop the stack & consider it as celebrity & check for
- anyone doesn't know celeb (!M[i][celeb])
- if celeb knows anyone (M[celeb][i])

} return -1.

∴ from i = 0 to 2 & celeb = 1

- i = 0 (!M[0][1] or M[1][0]) = 0
- i = 1 skip as celeb is i
- i = 2 (!M[2][1] or M[1][2]) = 0

} all are failed i.e no violation of conditions.

∴ return celeb i.e 1

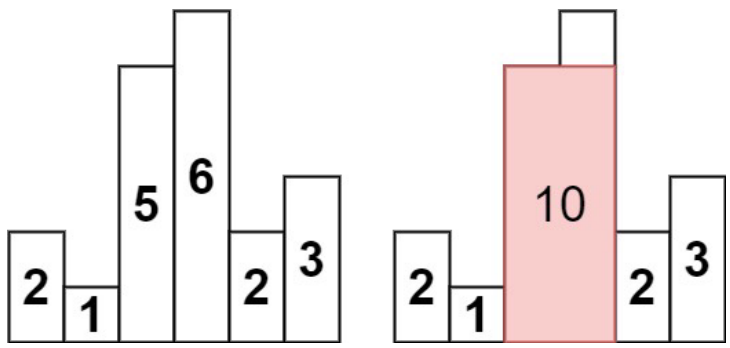
code →

TC = O(n)

SC = O(n)

```cpp
class Solution
{
    public:
    //Function to find if there is a celebrity in the party or not.
    int celebrity(vector<vector<int> >& M, int n) {

        stack<int> s;

        for(int i=0;i<n;i++)    s.push(i);

        // check and if is a celebrity then push into stack
        while(s.size()>1)
        {
            int a=s.top();
            s.pop();
            int b=s.top();
            s.pop();

            if(M[a][b]==1)
                s.push(b);
            else
               s.push(a);
        }

        int celeb = s.top();

        for (int i = 0; i < n; i++){
            // if i person doesn't know celeb or celeb knows anyone else
            // then return -1
            if ( (i!=celeb) && (!M[i][celeb]) || M[celeb][i] ))
                return -1;
        }

        return celeb;
    }
};
```

(11) Largest Rectangle in Histogram →



⇒ given an array of heights, return area of largest rectangle

Ans = 10.

0 1 2 3 4 5 Stack.

arr = [2, 1, 5, 6, 2, 3] [ ] area = 0 maxArea = 0

i=0 [2, 1, 5, 6, 2, 3] [0] area = 0 maxArea = 0

→ i=1 [2, 1, 5, 6, 2, 3] [0] area = 0 maxArea = 0

[~~0~~]

now arr[st.top()]^{2} > currElement^{1} ⇒ ht = arr[st.top()]^{2} & st.pop() ↑

& as stack is empty now, width = i^{1} & push(i)!

∴ ht = 2 & width = 1 ∴ area = 2 & maxArea = ∅ 2.

→ i=2 [2, 1, 5, 6, 2, 3] [1] area = 0 maxArea = 2

now arr[st.top()]^{1} > currElement^{5} ⇒ false ∴ push(i)^{2}

→ i=3 [2, 1, 5, 6, 2, 3] [1, 2] area = 0 maxArea = 2

now arr[st.top()]^{5} > currElement^{6} ⇒ false ∴ push(i)^{3}

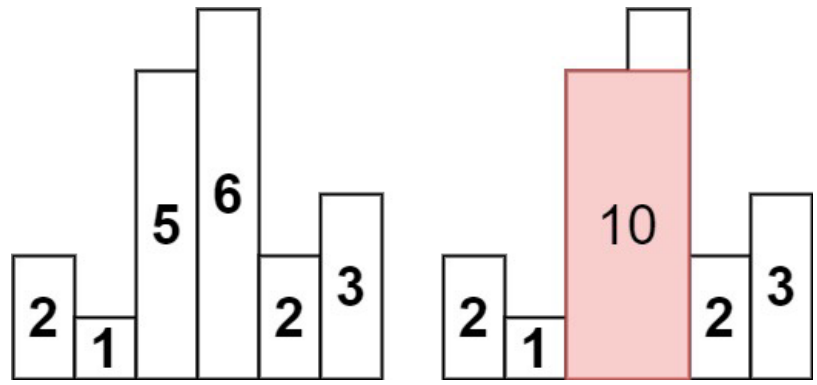
→ i=4 [2, 1, 5, 6, 2, 3] [1, 2, 3] area = 0 maxArea = 2

[1, 2, ~~3~~]

now arr[st.top()]^{6} > currElement^{2} ⇒ ht = arr[st.top()]^{6} & st.pop() ↑

width = i^{4} − st.top()^{2} − 1^{1} = 1 ∴ area = 6 * 1 = 6 maxArea = ~~2~~ 6.

& push(i)^{4}



0 1 2 3 4 5

→ [2, 1, 5, 6, 2, 3] [1, 2] area = 10 maxArea = 10

i = 4 5 2 5 [1, ~~2~~]

now arr[st.top()] > currElement ⇒ ht = arr[st.top()] & st.pop() ↑

width = 4 − 1 − 1 = 2 ∴ area = 5 * 2 = 10 maxArea = ~~8~~ 10.

& push(i)⁴

0 1 2 3 4 5

→ [2, 1, 5, 6, 2, 3] [1, 4] area = 0 maxArea = 10

i = 5 2 3

now arr[st.top()] > currentElement ⇒ false ∴ push(i)⁵

⇒ Last iteration to pop stack. ⇒ i = 6

0 1 2 3 4 5

→ [2, 1, 5, 6, 2, 3] [1, 4, 5] area = 3 maxArea = 10

3 [1, 4, ~~5~~]

ht = arr[st.top()] & pop() & as stack is not empty

width = 6 − 4 − 1 = 1 ∴ area = 3 * 1 = 3 maxArea = 10

0 1 2 3 4 5

→ [2, 1, 5, 6, 2, 3] [1, 4] area = 3 maxArea = 10

2 [1, ~~4~~]

ht = arr[st.top()] & pop() & as stack is not empty

width = 6 − 1 − 1 = 4 ∴ area = 2 * 4 = 8 maxArea = 10

0 1 2 3 4 5

→ [2, 1, 5, 6, 2, 3] [1,] area = 6 maxArea = 10

[1,]

ht = arr[st.top()] & pop() & as stack is empty

width = i = 6 ⇒ ∴ area = 1 * 6 = 6 maxArea = 10

∴ As stack is empty return maxArea = 10.

Code →

TC → O(n)

SC → O(n)

```cpp
class Solution {
public:
    int largestRectangleArea(vector<int>& heights) {
        stack < int > st;
        int maxArea = 0;
        int n = heights.size();

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

            while (!st.empty() && (i == n || heights[st.top()] >= heights[i])) {

                int height = heights[st.top()];
                st.pop();
                int width;
                if (st.empty()){
                    width = i;
                } else {
                    width = i - st.top() - 1;
                }

                int area = width*height;
                maxArea = max(maxArea, area);
            }
            st.push(i);
        }
        return maxArea;
    }
};
```

(12) Sliding Window Maximum →

→ process first 'k' elements before pushing into result arr.

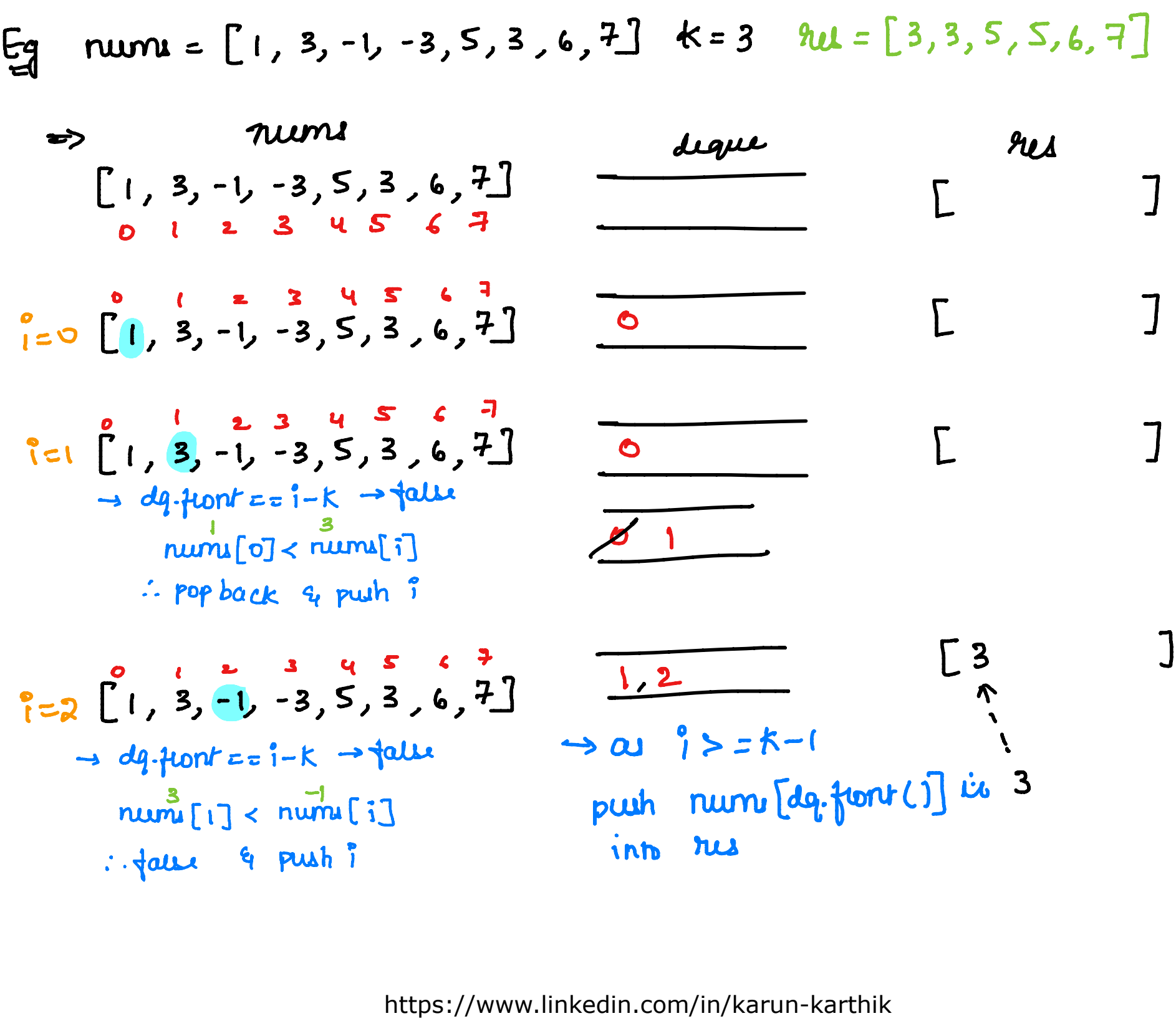
→ if dq.front() == i-k then pop-front (out of boundary case)

→ if nums[dq.back()] < nums[i] then pop-back
(meaningless to store smaller elements in window)

→ if i >= k-1 then push nums[dq.front()]

Eg nums = [1, 3, -1, -3, 5, 3, 6, 7] k = 3 res = [3, 3, 5, 5, 6, 7]

=>



i=3 [1, 3, -1, -3, 5, 3, 6, 7] (0 1 2 3 4 5 6 7)

→ dq.front == i-k → false

nums[2] < nums[i] (-1 < -3)

∴ false & push i

1, 2, 3

→ as i >= k-1

push nums[dq.front()] i.e 3 into res

[3, 3]

i=4 [1, 3, -1, -3, 5, 3, 6, 7]

→ dq.front == i-k true ∴ popfront

nums[3] < nums[i] (-3 < 5) ∴ pop_back

nums[2] < nums[i] (-1 < 5) ∴ pop_back

& push(i)

order of pop

① 1, ② 2, ③ 3, ④ 4

→ as i >= k-1

push nums[dq.front()] i.e 5 into res

[3, 3, 5]

i=5 [1, 3, -1, -3, 5, 3, 6, 7]

→ dq.front == i-k → false

nums[4] < nums[i] (5 < 3)

∴ false & push(i)

order of pop

4, 5

→ as i >= k-1

push nums[dq.front()] i.e 5 into res

[3, 3, 5, 5]

i=6 [1, 3, -1, -3, 5, 3, 6, 7]

→ dq.front == i-k → false

nums[5] < nums[i] (3 < 6) ∴ pop_back

nums[4] < nums[i] (5 < 6) ∴ pop_back

& push

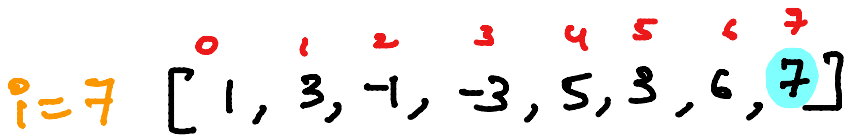
order of pop

② 4, ① 5, 6

→ as i >= k-1

push nums[dq.front()] i.e 6 into res

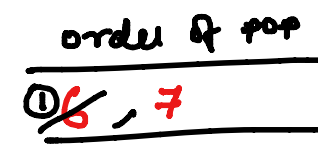
[3, 3, 5, 5, 6]



→ dq.front == i-k → false

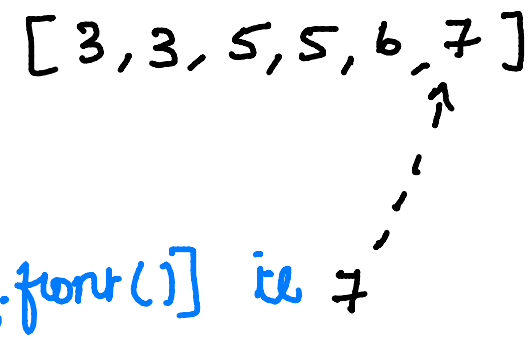
6 7
nums[6] < nums[i] ∴ pop_back

q push(i)



→ as i >= k-1

push nums[dq.front()] i.e 7 into res



code →

Tc → O(N)

Sc → O(K)

```cpp
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        deque <int> dq;
        vector <int> ans;
        for (int i = 0; i < nums.size(); i++) {

            if (!dq.empty() && dq.front() == i - k)
                dq.pop_front();

            while (!dq.empty() && nums[dq.back()] < nums[i])
                dq.pop_back();

            dq.push_back(i);

            if (i >= k - 1)
                ans.push_back(nums[dq.front()]);
        }
        return ans;
    }
};
```

Find the rest on

https://linktr.ee/KarunKarthik

Follow Karun Karthik For More Amazing Content !

Trees - Part 1

- Karun Karthik

Contents

Trees

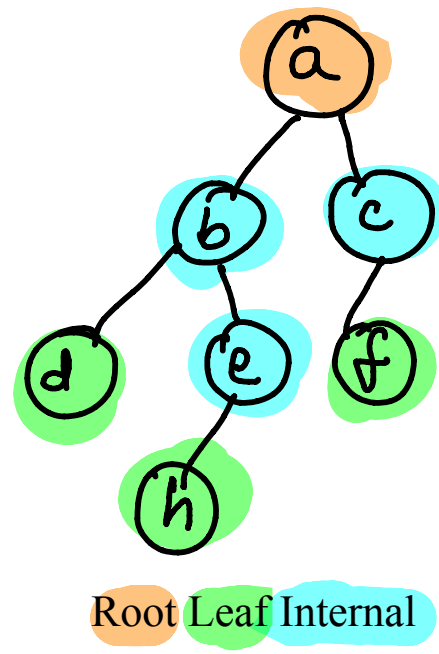
Tree - collection of tree-nodes

why trees?

1. Hiraichy
2. Computer system. (UNIX)

① Class Treenode
- data
- list<Treenode> children

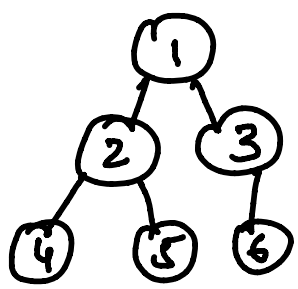
② Binary Tree → atmost 2 children (0,1,2)
- data
- leftchild
- rightchild



③ Types →

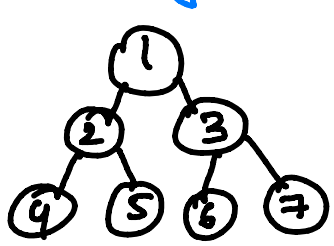
Ⓐ Complete Binary Tree
↳ all levels are completely filled except last one

Eg



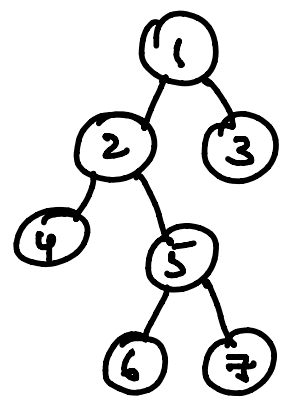
Ⓑ Perfect Binary Tree
↳ every internal node has exactly 2 children

Eg



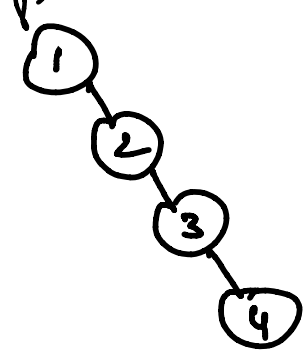
Ⓒ Full Binary tree
↳ if every node has 0 or 2 children

Eg



Ⓓ Skewed Binary Tree
(* used for finding complexity)
↳ all nodes have either one or no child.

Eg

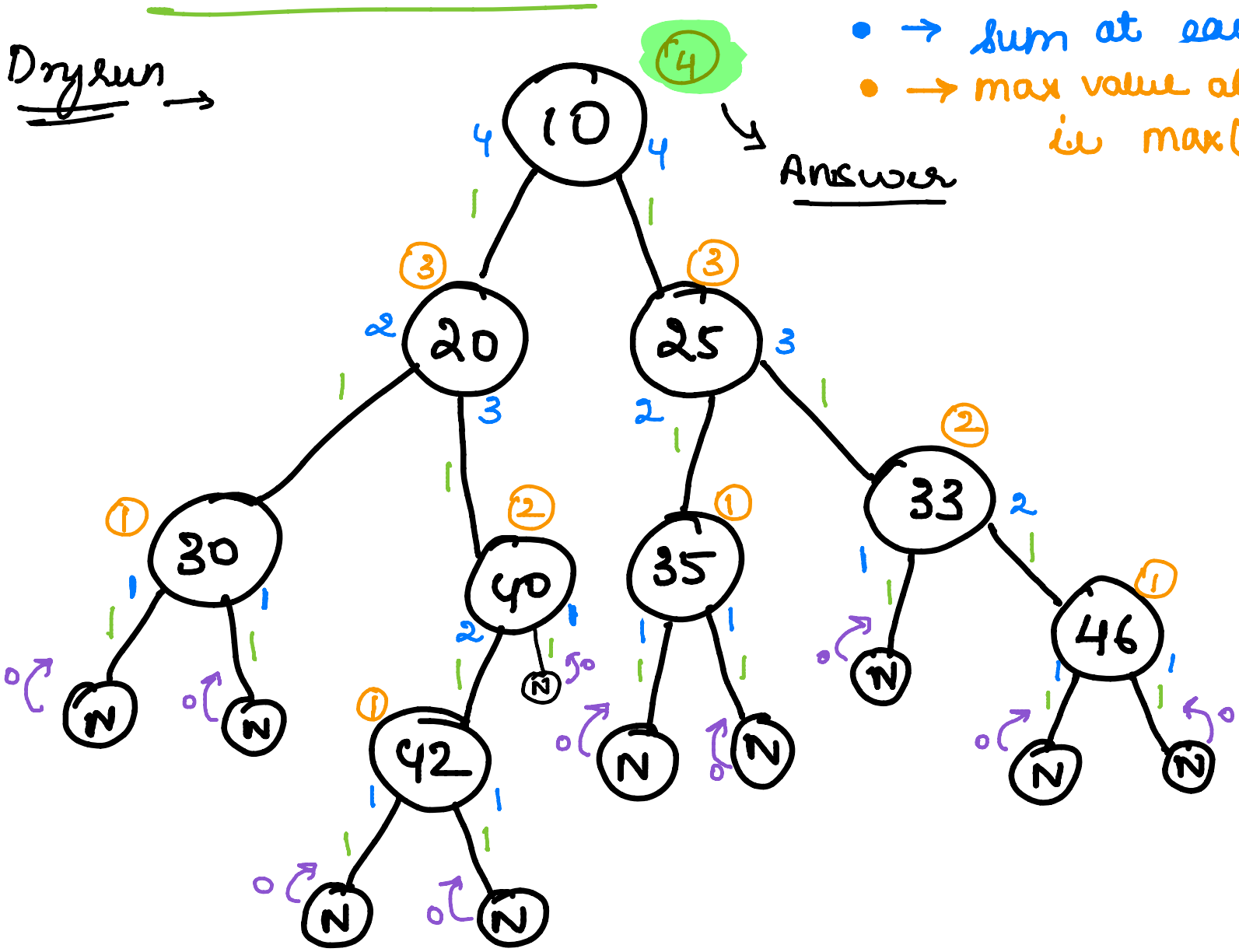


D1

① Depth of a binary tree (Max depth)

- → 1 added while returning.
- → sum at each node.
- → max value at node is max(left, right).

Dryrun →



if null then ht = 0

- consider max at a node ie either left or right

Tc → O(n)
SC → O(1)
AUX → O(h)
h → height

Code →

```C++
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nu
 *     TreeNode(int x) : val(x), left(nullptr), rig
 *     TreeNode(int x, TreeNode *left, TreeNode *r
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(root == NULL) return 0;

        int lefth= 1+ maxDepth(root->left);
        int righth = 1+maxDepth(root->right);
        return max(lefth,righth);
    }
};
```

② Maximum depth of n-ary tree

Idea is same as previous problem, only implementation changes

Code →

C++

```cpp
/*
// Definition for a Node.
class Node {
public:
    int val;
    vector<Node*> children;

    Node() {}

    Node(int _val) {
        val = _val;
    }

    Node(int _val, vector<Node*> _children) {
        val = _val;
        children = _children;
    }
};
*/

class Solution {
public:
    int maxDepth(Node* root) {
        if(root==NULL) return 0;
        int ans=0;
        for(int i=0;i<root->children.size();i++)
        {
            int tempans = maxDepth(root->children[i]);
            ans = max(ans,tempans);
        }
        return ans+1;
    }
};
```

D2

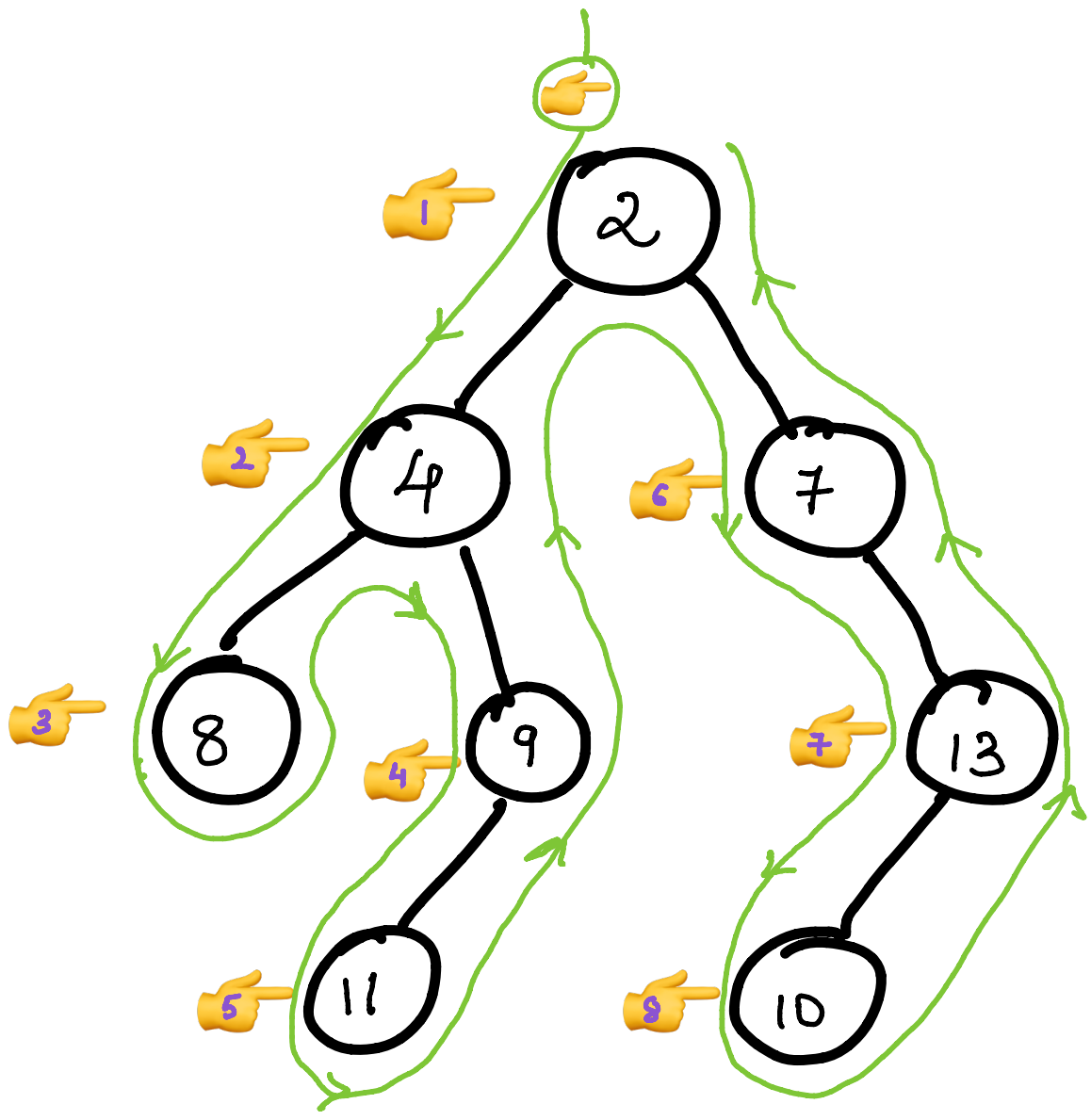
Traversals → DFS → Preorder
Inorder
Postorder

→ BFS → Level order

(A) Preorder →

processing order → node
left child
right child

Eg



* Point fingers as shown and traverse the tree starting from Root

* Order of visiting is the preorder traversal.

[2, 4, 8, 9, 11, 6, 13, 10]

TC → O(n)

SC → O(n) Recursive stack space → O(h) h → height.