**------------- ARRAY -------------**

***------------- ARRAY EASY -------------***

1. **To rotate an array to d places:**
   1. Left: reverse 0 to d – 1, reverse d to n – 1, reverse the total array
   2. RightL reverse first n – d, reverse rest d, reverse the total array
2. **Find 2nd largest:**
   1. Take 2 variable, largest and slargest. Make largest = a[0], slargest = INT\_MIN. Then traverse the array from index 1. If any number that is found greater than largest, then assign value of largest to slargest and value of array to largest, else if largest is strictly greater than the array value then assign the greater value between slargest and array to slargest.
3. **Remove duplicate from sorted array:**
   1. Take 2 pointers i=0, j=0. Traverse through the array till I < n. Then if any number found such that arr[j]!=arr[i] then increment j and assign the value at i at index j.
4. **Max consequetive 1’s after flipping k 0’s:**
   1. Take 2 pointers, i=0, j=0, maxConsOne=0, flipped=0. If flipped==k and arr[j]==0 then move the pointer i to the value after first 0, and decrement the value of flipped by 1. Else if arr[j]==0 then increment the value of flipped and assign the maximum of maxConsOne and j-i+1 to maxConsOne.
5. int i = 0, j = 0,  maxConsOne = 0, flipped = 0;
6. while(j < nums.size()){
7. if(nums[j] == 0 && flipped == k){
8. flipped--;
9. while(nums[i])
10. i++;
11. i++;
12. }else{
13. if(!nums[j])
14. flipped++;
15. maxConsOne = max(maxConsOne, j - i + 1);
16. j++;
17. }
18. }
19. return maxConsOne;
20. **Find the number that occure once in an array while all the other numbers occur twice.**
    1. Calculate xor of all numbers, the final value will be the answer. Because a number cancels out itself while doing XOR.

***------------- ARRAY MEDIUM -------------***

1. **2 sum problem when duplicate elements are present in the array.**
   1. While traversing through the array, first check if the map contains (target – nums[i]), if yes then return {i, map[target-nums[i]]}. It’ll support for duplicate element as we are checking the remaining value first and then we are inserting the value in map, so if 2 same element will be the answer then when the 2nd duplicate element will be encountered it first check that the remaing value(which is same as itself) is present in the map or not.
2. vector<int> twoSum(vector<int>& nums, int target) {
3. map<int, int> check;
4. for(int i = 0; i < nums.size(); i++){
5. int a = nums[i];
6. int remaining = target - nums[i];
7. if(check.find(remaining) != check.end())
8. return {i, check[remaining]};
9. check[a] = i;
10. }
11. return {0, 0};
12. }
    1. Another method is by taking 2 pointer.
13. vector<int> twoSum(vector<int>& nums, int target) {
14. vector<int> temp;
15. temp.insert(temp.end(), nums.begin(), nums.end());
16. sort(temp.begin(), temp.end());
17. int i = 0, j = temp.size() - 1;
18. while(i < j){
19. int sum = temp[i] + temp[j];
20. if(sum == target)
21. break;
22. else if(sum > target)
23. j--;
24. else i++;
25. }
26. vector<int>::iterator left = find(nums.begin(), nums.end(), temp[i]);
27. vector<int>::iterator right = find(nums.begin(), nums.end(), temp[j]);
28. if(left == right)
29. right = find(left + 1, nums.end(), temp[j]);
30. i = left - nums.begin();
31. j = right - nums.begin();
32. return {i, j};
33. }
34. **Find majority element which occurs more than half of size of array**
    1. As the number occurs more than half of size of array, so it’s frequency – sum of all frequencies of the other numbers > 0. So take 2 variable, count and candidate. Initialize count to be 0 initially. When count==0 then assign the candidate to the current value of array. If the candidate==num then count++ else count--. In this way all the rest element will cancel out their count and the majority element will be assigned to the candidte variable at the end.
35. int majorityElement(vector<int>& nums) {
36. int count = 0, candidate;
37. for(int num: nums){
38. if(count == 0) candidate = num;
39. if(num == candidate) count++;
40. else count--;
41. }
42. return candidate;
43. }
44. **Largest sum of a subarray in an array**
45. int maxSubArray(vector<int>& nums) {
46. int prev = INT\_MIN, sum = 0;
47. for(int num:nums){
48. sum += num;
49. prev = max(sum, prev);
50. if(sum < 0)
51. sum = 0;
52. }
53. return prev;
54. }
55. **Find the maximum among the sum of smallest ans 2nd smallest element of all possible subarray of an array. Take any subarray of an array of any length, find sum of it’s smallest and 2nd smallest element. Like this find sum of all possible subarrays and return the maximum of them.**
    1. As we need smallest and 2nd smallest so we don’t need an subarray having length more than 2. If we take a subarray of length 3, then it’s advantage should be of adding the first and last element, then the middle element must be greater for the first and last element to be smallest and 2nd smallest, so with this why shouldn’t we take the (first and middle) or (middle and last) which will give the maximum sum.
56. long long pairWithMaxSum(long long arr[], long long N){
57. long long sum = arr[0] + arr[1];
58. for(long long i = 1; i < N - 1; i++){
59. long long s = arr[i] + arr[i + 1];
60. sum = max(s, sum);
61. }
62. return sum;
63. }
64. **Maximum profit of buying and selling stock.**
65. int maxProfit(vector<int>& prices) {
66. int min\_price = INT\_MAX;
67. int max\_profit = INT\_MIN;
68. for(int price: prices){
69. min\_price = min(min\_price, price);
70. max\_profit = max(max\_profit, price - min\_price);
71. }
72. return max\_profit;
73. }
74. **Find the no. of subarrays of an given array, whose sum is equal to k.**
    1. Prefix method. Here we store the sums from 0 index of the array continuously till the last index of the array in prefix array. i.e. index 0 will be stored at index 0 of prefix array, sum of values from 0th index to 1st index will be stored ar prefix[1], sum of values from 0th index to 2nd index of the given array will be stored at prefix[2]… and so on.

Then we check for the sum. Traverse through prefix array, if any value of prefix is equal to k then increment the value of count. Now we’ll check for any other alternative of getting the sum equal to k, for this create a map to keep frequencies of the values in prefix array. Let s be a value in prefix array at a perticular index, now we’ll check wheather (s – k) is present or not in prefix array(to check this we have created the map), if present then how many times, then simply increment count to the frequency of (s – k).

**NOTE: As we have stored the contiguous sum of the array in prefix array, so if we subtract or a value at a perticular index of prefix array to another value at another index, then the result will be the sum of some consequitive element in the given array.**

1. int subarraySum(vector<int>& nums, int k) {
2. int count = 0, n = nums.size(), sum = 0;
3. vector<int> prefix;
5. for(int i = 0; i < n; i++){
6. sum += nums[i];
7. prefix.push\_back(sum);
8. }
9. unordered\_map<int, int> check;
10. for(int i = 0; i < n; i++){
11. if(prefix[i] == k)
12. count++;
13. if(check.find(prefix[i] - k) != check.end())
14. count += check[prefix[i] - k];
15. check[prefix[i]]++;
16. }
17. return count;
18. }
19. **Given an array, You have to find an array such that every element of this will be greater than or equal to all the elements of to the right of it.**
20. vector<int> leaders(int a[], int n){
21. int leader = a[n - 1];
22. vector<int> leader\_array;
23. leader\_array.push\_back(leader);
25. for(int i = n - 2; i >= 0; i--){
26. if(a[i] >= leader){
27. leader = a[i];
28. leader\_array.push\_back(leader);
29. }
30. }
31. reverse(leader\_array.begin(), leader\_array.end());
33. return leader\_array;
34. }
35. **Given an unsorted array. We have to find length of longest consequitive numbers that present in that array.**

Take a set and insert all elements of the array in it. Then traverse through the array, now check (num – 1) is present in the set or not, if not present then start counting for the next consequetive numbers of num in the set. As the loop continue only if the (num – 1) is not present in the set, so for all the consequetive elements present in the array, the loop will be skipped except once. So, this algorithm will be of O(N).

1. int longestConsecutive(vector<int>& nums) {
2. //You can also use check.find() method
3. set<int> check;
4. for(int num: nums)
5. check.insert(num);
6. int maxLen = 0;
7. for(int num: nums){
8. if(!check.count(num - 1)){
9. int currentNum = num;
10. int currentLen = 1;
11. while(check.count(currentNum + 1)){
12. currentNum++;
13. currentLen++;
14. }
15. maxLen = max(maxLen, currentLen);
16. }
17. }
18. return maxLen;
19. }
20. **Given matrix has some 0’s. Convert all the values of the rows and columns to 0 that passes through the 0’s.**
21. void setZeroes(vector<vector<int>>& matrix) {
22. int rows = matrix.size(), cols = matrix[0].size();
23. bool col0 = false; //Here j >= 1, see both the loops.
24. for(int i = 0; i < rows; i++){
25. if(matrix[i][0] == 0) col0 = true;
26. for(int j = 1; j < cols; j++){
27. if(matrix[i][j] == 0){
28. matrix[i][0] = 0;
29. matrix[0][j] = 0;
30. }
31. }
32. }
33. for(int i = rows - 1; i >= 0; i--){
34. for(int j = cols - 1; j >= 1; j--){
35. if(matrix[i][0] == 0 || matrix[0][j] == 0)
36. matrix[i][j] = 0;
37. }
38. if(col0) matrix[i][0] = 0;
39. }
40. }
41. **Rotate an array by 90 degrees clockwise.**
42. void rotate(vector<vector<int>>& matrix) {
43. int n = matrix.size();
44. for(int i = 0; i < n / 2; i++){
45. for(int j = i; j < n - i - 1; j++){
46. int temp = matrix[i][j];
47. matrix[i][j] = matrix[n - 1 - j][i];
48. matrix[n - 1 - j][i] = matrix[n - 1 - i][n - 1 - j];
49. matrix[n - 1 - i][n - 1 - j] = matrix[j][n - 1 - i];
50. matrix[j][n - 1 - i] = temp;
51. }
52. }
53. }
54. /\* --------------- OR --------------- \*/
55. /\* --------------- Transpose and rotate every row --------------- \*/
56. void rotate(vector<vector<int>>& matrix) {
57. int n = matrix.size();
58. for(int i = 0; i < n; i++){
59. for(int j = i; j < n; j++)
60. swap(matrix[i][j], matrix[j][i]);
61. }
62. for(int i = 0; i < n; i++)
63. reverse(matrix[i].begin(), matrix[i].end());
64. }
65. **Print a matrix in spiral order**
66. vector<int> spiralOrder(vector<vector<int>>& matrix) {
67. vector<int> spiral;
68. int rows = matrix.size(), cols = matrix[0].size();
69. int left = 0, right = cols - 1, top = 0, bottom = rows - 1;
70. while(top <= bottom && left <= right){
71. for(int j = left; j <= right; j++)
72. spiral.push\_back(matrix[top][j]);
73. top++;
74. for(int i = top; i <= bottom; i++)
75. spiral.push\_back(matrix[i][right]);
76. right--;
77. if(top <= bottom){
78. for(int j = right; j >= left; j--)
79. spiral.push\_back(matrix[bottom][j]);
80. bottom--;
81. }
82. if(left <= right){
83. for(int i = bottom; i >= top; i--)
84. spiral.push\_back(matrix[i][left]);
85. left++;
86. }
87. }
88. return spiral;
89. }

***------------- ARRAY HARD -------------***

1. **Majority element that occur more than n/3 times in an array, where n is the size of array.**
   1. As the occurance is more than n/3, so maximum 2 elements can be present in the array that occur more than n/3 times. So, we take 2 variables and their counters, i.e. n1, n2, c1, c2. First initializa n1 and n2 to -1. Now traverse through the loop:
      1. If num = n1 then increment c1
      2. Else if num = n2 then increment c2
      3. Else if c1 = 0 then increment c1 and n1 = num
      4. Else if c2 = 0 then increment c2 and n2 = num
      5. Else decrement both c1 and c2

*NOTE: As n1 contains the first element of the array, so for every unequal element it’ll go to n2 only if (****n1!=num, n2!=num and c1!=0****). At any instance, the problem that n1 and n2 will be equal, will not be occurred as n2 will be assigned a value when n1 is not equal to that.*

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

        int n1 = -1, n2 = -1, c1 = 0, c2 = 0;

        vector<int> elem;

        for(int num: nums){

            if(n1 == num) c1++;

            else if(n2 == num) c2++;

            else if(c1 == 0){

                c1++;

                n1 = num;

            }

            else if(c2 == 0){

                c2++;

                n2 = num;

            }

            else{

                c1--;

                c2--;

            }

        }

        c1 = c2 = 0;

        for(int num: nums){

            if(num == n1)   c1++;

            else if(num == n2) c2++;

        }

        if(c1 > nums.size() / 3) elem.push\_back(n1);

        if(c2 > nums.size() / 3) elem.push\_back(n2);

        return elem;

}

1. **Pascal Triangle**
2. vector<vector<int>> generate(int numRows) {
3. vector<vector<int>> pascal(numRows);
4. for(int i = 0; i < numRows; i++){
5. pascal[i].resize(i + 1);
6. pascal[i][0] = pascal[i][i] = 1;
7. for(int j = 1; j < i; j++)
8. pascal[i][j] = pascal[i - 1][j - 1] + pascal[i - 1][j];
9. }
10. return pascal;
11. }
12. **3 Sum problem. More than one triplet can be present whose sum is 0.**
    1. Similar to 2 sum problem. First sort the array. Traverse through the array from 0 to n-3, make the nums[i] as a point, then check the array from i+1 to n-1 using 2 pointer method in 2 sum.
       1. If the sum>0 then decrement right
       2. Else if the sum<0 then increment left
       3. Else push the 3 numbers to ans matrix, and increment left till nums[left]=nums[left+1] and decrement right till nums[right]=nums[right-1].
13. vector<vector<int>> threeSum(vector<int>& nums) {
14. sort(nums.begin(), nums.end());
15. int n = nums.size();
16. vector<vector<int>> ans;
17. for(int i = 0; i < n - 2; i++){
18. if(i == 0 || (nums[i] != nums[i - 1])){
19. int left = i + 1, right = n - 1;
20. while(left < right){
21. int sum = nums[left] + nums[right] + nums[i];
22. if(sum > 0)
23. right--;
24. else if(sum < 0)
25. left++;
26. else{
27. vector<int> temp;
28. temp.push\_back(nums[i]);
29. temp.push\_back(nums[left]);
30. temp.push\_back(nums[right]);
31. ans.push\_back(temp);
32. while(left < right && nums[left] == nums[left + 1])
33. left++;
34. while(left < right && nums[right]==nums[right - 1])
35. right--;
36. left++;right--;
37. }
38. }
39. }
40. }
41. return ans;
42. }
43. **4 sum. Given a target.**
    1. Similar to 3 sum, just one more loop.
44. vector<vector<int>> fourSum(vector<int>& nums, int target) {
45. sort(nums.begin(), nums.end());

        int n = nums.size();

1. vector<vector<int>> ans;
2. for(int i = 0; i < n - 3; i++){

            if(i == 0 || nums[i] != nums[i - 1]){

1. for(int j = i + 1; j < n - 2; j++){
2. if(j == i + 1 || nums[j] != nums[j - 1]){
3. int left = j + 1, right = n - 1;
4. long long sum1 = nums[i] + nums[j];
5. while(left < right){
6. long long sum2 = nums[left] + nums[right];
7. if(sum1 + sum2 == (long long)target){
8. vector<int>temp(4);
9. temp[0] = nums[i];
10. temp[1] = nums[j];
11. temp[2] = nums[left];
12. temp[3] = nums[right];
13. ans.push\_back(temp);
14. while(left < right && nums[left] == nums[left + 1]) left++;
15. while(left < right && nums[right] == nums[right - 1]) right--;
16. left++;
17. right--;
18. }
19. else if(sum1 + sum2 < (long long)target) left++;
20. else right--;
21. }
22. }
23. }
24. }
25. }
26. return ans;
27. }
28. **Given an array of positive and negative intezers. Find the length of longest subarray whose sum will be 0.**
    1. We’ll solve this problem using prefix sum method, For which we’ll take a hashmap which store the sum what we want. We’ll take a variable sum, which will be continuously updated to the sum of all element of the array from 0th index to current index while traversing the array.
       1. If sum=0 then assign the maxLen=i+1
       2. Else
          1. If sum is present in the map that the current sum is present or not, if present then it means the subarray starting from index at which we got sum to the current index has total sum = 0. I mean if we got sum = 25 at index 4, and now let the current index is 10 and the sum is again 25, then subarray from index 5 to index 10 has sum 0. So assign maxLen to max(maxLen, i-hashmap[sum])
          2. Else store sum in the hashmap i.e. hashmap[sum]=i
29. int maxLen(vector<int>&A, int n){
30. unordered\_map<int,int> check;
31. int sum = 0, maxLen = 0;
33. for(int i = 0; i < n; i++){
34. sum += A[i];
36. if(sum == 0){
37. maxLen = i + 1;
38. }
39. else{
40. if(check.find(sum) != check.end())
41. maxLen = max(maxLen, i - check[sum]);
42. else
43. check[sum] = i;
44. }
45. }
46. return maxLen;
47. }
48. **Merge overlapping interval**. [***Leetcode link***](https://leetcode.com/problems/merge-intervals/description/)
49. vector<vector<int>> merge(vector<vector<int>>& intervals) {
50. sort(intervals.begin(), intervals.end());
51. vector<vector<int>> ans;
52. int  rows = intervals.size();
54. for(int i = 0; i < rows; i++){
55. if(ans.empty() || ans.back()[1] < intervals[i][0])
56. ans.push\_back({intervals[i][0], intervals[i][1]});
57. else{
58. ans.back()[1] = max(ans.back()[1], intervals[i][1]);
59. }
60. }
61. return ans;
62. }
63. **Count inversion in an array i.e. no of pairs that can be possible where left element is greater than right element.** [***GFG link***](https://practice.geeksforgeeks.org/problems/inversion-of-array-1587115620/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=inversion-of-array)
    1. As we have to count inverted pairs, it’s simpler to use merge sorting method. Where we divide the array and merge, so while merging, if arr[i]>arr[j] then, all the element from the index i to mid are greater than arr[j], so increment inv\_count to mid – i.
64. long long merge(long long arr[], long long left, long long mid, long long right){
65. long long i = left, j = mid + 1, k = 0;
66. long long temp[right - left + 1];
67. long long inverse\_count = 0;
68. while(i <= mid && j <= right){
69. if(arr[i] <= arr[j])
70. temp[k++] = arr[i++];
71. else{
72. temp[k++] = arr[j++];
73. inverse\_count += (mid - i + 1);
74. }
75. }
76. while(i <= mid)
77. temp[k++] = arr[i++];
78. while(j <= right)
79. temp[k++] = arr[j++];
80. for(long long i = left; i <= right; i++)
81. arr[i] = temp[i - left];
82. return inverse\_count;
83. }
84. long long mergeSort(long long arr[], long long left, long long right){
85. long long inverse\_count = 0;
86. if(left < right){
87. int mid = (left + right) / 2;
89. inverse\_count += mergeSort(arr, left, mid);
90. inverse\_count += mergeSort(arr, mid + 1, right);
92. inverse\_count += merge(arr, left, mid, right);
93. }
94. return inverse\_count;
95. }
96. long long int inversionCount(long long arr[], long long N){
97. return mergeSort(arr, 0, N - 1);
98. }
99. **Find reverse pair, which means i < j and nums[i] > 2 \* nums[j].** [***Leetcode link***](https://leetcode.com/problems/reverse-pairs/description/)

Use merge sort approach. But in the merge function, before merging to sorted array, first calculate the number of reverse pair. Traverse through the first array, and inside the for loop check the second array in another for loop.

int merge(vector<int>&nums, int left, int mid, int right){

        int reverse\_count = 0, i, j;

        for(i = left, j = mid + 1; i <= mid; i++){

            while(j <= right && nums[i] > 2 \* (long long)nums[j])

                j++;

            reverse\_count += (j - mid - 1);

        }

        i = left;

        j = mid + 1;

        int temp[right - left + 1], k = 0;

        while(i <= mid && j <= right){

            if(nums[i] <= nums[j])

                temp[k++] = nums[i++];

            else

                temp[k++] = nums[j++];

        }

        while(i <= mid)

            temp[k++] = nums[i++];

        while(j <= right)

            temp[k++] = nums[j++];

        for(int i = left; i <= right; i++)

            nums[i] = temp[i - left];

        return reverse\_count;

    }

    int mergeSort(vector<int>&nums, int left, int right){

        int reverse\_count = 0;

        if(left < right){

            int mid = (left + right) / 2;

            reverse\_count += mergeSort(nums, left, mid);

            reverse\_count += mergeSort(nums, mid + 1, right);

            reverse\_count += merge(nums, left, mid, right);

        }

        return reverse\_count;

    }

    int reversePairs(vector<int>& nums) {

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

    }

1. **Maximum Produt subarray. Both positive and negative numbers are present in the array.** [***Leetcode link***](https://leetcode.com/problems/maximum-product-subarray/)

First check from left continuously till the end and assign the maximum to a variable maxLeft. Similarly do the same thing from end to start index and assign the maximum value to maxRight. Remember, if 0 is present in the array, then make product = 1 and count the product from next/previous index in forward/backward loop.

One problem will be occurred when 0 is present.

Consider a test case: -2 0 -1

Here if maxLeft will be -1 and maxRight will be -1. But the answer should be 0. So

If zero is present then return max(max(maxLeft, maxRight), 0)

Else return max(maxLeft, maxright)

int maxProduct(vector<int>& nums) {

        int maxLeft = INT\_MIN, maxRight = INT\_MIN;

        bool zeroPresent = false;

        int prod = 1;

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

            if(nums[i] == 0){

                zeroPresent = true;

                prod = 1;

                continue;

            }

            prod \*= nums[i];

            maxLeft = max(maxLeft, prod);

        }

        prod = 1;

        for(int i = nums.size() - 1; i >= 0; i--){

            if(nums[i] == 0){

                prod = 1;

                continue;

            }

            prod \*= nums[i];

            maxRight = max(maxRight, prod);

        }

        if(zeroPresent) return max(max(maxRight, maxLeft), 0);

        return max(maxLeft, maxRight);

    }