Part-1

Array Common Problem

DSA Problem for Placement

amazon Microsoft

Coding!

Array:

- Missing number in array
- Rotate Array
- Maximum Subarray (Kadane's Algo.)
- Reverse the Array
- Max Circular Subarray Sum
- Maximum Product of subarray
- 3Sum
- Trapping Rain Water
- Best time to Buy and Sell Stock
- Merge Intervals

1. Missing Number

268. Missing Number

Easy 6452 2911 Add to List Share

Given an array `nums` containing `n` distinct numbers in the range `[0, n]`, return *the only number in the range that is missing from the array.*

Example 1:

```
Input: nums = [3,0,1]
Output: 2
Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.
```

Example 2:

```
Input: nums = [0,1]
Output: 2
Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the range since it does not appear in nums.
```

Approach:-

1. Make an array temp[ ] & initialized with true at all indicies.
2. Iterating over a given array and marking temp array indicies false w.r.t given array element

means :- temp [nums [i]] = false;

3. Now, Iterate in the temp [ ] from i=0, BREAK the loop when you find check[i] = True and store that i in the ans variable.

```cpp
class Solution {
public:
    int missingNumber(vector<int>& nums) {
        int N=1e6+2;
        bool temp[N];
        int ans=-1;
        for(int i=0;i<N;i++)

        {
            temp[i] = true;
        }

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

            temp[nums[i]] = false;
        }

        for(int i=0;i<=nums.size();i++)
        {
            if(temp[i]==true)
            {
                ans = i;
                   break;

            }
        }
       return ans;
    }
};
```

Dry Run :-

[3, 0, 1]

Approach :- 1

temp = [T, T, T ---]

i = 0,

temp[nums[0]] = false

tem[3] = false

[T, T, T, F, T ---]

i = 1,

temp[nums[1]] = false

[F, T, T F, T ---]

i = 2,

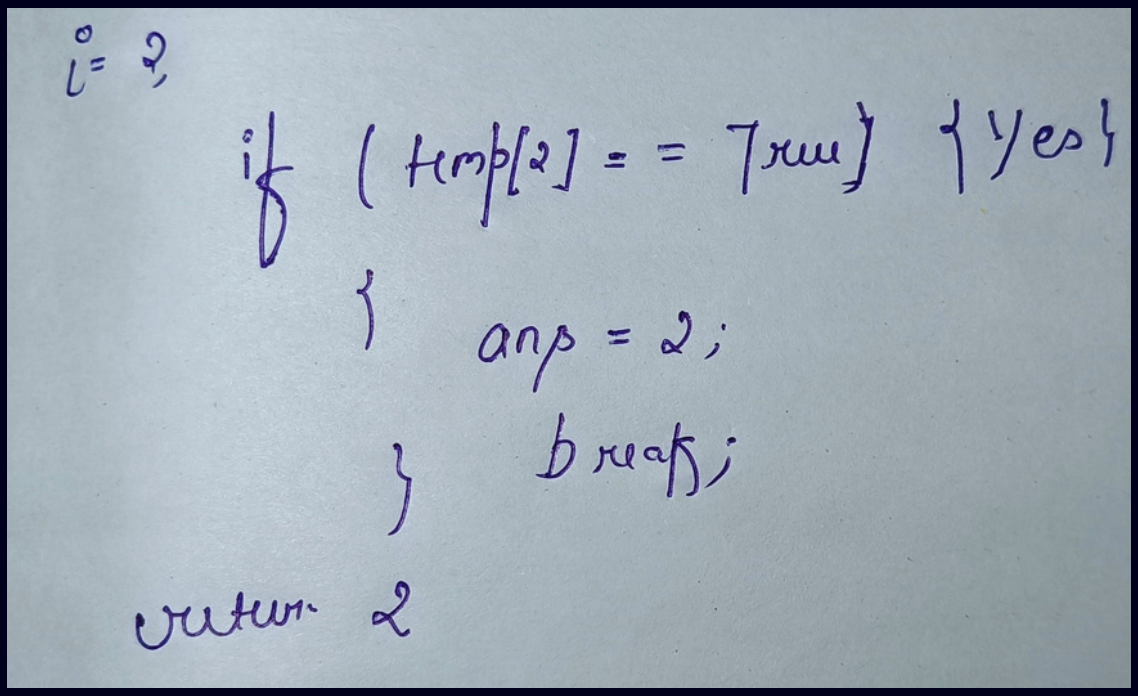
temp[nums[2]] = false

[F, F, T, F, T ---]

Now, loop 3

i = 0, if(temp[0] == true] (Also, i = 1 not true)

No than chuck for i = 1 and 2



Approach-2:-

1. Firstly , sort an given array by using sort function.
2. Check every element of given array w.r.t loop iteration, if it is found not equal than return value which is not matched.

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

        sort(nums.begin(),nums.end());

        int ans;

        for(int i=0;i<nums.size();i++)
        {
            if(i!=nums[i])
            {
                ans = i;
                break;
            }
        }
        return ans;
    }
};
```

Time complexity :- O(nlogn)

Dry Run :-

Approach :- 2

[3, 0, 1]

Sort this by using sort function

[0, 1, 3]

i=0.

if (i != nums[i])

0 != 0 (NO)

i=1.

if (1 != 1)

No

i=2,

if (2 != 3)

Yes

ans = 2

break

return 2.,

2

2. Rotate Array

189. Rotate Array

Medium 10105 1343 Add to List Share

Given an array, rotate the array to the right by `k` steps, where `k` is non-negative.

Example 1:

```
Input: nums = [1,2,3,4,5,6,7], k = 3
Output: [5,6,7,1,2,3,4]
Explanation:
rotate 1 steps to the right: [7,1,2,3,4,5,6]
rotate 2 steps to the right: [6,7,1,2,3,4,5]
rotate 3 steps to the right: [5,6,7,1,2,3,4]
```

Example 2:

```
Input: nums = [-1,-100,3,99], k = 2
Output: [3,99,-1,-100]
Explanation:
rotate 1 steps to the right: [99,-1,-100,3]
rotate 2 steps to the right: [3,99,-1,-100]
```

Approach:-

Brute Force :-

1.Iterate k using while loop :- while(k--)

2. Increment the position of every given element to 1 and place the last element to first position by using temp variable

```cpp
class Solution {
public:
    void rotate(vector<int>& nums, int k) {

        int n = nums.size();

        while(k--){
            int temp = nums[n-1];
            for(int i = n-2; i>= 0 ;i--)
            {
                nums[i+1] = nums[i];
            }
            nums[0] = temp;
        }
    }
};
```

This is not submitted on leetcode due to time complexity.

Dry Run :-

Approach :- 1 (Brute force)

[1, 2, 3, 4, 5, 6, 7] , K = 3
 0 1 2 3 4 5 6

n = 7
K = 3

while (k--)
{
  temp = 7
  i = 5,
    nums[6] = 6
  i = 4
    nums[5] = 5
  i = 3,
    nums[4] = 4
  i = 2,
    nums[3] = 3
  i = 1,
    nums[2] = 2
  i = 0
    nums[1] = 1
}

nums[0] = 7

[7, 1, 2, 3, 4, 5, 6] (Repeat 3 times this.)

Approach-2:-

1. In this approach we reverse the array 3 times but from diffrent position to diffrent position

2. k =k % nums.size( );
It means , if in any case k is greater than the size of array , So if we have K = 9 for array size 4 we're really just rotating it by 1 step.

```
class Solution {
public:
    void rotate(vector<int>& nums, int k) {

        k = k%nums.size();
        reverse(nums.begin(), nums.end());
        reverse(nums.begin(), nums.begin() + k);
        reverse(nums.begin() + k, nums.end());
    }
};
```

Dry Run :-

nums = [1, 2, 3, 4, 5, 6, 7] , K = 3

K = K % nums.size( );

K = 3 % 7 = 3

Now,

reverse(nums.begin( ), nums.end( ));
(0, 6)

[7, 6, 5, 4, 3, 2, 1]

reverse(nums.begin( ), nums.begin( ) + K);
(0, 3)

[5, 6, 7, 4, 3, 2, 1]

reverse(nums.begin( ) + K, nums.end( ));

[5, 6, 7, 1, 2, 3, 4]

3. Maximum Subarray

53. Maximum Subarray

Medium 22481 1104 Add to List Share

Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the largest sum and return *its sum*.

A **subarray** is a **contiguous** part of an array.

Example 1:

```
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: [4,-1,2,1] has the largest sum = 6.
```

Example 2:

```
Input: nums = [1]
Output: 1
```

Example 3:

```
Input: nums = [5,4,-1,7,8]
Output: 23
```

Approach:-

Brute Force :-

1. Break the array into subarray by for loop.

2. Calculate the sum of subarray (k) and find the maximum of that.

```cpp
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
       int  n = nums.size();
   int  maxsum = INT_MIN;
        for(int i=0;i<n;i++)
        {
            for(int j = i;j<n;j++)

            {
                     int sum=0;
                for(int k=i;k<j;k++)
                {
                    sum = sum+nums[k];
                }
                maxsum = max(maxsum,sum);
            }
        }

        return maxsum;
    }
};
```

Time Complexity :- O(n^3)

Approach-2:- (Kadane's Algo.)

1. Calculate the sum of array from starting

2. If it is neagative then discard the current subarray and start the new sum

```cpp
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
       int  n = nums.size();
        int sum=0;
        int maxsum = INT_MIN;
        for(int i=0;i<n;i++)
        {
            sum = sum+nums[i];
           if(maxsum < sum)
           {
               maxsum = sum;
           }
            if(sum<0)
            {
                sum =0;
            }
        }
        return maxsum;

    }
};
```

Time Complexity :- O(n)

4. Reverse the array

Write a program to reverse an array or string

Difficulty Level : Basic • Last Updated : 08 Sep, 2020

Given an array (or string), the task is to reverse the array/string.

Examples :

```
Input  : arr[] = {1, 2, 3}
Output : arr[] = {3, 2, 1}

Input :  arr[] = {4, 5, 1, 2}
Output : arr[] = {2, 1, 5, 4}
```

Approach:-

1. *Intialize array as start and end , Swap element of start and end*

2. *After swapping start increment by 1 and end decrement by 1*

```
void Array(int arr[], int n)
{
    int start =0;
    int end = n-1;
    while (start < end)
    {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}

int main()
{
    ----------
        -------
        ------------
}
```

Time Complexity :- O(n)

5.Maximum Circular Subarray

918. Maximum Sum Circular Subarray

Medium 3424 152 Add to List Share

Given a **circular integer array** `nums` of length `n`, return *the maximum possible sum of a non-empty* ***subarray*** *of* `nums`.

A **circular array** means the end of the array connects to the beginning of the array. Formally, the next element of `nums[i]` is `nums[(i + 1) % n]` and the previous element of `nums[i]` is `nums[(i - 1 + n) % n]`.

A **subarray** may only include each element of the fixed buffer `nums` at most once. Formally, for a subarray `nums[i], nums[i + 1], ..., nums[j]`, there does not exist `i <= k1, k2 <= j` with `k1 % n == k2 % n`.

Example 1:

```
Input: nums = [1,-2,3,-2]
Output: 3
Explanation: Subarray [3] has maximum sum 3.
```

Example 2:

```
Input: nums = [5,-3,5]
Output: 10
Explanation: Subarray [5,5] has maximum sum 5 + 5 = 10.
```

Approach:-

1. If the maximum sum is not in the circular array than simply we get the maxsum by kadane algorithm

2. If the maximum sum is in the circular array than we multiply the array by -1 to get the min array and by adding kadane algo. in this get the maximum sum subarray.

```cpp
class Solution {
public:

  int kadane(vector<int>& nums) {
        int maxsum=INT_MIN;
        int sum=0;
        for(int i=0;i<nums.size();i++)
        {
            sum+=nums[i];
            maxsum=max(maxsum,sum);
            if(sum<0)sum=0;
        }
        return maxsum;
    }
    int maxSubarraySumCircular(vector<int>& nums) {

        int Normal=kadane(nums);
        int sum=0;
        for(int i=0;i<nums.size();i++)
        {
            sum+=nums[i];
            nums[i]=-nums[i];
        }

        int res = sum+ kadane(nums);

        if(Normal<0)
            return Normal;

        return max(Normal,res);
    }
};
```

Time Complexity :- O(n)

6.Maximum Product Subarray

918. Maximum Sum Circular Subarray

Medium 3424 152 Add to List Share

Given a **circular integer array** `nums` of length `n`, return *the maximum possible sum of a non-empty* ***subarray*** *of* `nums`.

A **circular array** means the end of the array connects to the beginning of the array. Formally, the next element of `nums[i]` is `nums[(i + 1) % n]` and the previous element of `nums[i]` is `nums[(i - 1 + n) % n]`.

A **subarray** may only include each element of the fixed buffer `nums` at most once. Formally, for a subarray `nums[i], nums[i + 1], ..., nums[j]`, there does not exist `i <= k1`, `k2 <= j` with `k1 % n == k2 % n`.

Example 1:

```
Input: nums = [1,-2,3,-2]
Output: 3
Explanation: Subarray [3] has maximum sum 3.
```

Example 2:

```
Input: nums = [5,-3,5]
Output: 10
Explanation: Subarray [5,5] has maximum sum 5 + 5 = 10.
```

Approach:-

1. In this we can take a two variable maxprod and minprod and we can find the max and min after product of them with the given element.

2. If the element is less than 0 than swap the maximum product with minimum product because product of two negative (-) element is postive (swap ?)

```
class Solution {
public:
    int maxProduct(vector<int>& nums) {
        int ans = nums[0];
        int minprod = ans;
        int maxprod = ans;
        for(int i=1;i<nums.size();i++)
        {
            if(nums[i]<0)
            {
                swap(maxprod,minprod);
            }
            maxprod = max(nums[i],maxprod*nums[i]);
            minprod = min(nums[i],minprod*nums[i]);
            ans = max(maxprod,ans);

        }

        return ans;

    }
};
```

[-2, 3, -4]

Initial, ans = -2

maxprod = -2

minprod = -2

i = 1, 2

-4 < 0

if (3 < 0)

No, Yes Swap (-6, 3) (swap)

minprod = min (-4, -12) (3, -6) = -6 = 3 = -12 (i = 2)

maxprod = max (-4, 24) (3, -6) = 3 = -6 = 24

ans = ~~3~~ 24 { max (minprod, maxprod) }

Time Complexity :- O(n)

7. 3Sum

15. 3Sum

Medium 19291 1846 Add to List Share

Given an integer array nums, return all the triplets `[nums[i], nums[j], nums[k]]` such that `i != j`, `i != k`, and `j != k`, and `nums[i] + nums[j] + nums[k] == 0`.

Notice that the solution set must not contain duplicate triplets.

Example 1:

```
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[1] = (-1) + 0 + 1 = 0.
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does not
matter.
```

Example 2:

```
Input: nums = [0,1,1]
Output: []
Explanation: The only possible triplet does not sum up to 0.
```

Approach:-

1. Firstly, we will sort the array by using sort function

2. check for i if the number is duplicate than continue.

3. Iterate until j<k and check if the element is equal to 0 than it will be push to the vector.

4. In this also check for duplicate number . and also check for sum is equal to 0 if not than increment /decrement the iteration

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

         vector<vector<int>> ans;

    sort(nums.begin(),nums.end());

    for(int i=0;i<nums.size();i++)
    {
        if(i>0 && nums[i]==nums[i-1])
            continue;

        int j = i+1;
        int k = nums.size()-1;

        while(j<k)
        {
         if(nums[i]+nums[j]+nums[k]==0)
        {
           ans.push_back({nums[i],nums[j],nums[k]});

            int val1=nums[j];

            while(j<k && nums[j]==val1)
                j++;

             int val2=nums[k];

            while(j<k && nums[k]==val2)
                k--;
        }
            else if(nums[i]+nums[j]+nums[k]<0)
                j++;

            else if(nums[i]+nums[j]+nums[k]>0)
                k--;
    }
}
    return ans;
  }
```

8. Trapping Rain water

42. Trapping Rain Water

Hard 19981 282 Add to List Share

Given `n` non-negative integers representing an elevation map where the width of each bar is `1`, compute how much water it can trap after raining.

Example 1:


```
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.
```

Example 2:

```
Input: height = [4,2,0,3,2,5]
Output: 9
```

Constraints:

- `n == height.length`
- `1 <= n <= 2 * 10^4`
- `0 <= height[i] <= 10^5`

Approach:-

1. In this we will find the maximum height from the left side and similarly from right side by using while loop

2. After that we will compare the maximum of left height with right height if it is less then the maximum of right height then substract the element of left height and l++.

water + = lmax-height[l];
l++;

3. Otherwise,

water += rmax-height[r];
r--;

```
class Solution {
public:
    int trap(vector<int>& height) {

        int water = 0, l = 0, r = height.size()-1, lmax = INT_MIN, rmax = INT_MIN;
    while(l < r){
        lmax = max(height[l], lmax);
        rmax = max(height[r], rmax);

        if(lmax<rmax)
        {
            water += lmax-height[l];
            l++;
        }
        else
        {
            water += rmax-height[r];
            r--;
        }
    }
    return water;

    }
};
```

Time Complexity :- O(n)

9. Best Time To Buy and Sell Stock

121. Best Time to Buy and Sell Stock

Easy 17805 575 Add to List Share

You are given an array `prices` where `prices[i]` is the price of a given stock on the i^{th} day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return `0`.

Example 1:

```
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.
```

Example 2:

```
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.
```

Constraints:

- `1 <= prices.length <= 10^5`
- `0 <= prices[i] <= 10^4`

Approach:-

1. In this we will find the minimum price and simultaneously we will calculate maximum profit by subtracting min price

```cpp
class Solution {
public:
    int maxProfit(vector<int>& prices) {

        int maxpro=0;
        int minpri = INT_MAX;

        for(int i=0;i<prices.size();i++)
        {
            minpri = min(minpri,prices[i]);
            maxpro = max(maxpro,prices[i]-minpri);

        }
        return maxpro;
    }
};
```

Time Complexity :- O(n)

10. Merge Intervals

56. Merge Intervals

Medium 14713 547 Add to List Share

Given an array of `intervals` where `intervals[i] = [start_i, end_i]`, merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input.*

Example 1:

```
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
```

Example 2:

```
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

Constraints:

- `1 <= intervals.length <= 10^4`
- `intervals[i].length == 2`
- `0 <= start_i <= end_i <= 10^4`

Approach:-

1. Firstly , Sort an array by using sort function.

2. Then by using vector we will push the array of 0 index interval into the vector as :-

ans.push_back(intervals[0]);

3. After that we will check the ans array end element with the interval next position array start element , If it is greater than this then we will find the max of both end position element and update ans .

```
if ( ans [ j ] [ 1 ] > = intervals [ i ] [ 0 ] )
        {
ans [ j ] [ 1 ] = max ( ans [ j ] [ 1 ] , intervals [ i ] [ 1 ] );
        }
```

4. Otherwise ,

```
ans . push_back ( intervals [ i ] );
j++;
```

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

        vector<vector<int>> ans;

        sort(intervals.begin(),intervals.end());

        int n=intervals.size();

        ans.push_back(intervals[0]);

        int j=0;

        for(int i=1;i<n;i++)
        {
            if(ans[j][1]>=intervals[i][0])
            {
                ans[j][1]=max(ans[j][1],intervals[i][1]);
            }
            else
            {
                ans.push_back(intervals[i]);
                j++;
            }
        }
    return ans;

    }
};
```

Part -2 soon.....