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

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

1. **To rotate an array to d places:**[***GFG Link***](https://practice.geeksforgeeks.org/problems/reversal-algorithm5340/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=reversal-algorithm)
   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:** [***GFG Link***](https://practice.geeksforgeeks.org/problems/second-largest3735/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=second-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:**[***GFG Link***](https://practice.geeksforgeeks.org/problems/remove-duplicate-elements-from-sorted-array/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=remove-duplicate-elements-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:** [***Leetcode Link***](https://leetcode.com/problems/max-consecutive-ones/)
   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.** [***Leetcode Link***](https://leetcode.com/problems/single-number/)
    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);

    }

**------------- BINARY SEARCH -------------**

***------------- BS ON 1-Dimensional ARRAY -------------***

1. **Search an element in the array using binary search.** [***Leetcode link***](https://leetcode.com/problems/binary-search/description/)
   1. Recursion method. Time: O(log N), Space: O(N)
2. int binSearch(vector<int>& nums, int target, int left,int right){
3. if(left <= right){
4. int mid = (left + right) / 2;
5. if(nums[mid] == target)
6. return mid;
7. else if(nums[mid] > target)
8. return binSearch(nums, target, left, mid - 1);
9. else
10. return binSearch(nums, target, mid + 1, right);
11. }
12. else return -1;
13. }
14. int search(vector<int>& nums, int target) {
15. return binSearch(nums, target, 0, nums.size() - 1);
16. }
    1. Loop method. Time: O(log N), Space: O(1)
17. int search(vector<int>& nums, int target) {
18. int start = 0, end = nums.size() - 1;
19. while(start <= end){
20. int mid = (start + end) / 2;
21. if(target == nums[mid])
22. return mid;
23. else if(target > nums[mid])
24. start = mid + 1;
25. else
26. end = mid - 1;
27. }
28. return -1;
29. }
30. **First and Last occurance of an element in an sorted array.**[***Leetcode Link***](https://leetcode.com/problems/find-first-and-last-position-of-element-in-sorted-array/description/)
31. vector<int> searchRange(vector<int>& nums, int target) {
32. int first = nums.size(), last = -1;
33. int start = 0, end = nums.size() - 1;
34. //last occurance
35. while(start <= end){
36. int mid = (start + end) / 2;
37. if(target == nums[mid]){
38. start = mid + 1;
39. last = max(last, mid);
40. }
41. else if(target < nums[mid])
42. end = mid - 1;
43. else
44. start = mid + 1;
45. }
46. //first occurance
47. start = 0, end = nums.size() - 1;
48. while(start <= end){
49. int mid = (start + end) / 2;
50. if(target == nums[mid]){
51. end = mid - 1;
52. first = min(first, mid);
53. }
54. else if(target < nums[mid])
55. end = mid - 1;
56. else
57. start = mid + 1;
58. }
59. if(first == nums.size())
60. first = -1;
61. return vector<int>({first, last});

}

1. **Find peak element, i.e. the element which is strictly greater than it’s neighbours. Assume the -1th and nth index element be INT\_MIN, where n is size of the array. If there is more than one element, then return any of them.**

Simply the question is to find out any of local maxima. If we represent it as a graph, for local maxima, there ara following conditions can be arrived.

Mid = (Start + End)/2;

* + 1. The curve is a line with positive slope, in this case A[mid]>A[mid-1] and A[mid]<A[mid+1]

So start=mid+1;

* + 1. The curve is a line with negative slope, then A[mid]<A[mid-1] and A[mid]>A[mid+1]

So end=mid-1;

* + 1. The start and mid point make a curve like **inverted V**, so here A[mid]>A[mid-1], We know there is a maxima in between the start and mid point(If there might be more than one maxima also like **WW** shape), but we can also see that there must be atleast one maxima between mid and end, if there is only this positive slope line also then the end point will be the maxima,

So start=mid+1;

* + 1. The start and mid point make **V** shape, so here A[mid]>A[mid-1], We know there is no maxima present in between the start and mid point, if this will be like **VVV**, here A[mid]>A[mid-1] and 2 maxima are present, but we can notice that in this case, There must be one maxima in between mid to end, if no local maxima are present then also the end point is a maxima as it’s a line with increasing slope.

So the conclusion is, if A[mid]>A[mid-1] then start=mid+1

Else if A[mid]<A[mid-1] then end=mid-1;

int findPeakElement(vector<int>& nums) {

        int n = nums.size();

        if(n == 1)

            return 0;

        int start = 0, end = n - 1;

        while(start <= end){

            int mid = (start + end) / 2;

            bool cond1 = (mid == n - 1) && nums[mid] > nums[n - 2];

            bool cond2 = (mid == 0) && nums[mid] > nums[mid + 1];

            bool cond3 = mid != 0 && nums[mid] > nums[mid - 1] && mid != n - 1 && nums[mid] > nums[mid + 1];

            if(cond1 || cond2 || cond3)

                return mid;

            if(mid == 0 || nums[mid] > nums[mid - 1])

                start = mid + 1;

            else

                end = mid - 1;

        }

        return -1;

    }

1. **Find the target element in an array which is left rotated at an index (The index is not given). Time complexity should be O(log n).**

Similar to previous, draw a graph and analyse.

1. int search(vector<int>& nums, int target) {
2. int n = nums.size();
3. int start = 0, end = n - 1;
4. while(start <= end){
5. int mid = (start + end) / 2;
6. if(nums[mid] == target)
7. return mid;
8. if(nums[start] <= nums[mid]){
9. if(target < nums[mid] && target >= nums[start])
10. end = mid - 1;
11. else start = mid + 1;
12. }
13. else{
14. if(target > nums[mid] && target <= nums[end])
15. start = mid + 1;
16. else end = mid - 1;
17. }
18. // if(nums[mid] > nums[end]){
19. //     if(nums[mid] > target && target >= nums[start])
20. //         end = mid - 1;
21. //     else
22. //         start = mid + 1;
23. // }
24. // else if(nums[mid] < nums[start]){
25. //     if(nums[mid] < target && target <= nums[end])
26. //         start = mid + 1;
27. //     else
28. //         end = mid - 1;
29. // }
30. // else{
31. //     if(nums[mid] < target)
32. //         start = mid + 1;
33. //     else
34. //         end = mid - 1;
35. // }
36. }
37. return -1;
38. }
39. **Above question but array contains duplicate elements.**
40. bool search(vector<int>& nums, int target) {
41. int n = nums.size();
42. int start = 0, end = n - 1;
43. while(start <= end){
44. int mid = (start + end) / 2;
45. if(nums[mid] == target)
46. return true;
48. if(nums[start] == nums[mid] && nums[start] == nums[end]){
49. start++;
50. end--;
51. }
52. else if(nums[mid] > nums[end]){
53. if(nums[mid] > target && target >= nums[start])
54. end = mid - 1;
55. else
56. start = mid + 1;
57. }
58. else if(nums[mid] < nums[start]){
59. if(nums[mid] < target && target <= nums[end])
60. start = mid + 1;
61. else
62. end = mid - 1;
63. }
64. else{
65. if(nums[mid] < target)
66. start = mid + 1;
67. else
68. end = mid - 1;
69. }
70. }
71. return false;
72. }
73. **Find minimum in a rotated sorted array.**[***Leetcode Link***](https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/description/)
74. int findMin(vector<int>& nums) {
75. int start = 0, end = nums.size() - 1, min\_val = INT\_MAX;
76. while(start <= end){
77. if(nums[start] < nums[end]){
78. min\_val = min(min\_val, nums[start]);
79. break;
80. }
81. int mid = (start + end) / 2;
82. if(nums[start] <= nums[mid]){
83. min\_val = min(min\_val, nums[start]);
84. start = mid + 1;
85. }
86. else{
87. end = mid - 1;
88. min\_val = min(min\_val, nums[mid]);
89. }
90. }
91. return min\_val;
92. // int start = 0, end = nums.size() - 1;
93. // if(nums.size() == 1)
94. //     return nums[0];
95. // else if(nums[0] < nums[nums.size() - 1])
96. //     return nums[0];
97. // else if(nums[0] > nums[1])
98. //     return nums[1];
99. // else if(nums[nums.size() - 1] < nums[nums.size() - 2])
100. //     return nums[end];
101. // while(start <= end){
102. //     int mid = (start + end) / 2;
104. //     if(mid != 0 && mid != nums.size() - 1){
105. //         if(nums[mid] < nums[mid - 1] && nums[mid] < nums[mid + 1])
106. //             return nums[mid];
107. //         else if(nums[mid] > nums[0])
108. //             start = mid + 1;
109. //         else if(nums[mid] < nums[0])
110. //             end = mid - 1;
111. //     }
112. // }
113. // return -1;
114. }
115. **Find the single element in an sorted array, where all the elements occur twice except one element.** [***Leetcode Link***](https://leetcode.com/problems/single-element-in-a-sorted-array/description/)***]***
116. int singleNonDuplicate(vector<int>& nums) {
117. int n = nums.size();
118. if(n == 1)
119. return nums[0];
120. if(nums[0] != nums[1])
121. return nums[0];
122. if(nums[n - 1] != nums[n - 2])
123. return nums[n - 1];
124. int start = 0, end = n - 1;
125. while(start <= end){
126. int mid = (start + end) / 2;
128. if(mid == 0)
129. start = mid + 1;
130. else if(nums[mid] == nums[mid + 1]){
131. if(mid % 2 == 0)
132. start = mid + 2;
133. else
134. end = mid - 1;
135. }
136. else if(nums[mid] == nums[mid - 1]){
137. if(mid % 2 == 0)
138. end = mid - 2;
139. else
140. start = mid + 1;
141. }else
142. return nums[mid];
143. }
144. return -1;
145. }
146. **An sorted array is rotated at an index, let the value present at that index is K. Find the current index of K in the rotated array.**
147. int findKRotation(int arr[], int n) {
148. int start = 0, end = n - 1, mid;
149. while(start <= end){
150. mid = (start + end) / 2;
152. if(arr[start] <= arr[end])
153. return start;
154. else if(arr[mid] > arr[start])
155. start = mid + 1;
156. else if(arr[mid] < arr[end]){
157. if(arr[mid] < arr[mid - 1])
158. return mid;
159. else
160. end = mid - 1;
161. }
162. else if(arr[mid] >= arr[end])
163. return end;
165. }
166. }

**--------BS ON 2-D ARRAY--------**

1. **Find an element in 2-D array using binary search (O(log (m \* n))).**[***Leetcode Link***](https://leetcode.com/problems/search-a-2d-matrix/description/)
2. bool searchMatrix(vector<vector<int>>& matrix, int target) {
3. int rows = matrix.size(), cols = matrix[0].size();
4. int first = 0, last = (rows \* cols) - 1;
5. while(first <= last){
6. int mid = (first + last) / 2;
7. if(matrix[mid / cols][mid % cols] == target)
8. return true;
9. else if(matrix[mid / cols][mid % cols] < target)
10. first = mid + 1;
11. else
12. last = mid - 1;
14. }
15. return false;
16. }

bool searchMatrix(vector<vector<int>>& matrix, int target) {

        int row = matrix.size(), col = matrix[0].size();

        int top = 0, bottom = row - 1;

        while(top <= bottom){

            int mid = (top + bottom) / 2;

            if(target < matrix[mid][0])

                bottom = mid - 1;

            else if(target > matrix[mid][0])

                top = mid + 1;

            else

                return true;

        }

        int r = top - 1;

        int left = 0, right = col - 1;

        while(r >= 0 && left <= right){

            int mid = (left + right) / 2;

            if(matrix[r][mid] < target)

                left = mid + 1;

            else if(matrix[r][mid] > target)

                right = mid - 1;

            else

                return true;

        }

        return false;

    }

1. **Find peak element in a 2D array. (Peak element is what is greater than it’s left, right, top, bottom element).** [***Leetcode Link***](https://leetcode.com/problems/find-a-peak-element-ii/description/)

Steps:

Find the mid column.

Find the row that contain max element at that mid column.

If the max element is greater than it’s right as well as left element then return the index, else if the mid element is less than it’s left element then check in the left part (right=mid-1) else check in the right part (left=right+1).

vector<int> findPeakGrid(vector<vector<int>>& mat) {

        int n = mat.size(), m = mat[0].size();

        int left = 0, right = m - 1;

        while(left <= right){

            int mid = (left + right) / 2;

            int max\_row = 0;

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

                if(mat[i][mid] > mat[max\_row][mid])

                    max\_row = i;

            if((mid == 0 || mat[max\_row][mid] > mat[max\_row][mid - 1]) && (mid == m - 1 || mat[max\_row][mid] > mat[max\_row][mid + 1]))

                return vector<int>({max\_row, mid});

                else if(mid > 0 && mat[max\_row][mid] < mat[max\_row][mid - 1])

                    right = mid - 1;

                else

                    left = mid + 1;

        }

        return vector<int>({0, 0});

    }

1. **Find median of a row wise sorted matrix.**[***GFG Link***](https://practice.geeksforgeeks.org/problems/median-in-a-row-wise-sorted-matrix1527/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=median-in-a-row-wise-sorted-matrix)

First take 2 points, **start=1, end=10^9**. Then find the mid of these start and end. Now check how many element present in the matrix which are less than or equal to mid. If the **count <= (R\*C)/2** then **start=mid+1** otherwise **end=mid-1**. Then return start.

To calculate the no. of element those are less than or equal to the mid, we use binary search approach, as the elements in every row are sorted. Simple method to find the no. of elements that are less than or equal to a number X is to find the index of the element which is greater than X but it’s left index element should be less than or equal to X.

    int countLessThanMid(vector<int>&arr, int mid){

        int left = 0, right = arr.size() - 1;

        while(left <= right){

            int md = (left + right) / 2;

            if(arr[md] <= mid)

                left = md + 1;

            else

                right = md - 1;

        }

        return left;

    }

    int median(vector<vector<int>> &matrix, int R, int C){

        int start = 1, end = 1e9;

        int count = 0;

        while(start <= end){

            int mid = (start + end) / 2;

            count = 0;

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

                 count += countLessThanMid(matrix[i], mid);

            if(count <= (R \* C) / 2)

                start = mid + 1;

            else

                end = mid - 1;

        }

        return start;

    }

**--------Find Answe by BS search Space--------**

1. **Remove the outer parenthesis.** [***Leetcode Link***](https://leetcode.com/problems/remove-outermost-parentheses/description/)
2. string removeOuterParentheses(string s) {
3. int temp = 0;
4. string str;
5. for(char c: s){
6. if(c == '(') temp++;
7. else if(c == ')') temp--;
8. if(temp == 0)
9. continue;
10. if(temp > 1 || c != '(')
11. str.push\_back(c);
12. }
13. return str;
14. }
15. **Reverse a words of a string.**[***Leetcode Link***](https://leetcode.com/problems/reverse-words-in-a-string/description/)
16. string reverseWords(string s) {
17. int i = 0;
18. string ans = "";
19. string temp = "";
20. while(i < s.length()){
21. if(s[i] != ' '){
22. temp += s[i];
23. }else if(temp != ""){
24. if(ans != "")
25. ans = temp + " " + ans;
26. else
27. ans = temp;
28. temp = "";
29. }
30. i++;
31. }
32. if(temp != ""){
33. if(ans != "")
34. ans = temp + " " + ans;
35. else
36. ans = temp;
37. }
38. return ans;
39. }

**--------Linked List--------**

1. **Delete a node named *node* from a linked list where head is not known.**

node -> data = node -> next -> data;

node -> next = node -> next -> next;

1. **Delete an element of a linked list at a perticular index(index start from 1).** [***GFG Link***](https://practice.geeksforgeeks.org/problems/delete-node-in-doubly-linked-list/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=delete-node-in-doubly-linked-list)
2. Node\* deleteNode(Node \*head, int x)
3. {
4. if(x == 1){
5. head -> next -> prev = NULL;
6. return head -> next;
7. }
8. Node\* ptr = head;
9. int i = 1;
10. while(i + 1 < x && ptr != NULL){
11. i++;
12. ptr = ptr -> next;
13. }
14. if(ptr -> next -> next != NULL)
15. ptr -> next -> next -> prev = ptr -> next -> prev;
16. ptr -> next = ptr -> next -> next;
17. return head;
18. }
19. **Reverse a doubly linked list in O(1) space complexity.**

Consider an example 1 -> 2 -> 3 -> 4. Here prev of 1 is NULL and next of 4 is NULL.

After reversing the sequence will be like 4 -> 3 -> 2 -> 1. Here prev of 4 is NULL and next of 1 is NULL.

So we can see, for every element, their prev and next is swapped. So, we’ll iterate through the linked list and swap the prev and next of each element. Let temp be a variable used during swapping and it contains the prev element of current element, then at the end of the loop, i.e. during the swapping the prev of next of the element 4, the temp will be pointing to 3. And after reversing the head of the linked list should be the address that contains the node whose data is 4. And this will be prev of 3, as temp contain the address of 3, so temp->prev will be the head.

Again if the liked list is of length 1 or 0, then the temp will be NULL, so return the head itself in this case.

Node\* reverseDLL(Node \* head){

Node \*curr = head, \*temp = head -> prev;

        while(curr != NULL){

            temp = curr -> prev;

            curr -> prev = curr -> next;

            curr -> next = temp;

            curr = curr -> prev;

        }

        if(temp != NULL)

            return temp -> prev;

        return head;

        // Node\* ptr = head;

        // vector<int> arr;

        // while(ptr != NULL){

        //     arr.push\_back(ptr -> data);

        //     ptr = ptr -> next;

        // }

        // ptr = head;

        // int i = arr.size() - 1;

        // while(ptr != NULL){

        //     ptr -> data = arr[i];

        //     i--;

        //     ptr = ptr -> next;

        // }

        // return head;

    }

1. **Reverse a singly linked list with O(1) space complexity.** [**Leetcode Link**](https://leetcode.com/problems/reverse-linked-list/description/)

Similar to doubly linked list reverse. But here we have to take another pointer next.

Example: Consider an linked list containing 1 -> 2 -> 3.

After reversing it’ll look like 3 -> 2 -> 1.

Curr will point to the current pointer, i.e. 1 (initially) and prev is initially assigned to NULL.

Now next = curr->next (to keep the address of next), curr->next will be assign to prev. Now for the next iteration, curr of now will be that’s previous, so prev=curr and as we’ll go to 2, which is stored in next, so curr=next.

ListNode\* reverseList(ListNode\* head) {

        ListNode\* prev = NULL, \*curr = head, \*next = NULL;

        while(curr != NULL){

            next = curr -> next;

            curr -> next = prev;

            prev = curr;

            curr = next;

        }

        return prev;

    }

1. **Detect if the linked list contains any circle or not.** [**Leetcode Link**](https://leetcode.com/problems/linked-list-cycle/description/)

As duplicate numbers may be present in the linked list, so if 2 numbers are same then we can’t say that the linked list contains any circle or not, so we make a unorderd set that contains the addresses, as the addrsses can’t be duplicate, so if any address is found in the hashSet then the linked list contains a circle oterwise not. The sample code taking hashSet:

    bool hasCycle(ListNode \*head) {

        ListNode \*ptr = head;

        unordered\_set<ListNode\*> hashSet;

        while(ptr != NULL){

            if(hashSet.find(ptr) != hashSet.end())

                return true;

            hashSet.insert(ptr);

            ptr = ptr -> next;

        }

        return false;

    }

It’s a 2 pointer approach. If head is null then return false. Otherwise declare 2 pointer fast and slow. slow pointer slowly moves towards right by one places, fast moves fast i.e. by 2 places a at once. When slow will be equal to fast, return true, if head -> next = null or head->next->next=null then return false.

bool hasCycle(ListNode \*head) {

        if(head == NULL) return false;

        ListNode \*slow, \*fast;

        slow = fast = head;

        while(fast -> next != NULL && fast -> next -> next != NULL){

            fast = fast -> next -> next;

            slow = slow -> next;

            if(slow == fast) return true;

        }

        return false;

    }