WEEK – 1

1. **Set Matrix Zeros**

```
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {
        int rows = matrix.size(), cols = matrix[0].size();
        vector<int>temp1(rows,-1), temp2(cols,-1);
        for(int i=0;i<rows;i++){
            for(int j=0;j<cols;j++){
                if(matrix[i][j] == 0){
                    temp1[i]=0;
                    temp2[j]=0;
                }
            }
        }
        for(int i=0;i<rows;i++){
            for(int j=0;j<cols;j++){
                if(temp1[i]==0 || temp2[j]==0){
                    matrix[i][j] = 0;
                }
            }
        }
    }
};
```

2. **Pascal's Triangle**

```
vector<int> pascalTri(int i) {
    vector<int> v;
    int ans = 1;
    v.push_back(ans);
    for(int j=1;j<i;j++) {
```

```
        ans=ans*(i-j)/j;
        v.push_back(ans);
    }
    return v;
}
vector<vector<int>> pascalTriangle(int N) {
    // Write your code here.
    vector<vector <int>> r;
    for(int i=1;i<=N;i++) {
        r.push_back(pascalTri(i));
    }
    return r;
}
```

3. Next Permutation

```
class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        int ind = -1;
        int n = nums.size();
        for(int i = n-2; i>=0; i--) {
            if(nums[i] < nums[i+1]) {
                ind = i;
                break;
            }
        }
        if(ind == -1) {
            reverse(nums.begin(), nums.end());
            return ;
        }
        for(int i = n-1; i>ind; i--) {
```

```
            if(nums[i] > nums[ind]) {
                swap(nums[i], nums[ind]);
                break;
            }
        }
        reverse(nums.begin() + ind + 1, nums.end());
        return;
    }
};
```

4. Kadane's Algorithm

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int sum = 0;
        int maxi = INT_MIN;
        for(auto it : nums) {
            sum += it;
            maxi = max(sum, maxi);
            if(sum < 0) sum = 0;
        }
        return maxi;
    }
};
```

5. Sort an array of 0's, 1's and 2's

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int l = 0, r = nums.size() - 1;
        int i = 0;
        while(i <= r){
```

```
            if(nums[i] == 1) ++i;
            else if(nums[i] == 0){
                nums[i++] = nums[l];
                nums[l++] = 0;
            }else{
                nums[i] = nums[r];
                nums[r--] = 2;
            }
        }
    }
};
```

6. Stock Buy and Sell

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        if(prices.size() == 0) return 0;

        int ans = 0;

        int start = prices[0], end = prices[0];

        for(int i = 0; i < prices.size(); i++){
            if(prices[i] < start){
                //restart as session
                ans = max(ans, end - start);
                start = prices[i];
                end = prices[i];
            }else{
                //continue current session
```

```
            end = max(end, prices[i]);
        }
    }
    ans = max(ans, end - start);
    return ans;
  }
};
```

7. Rotate Matrix

```
class Solution {
public:
  void rotate(vector<vector<int>>& matrix) {
    for(int i=0;i<matrix.size()-1;i++) {
      for(int j=i+1;j<matrix.size();j++) {
        swap(matrix[i][j], matrix[j][i]);
      }
    }
    for(int i=0;i<matrix.size();i++) {
      int a=0, b=matrix.size()-1;
      while(a<b) {
        swap(matrix[i][a], matrix[i][b]);
        a++;
        b--;
      }
    }
  }
};
```

8. Merge Overlapping Subintervals

```
class Solution {
public:
  vector<vector<int>> merge(vector<vector<int>>& arr) {
```

```
 int n = arr.size(); // size of the array

//sort the given intervals:
sort(arr.begin(), arr.end());

vector<vector<int>> ans;

for (int i = 0; i < n; i++) { // select an interval:
    int start = arr[i][0];
    int end = arr[i][1];

    //Skip all the merged intervals:
    if (!ans.empty() && end <= ans.back()[1]) {
        continue;
    }

    //check the rest of the intervals:
    for (int j = i + 1; j < n; j++) {
        if (arr[j][0] <= end) {
            end = max(end, arr[j][1]);
        }
        else {
            break;
        }
    }
    ans.push_back({start, end});
}
return ans;

}
```

```
};
```

9. Merge Two sorted array without extra space

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        for (int j = 0, i = m; j<n; j++){
            nums1[i] = nums2[j];
            i++;
        }
        sort(nums1.begin(),nums1.end());
    }
};
```

10. Find the duplicate in an array of n+1 integers

```
class Solution {
public:
    int findDuplicate(vector<int>& nums) {
        int lo = 1, hi = nums.size() - 1;
        while(lo < hi)
        {
            int mid = (lo + hi) / 2;
            int count = 0;
            for(int num : nums){
                if(num <= mid)
                    ++count;
            }
            if(count <= mid){
                lo = mid + 1;
            }else{
                hi = mid;
            }
```

```
        }
        return lo;
    }
};
```

11. Repeat and missing number

```
#include <bits/stdc++.h>
pair<int,int> missingAndRepeating(vector<int> &arr, int n)
{
    //extreme better solution
    int hash[n+1] = {0};
    for(int i=0 ; i<n ; i++)
    {
        hash[arr[i]]++;
    }
    int repeating=-1, missing =-1;
    for(int i=1 ; i<=n ; i++)
    {
        if(hash[i] == 2)
        {
            repeating=i;
        }
        else if(hash[i] == 0)
        {
            missing=i;
        }
        if(repeating!=-1 && missing!=-1)
        {
            break;
        }
    }
```

```
    return {missing,repeating};
}
```

12. Inversion of Array

```
#include <bits/stdc++.h>
int merge(long long *arr , int low , int mid , int high)
{
  vector<int>temp;
  int left = low;
  int right = mid+1;
  int count=0;
  while(left <=mid && right <= high)
  {
    if(arr[left] <= arr[right])
    {
       temp.push_back(arr[left]);
       left++;
    }
    else
    {
       temp.push_back(arr[right]);
       count=count+(mid-left+1);  // count the possible sets
       right++;
    }
  }
  while(left <= mid) // if the left pointer exceed means add the remaining left  values
  {
     temp.push_back(arr[left]);
       left++;
  }
  while(right <= high)   // if the right pointer exceed means add the remaining values
```

```
  {
     temp.push_back(arr[right]);
     right++;
  }
  for(int i=low ; i<=high ; i++)
  {
     arr[i] = temp[i - low];
  }
  return count;
}
int mergesort(long long *arr, int low, int high)
{
    int count=0;
    if(low == high)return count;
   int mid=(low+high)/2;
   count = count + mergesort(arr , low , mid); //to get the 1 sorted array
   count = count + mergesort(arr , mid+1 , high);// to get the 2 sorted array
   count = count + merge(arr , low , mid , high); //finally merger the two sorted array
   return count;
}
long long getInversions(long long *arr, int n)
{
  return mergesort(arr, 0 , n-1);
}
```

13. Search in a 2D matrix

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& arr, int target) {
        int n=arr.size();
        int m=arr[0].size();
```

```
        int i=0;


        while(i<n)
        {
        if(target<=arr[i][m-1])
        {
           int j=0;
           while(j<m)
           {
              if(target==arr[i][j])
              return true;
              else if(target<arr[i][j])
              return false;
              else
              j++;
           }
           return false;
        }
        i++;
        }


    return false;
    }
};
```

14. Pow(x,n)

```
class Solution {
public:
    double myPow(double x, int n) {
        if(n==0){
            return 1;
```

```
        }

        if(n==1){
            return x;
        }

        if(n>0){
            if(n%2==0){
                return myPow(x*x, n/2);
            }
            else{
                return x*myPow(x*x, n/2);
            }
        }

        else{
            n= abs(n);
            if(n%2==0){
                return 1/myPow(x*x, n/2);
            }
            else{
                return 1/(x*myPow(x*x, n/2));
            }
        }
    }
};
```

15. Majority Element(>n/2 times)

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
```

```
        sort(nums.begin(), nums.end());
        int n = nums.size();
        return nums[n/2];
    }
};
```

16. Majority Element(>n/3 times)

```
class Solution {
public:
    vector<int> majorityElement(vector<int>& nums) {
        int cnt1 = 0, cnt2 = 0;
        int ele1 = INT_MIN;
        int ele2 = INT_MIN;
        for(int i=0; i<nums.size();i++) {
            if(cnt1 == 0 && ele2 != nums[i]) {
                cnt1 = 1;
                ele1 = nums[i];
            }
            else if(cnt2 == 0 && ele1 != nums[i]) {
                cnt2 = 1;
                ele2 = nums[i];
            }
            else if(nums[i] == ele1) cnt1++;
            else if(nums[i] == ele2) cnt2++;
            else {
                cnt1--;
                cnt2--;
            }
        }
        vector<int> result;
        cnt1=0, cnt2 = 0;
```

```
        for(int i=0;i<nums.size();i++) {
            if(nums[i] == ele1) cnt1++;
            if(nums[i] == ele2) cnt2++;
        }
        int mini = int(nums.size()/3)+1;
        if(cnt1 >= mini) result.push_back(ele1);
        if(cnt2 >= mini) result.push_back(ele2);
        return result;
    }
};
```

17. Grid Unique Paths

```
class Solution {
public:
    int uniquePaths(int m, int n) {
        //mXn blocks
        vector<vector<int>> dp(m,vector<int>(n,1));
        //If olny one block is present
        if(m==1&&n==1){return 1;}
        //Path to (i,j)block =path to(i-1,j)+path to(i,j-1)
        for(int i=1;i<m;i++){
            for(int j=1;j<n;j++){
                dp[i][j] = dp[i][j-1]+dp[i-1][j];
            }
        }
        return dp[m-1][n-1];
    }
};
```

18. Reverse Pairs

```
class Solution {
public:
```

```
    int reversePairs(vector<int>& nums) {
        int n = nums.size();
        long long reversePairsCount = 0;
        for(int i=0; i<n-1; i++){
            for(int j=i+1; j<n; j++){
                if(nums[i] > 2*(long long)nums[j]){
                    reversePairsCount++;
                }
            }
        }
        return reversePairsCount;
    }
};
```

19. 2Sum Problem

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        map<int, int> mpp;
        int n = nums.size();
        for(int i = 0; i<n; i++) {
            int num = nums[i];
            int more = target - num;
            if(mpp.find(more) != mpp.end()) return {mpp[more], i};
            mpp[num] = i;
        }
        return  {-1,-1};
    }
};
```

20. 4-Sum Problem

```
class Solution {
```

```
public:
    vector<vector<int>> fourSum(vector<int>& nums, int target) {
        int n = nums.size();
        sort(nums.begin(), nums.end());
        set<vector<int>> set;
        vector<vector<int>> output;
        for(int i=0; i<n-3; i++){
            for(int j=i+1; j<n-2; j++){
                for(int k=j+1; k<n-1; k++){
                    for(int l=k+1; l<n; l++){
                        if((long long)nums[i] + (long long)nums[j] + (long long)nums[k] +
                        (long long)nums[l] == target){
                            set.insert({nums[i], nums[j], nums[k], nums[l]});
                        }
                    }
                }
            }
        }
        for(auto it : set){
            output.push_back(it);
        }
        return output;
    }
};
```

21. Longest Consecutive Sequence

```
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        set<int> hashSet;
        for(int num : nums) {
```

```
        hashSet.insert(num);
      }
      int longest_Streak = 0;
      for(int num : nums) {
        if(!hashSet.count(num-1)) {
          int curr_num = num;
          int curr_Streak = 1;
          while(hashSet.count(curr_num+1)) {
            curr_num += 1;
            curr_Streak += 1;
          }
          longest_Streak = max(longest_Streak, curr_Streak);
        }
      }
      return longest_Streak;
    }
};
```

22. Count number of subarrays with given xor k

```
#include<bits/stdc++.h>
int subarraysWithSumK(vector < int > a, int k) {
    int xr=0;
    map<int,int>m;
    m[xr]++;
    int cnt=0;
    for(int i=0; i<a.size(); i++){
        xr=xr^a[i];
        int x = xr^k;
        cnt+=m[x];
        m[xr]++;
    }
```

```
    return cnt;
}
```

23. Longest SubString without repeating

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int n = s.length();
        int maxLength = 0;
        unordered_set<char> charSet;
        int left = 0;

        for (int right = 0; right < n; right++) {
            if (charSet.count(s[right]) == 0) {
                charSet.insert(s[right]);
                maxLength = max(maxLength, right - left + 1);
            } else {
                while (charSet.count(s[right])) {
                    charSet.erase(s[left]);
                    left++;
                }
                charSet.insert(s[right]);
            }
        }

        return maxLength;
    }
};
```

24. Reverse a linked list

```
class Solution {
public:
```

```
    ListNode* reverseList(ListNode* head) {
        ListNode* prev = NULL;
        ListNode* curr = head;
        while(curr){
            ListNode* next = curr->next;
            curr->next = prev;
            prev = curr;
            curr = next;
        }
        head = prev;
        return head;
    }
};
```

25. Find the middle of the linked list

```
class Solution {
    queue<ListNode*> q;
public:
    ListNode* middleNode(ListNode* head) {
        if( head != NULL )
        {
            while( head != NULL )
            {
                q.push(head);
                head = head->next;
            }
            int n = q.size()/2;
            while(n--)
                q.pop();
            return q.front();
        }
```

```
        return NULL;
    }
};
```

26. Merge two sorted linked list

```
class Solution {
public:
        ListNode* mergeTwoLists(ListNode* l1, ListNode* l2)
  {
                // if list1 happen to be NULL
                // we will simply return list2.
                if(l1 == NULL)
      {
                        return l2;
                }

                // if list2 happen to be NULL
                // we will simply return list1.
                if(l2 == NULL)
      {
                        return l1;
                }

                // if value pointend by l1 pointer is less than equal to value pointed by l2
pointer
                // we wall call recursively l1 -> next and whole l2 list.
                if(l1 -> val <= l2 -> val)
      {
                        l1 -> next = mergeTwoLists(l1 -> next, l2);
                        return l1;
                }
                // we will call recursive l1 whole list and l2 -> next
```

```
            else
        {
                    l2 -> next = mergeTwoLists(l1, l2 -> next);
                    return l2;
            }
        }
};
```

27. Remove N-th node from the back of the linked list

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int k) {
        ListNode *pre_slow, *slow, *fast;
        pre_slow=NULL;
        slow=fast=head;

        for(int i=0;i<k;i++) fast=fast->next;

        while(fast!=NULL){
            pre_slow=slow;
            slow=slow->next;
            fast=fast->next;
        }

        if(pre_slow==NULL){
            ListNode* new_head = head->next;
            delete head;
            return new_head;
        }

        pre_slow->next=slow->next;
```

```
        slow->next=NULL;
        delete slow;

        return head;
    }
};
```

28. Add two numbers as linked list

```
class Solution {
public:

    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        int val_1 , val_2, new_val, remainder;
        ListNode* not_null;
        ListNode* maybe_null;

        if(l1 == NULL && l2 == NULL){
            return NULL;
        }

        val_1 = (l1 == NULL? 0: l1->val);
        val_2 = (l2 == NULL? 0: l2->val);

        not_null = (l1 == NULL? l2: l1);
        maybe_null = ((l1 == not_null)? l2:l1);

        not_null->val= val_1+val_2;
        not_null->next = addTwoNumbers(not_null->next, (maybe_null ==
NULL?NULL:maybe_null->next));

        remainder = floor(not_null->val/10);
        not_null->val = (not_null->val%10);
```

```
        ListNode* this_node= not_null;

        while(remainder != 0 && this_node->next!=NULL){
            (this_node->next->val)+=remainder;
            remainder= floor(this_node->next->val/10);
            (this_node->next->val) = (this_node->next->val%10);
            this_node = this_node->next;
        }

        if(remainder!= 0 && this_node->next==NULL){
            this_node->next = new ListNode(remainder);
        }

        return not_null;
    }
};
```

29. Delete a given node when a node is given

```
class Solution {
public:
    void deleteNode(ListNode* node) {
        node->val = node->next->val; //copying the next node value
        node->next = node->next->next; // deleting the next node
    }
};
```

30. Find intersection point of Y linked list

```
class Solution {
public:
    int length(ListNode *head){
        int len = 0;
        while(head){
```

```
        len++;
        head = head->next;
    }
    return len;
}

ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
    if(!headA || !headB) return NULL;

    //step1
    int lenA = length(headA), lenB = length(headB);

    //step2
    if(lenA>lenB){
        while(lenA>lenB){
            headA = headA->next;
            lenA--;
        }
    }
    else if(lenA<lenB){
        while(lenA<lenB){
            headB = headB->next;
            lenB--;
        }
    }

    //step 3
    while(headA && headB){
        if(headA==headB) return headA;
        headA = headA->next;
```

```
            headB = headB->next;
        }
        return NULL;
    }
};
```

31. Detect a cycle in linked list

```
class Solution {
public:
    bool hasCycle(ListNode *head) {

                    // if head is NULL then return false;
        if(head == NULL)
            return false;

                    // making two pointers fast and slow and assignning them to head
        ListNode *fast = head;
        ListNode *slow = head;

                    // till fast and fast-> next not reaches NULL
                    // we will increment fast by 2 step and slow by 1 step
        while(fast != NULL && fast ->next != NULL)
        {
            fast = fast->next->next;
            slow = slow->next;


                        // At the point if fast and slow are at same address
                        // this means linked list has a cycle in it.
            if(fast == slow)
                return true;
```

```
        }

                // if traversal reaches to NULL this means no cycle.
        return false;
    }
};
```

32. Reverse a linked list in group of size k

```
class Solution {
public:
    int length (ListNode* head)
    {
        int len = 0;
        while(head != NULL)
        {
            len++;
            head = head -> next;
        }
        return len;
    }

    ListNode* reverseKGroup(ListNode* head, int k) {

        //head = 1

        int len = length(head); //Calculate length of LL
        if(len < k) //As mentioned in aue, if len < k don't reverse
        {
            return head;
        }
        int cnt = 0;
```

```
        ListNode* curr = head; //1 --- After 1st step, curr = 2
        ListNode* prev = NULL; //NULL
        ListNode* forward = NULL;

        while(curr != NULL && cnt < k)  //Reverseing 'k' nodes initially
        {
            forward = curr -> next; //2 --- 3
            curr -> next = prev; //1 -> 2 is broken and NULL <- 1 --- 2 -> 1
            prev = curr; //prev = 1 --- prev = 2
            curr = forward; // curr = 2 --- curr = 3
            cnt++;
        }
        if(forward != NULL)
        {
            head -> next = reverseKGroup(forward, k); //Recursively calling for remaining nodes
        }
        //I've stored it in head -> next bcz, head = 1 and I've coneected it with 4, head of the new
LL

        return prev; // return prev bcz, 2 is the head of our final LL and it is stored in prev
    }
};
```

33. Check if linked list is palindrome or not

```
class Solution {
public:
    bool isPalindrome(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head;
        while (fast != NULL && fast -> next != NULL) {
            slow = slow -> next;
            fast = fast -> next -> next;
```

```
        }
        if (fast != NULL && fast -> next == NULL) slow = slow -> next;
        ListNode* prv = NULL;
        ListNode* tmp = NULL;
        while (slow != NULL) {
            tmp = slow -> next;
            slow -> next = prv;
            prv = slow;
            slow = tmp;
        }
        slow = prv, fast = head;
        while (slow && fast) {
            if (slow -> val != fast -> val) return false;
            slow = slow -> next;
            fast = fast -> next;
        }
        return true;
    }
};
```

34. Find the starting point of loop in linked list

```
class Solution {
 public:
  ListNode* detectCycle(ListNode* head) {
    ListNode* slow = head;
    ListNode* fast = head;
    while (fast && fast->next) {
      slow = slow->next;
      fast = fast->next->next;
      if (slow == fast) {
        slow = head;
```

```
        while (slow != fast) {
          slow = slow->next;
          fast = fast->next;
        }
        return slow;
      }
    }
    return nullptr;
  }
};
```

35. Flattening of linked list

```
Node* merge(Node* head1, Node* head2)
{
    if(head1 == nullptr)
    {
        return head2;
    }
    if(head2 == nullptr)
    {
        return head1;
    }

    Node* ans = nullptr;

    if(head1->data <= head2->data)
    {
        ans = head1;
        ans->child = merge(head1->child , head2);
    }
    else
```

```
    {
        ans = head2;
        ans->child = merge(head1,head2->child);
    }

    return ans;
}

Node* flattenLinkedList(Node* head)
{
    if(head==nullptr || head->next == nullptr)
    {
        return head;
    }

        Node *newLL = flattenLinkedList(head->next);

        head->next = nullptr;

        Node *newHead = merge(newLL, head);


        return newHead;
}
```