Two Pointers →

↳ used to find a set of element that fulfill certain constraits, the set of elements could be a pair, a triplets or even a subarray

Certain things to Note →

(i) Two pointer primarily only works when array is sorted.

(ii) If in any question we have to return index then cannot modify the array like sorting and all.

(iii) But if any question we have to return the element of the array we can modify the original arrays.

Problem Statement

Given an array of sorted numbers and a target sum, find a **pair in the array whose sum is equal to the given target.**

Write a function to return the indices of the two numbers (i.e. the pair) such that they add up to the given target.

Example 1:

```
Input: [1, 2, 3, 4, 6], target=6
Output: [1, 3]
Explanation: The numbers at index 1 and 3 add up to 6: 2+4=6
```

Example 2:

```
Input: [2, 5, 9, 11], target=11
Output: [0, 2]
Explanation: The numbers at index 0 and 2 add up to 11: 2+9=11
```

Try it yourself

Try solving this question here:

↳ Since the array is sorted we can solve it in two ways.

1)
```
# include < bits/stdc++.h>
using namespace std;

vector<int> findPairs(vector<int> arr, int target){
    int left = 0;
    int right = arr.size() - 1;
    while (left < right){
        int currentSum = arr[left] + arr[right];
        if( currentSum == target) return {left, right};
```

```
            if(currentSum > targetSum) right--;
            else left++;
        }
        return {-1,-1};
    }
```

2- Solution →

```
vector<int> twoPair2(vector<int> arr, int target){
    vector<int,int> mab;
    for(int i=0; i<arr.size(); i++){
        int targetVal = target - arr[i];
        if(mab.count(targetVal)){
            return {mab[targetval], i};
        }
        mab[arr[i]] = i;
    }
    return {-1,-1};
}
```

$x + y = target$

$x = target - y$

1 : 0
2 : 1
3 : 2

Problem - 2) Find Duplicates

Problem Statement

Given an array of sorted numbers, **remove all duplicates** from it. You should **not use any extra space**; after removing the duplicates in-place return the new length of the array.

Example 1:

```
Input: [2, 3, 3, 3, 6, 9, 9]
Output: 4
Explanation: The first four elements after removing the duplicates will be [2, 3, 6, 9].
```

Example 2:

```
Input: [2, 2, 2, 11]
Output: 2
Explanation: The first two elements after removing the duplicates will be [2, 11].
```

Try it yourself

Try solving this question here:

```
int removeDuplicates(vector<int> arr){
    int slow = 1;
    for(int fast = 1; fast < arr.size; fast++){
        if(arr[slow - 1] != arr[fast]){
            arr[slow] = arr[fast];
            slow++;
        }
    return slow;
    }
```

Problem - 3) Squares of a sorted Array

Problem Statement

Given a sorted array, create a new array containing **squares of all the number of the input array** in the sorted order.

Example 1:

```
Input: [-2, -1, 0, 2, 3]
Output: [0, 1, 4, 4, 9]
```

Example 2:

```
Input: [-3, -1, 0, 1, 2]
Output: [0 1 1 4 9]
```

Try it yourself

Try solving this question here:

↳ Important tip to for pushing elements from right to left if we know the length of the array with out using a deque.

```
vector<int> makeSquare(vector<int>& arr){
    int n = arr.size();
    vector<int> squares(n);
    int left = 0;
    int right = arr.size() - 1;
    int highestSquareIdx = n - 1;
    while(left <= right){
```

```
        int leftSqaure = arr[left] * arr[left];
        int rightSqaure = arr[right] * arr[right];
        if (leftSqaure > rightSqaure) {
            Sqaure[highestSqaureIdx--] = leftsqaure;
            left++;
        } else {
            Sqaure[highestSqaureIdx--] = rightSqaure;
            right--;
        }
    }
    return squares;
}
```

Problem Statement - 3 Sum

Problem Statement

Given an array of unsorted numbers, find all **unique triplets in it that add up to zero.**

Example 1:

```
Input: [-3, 0, 1, 2, -1, 1, -2]
Output: [-3, 1, 2], [-2, 0, 2], [-2, 1, 1], [-1, 0, 1]
Explanation: There are four unique triplets whose sum is equal to zero.
```

Example 2:

```
Input: [-5, 2, -1, -2, 3]
Output: [[-5, 2, 3], [-2, -1, 3]]
Explanation: There are two unique triplets whose sum is equal to zero.
```

Try it yourself

Try solving this question here:

given → ATQ

↳ $x + y + z = 0$

$x = -(y + z)$; → same as two sum

↓ target ↳ Sum

Since the given array is unsorted and we need to return triplets we can modify the original array.

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

```
vector<vector<int>> threeSum(vector<int>& nums){
    sort(nums.begin(), nums.end());
    vector<vector<int>> result;
    for(int i=0; i<nums.size(); i++){
        if(i>0 && nums[i]==nums[i-1]) continue;
                                  ↳ skip Duplicates
        int left = i+1;
        int right = nums.size()-1;
        while(left<right){
            int sum = nums[0]+nums[left]+nums[right];
            if(sum==0){
                result.push_back({nums[i],nums[left],nums[right]})
                // skips Duplicates >
                while(left<right && nums[left]==nums[left+1]){
                    left++;
                }
                while(left<right && nums[right]==nums[right+1]){
                    right--;
                }
                left++;
                right--;
            } else if(sum<0){
                left++;
            } else{
                right--;
```

```
        }
    }
  }
  return result;
 }
}
```

Problem - Triplet Sum Close to target

Problem Statement

Given an array of unsorted numbers and a target number, find a **triplet in the array whose sum is as close to the target number as possible**, return the sum of the triplet. If there are more than one such triplet, return the sum of the triplet with the smallest sum.

Example 1:

```
Input: [-2, 0, 1, 2], target=2
Output: 1
Explanation: The triplet [-2, 1, 2] has the closest sum to the target.
```

```
#include<bits/stdc++.h>
using namespace std;
    int searchTriplets(vector<int>& arr, int targetSum){
        sort(arr.begin(), arr.end());
        int smallestDifference = numeric_limits<int>::max();
        for(int i=0; i<arr.size(); i++){
            int left = i+1;
            int right = arr.size()-1;
            while(left<right){
                int targetDiff = targetSum - arr[i]- arr[left]-arr[right];
                if(targetDiff==0){
                    return targetSum;
                }
```

```
            if(abs(targetDiff)< abs(smallDifference)){
                smallestDifference = targetDiff;
            }
            if ( targetDiff > 0){
                left++;
            } else{
                right--;
            }
        }
    }
    return targetSum - SmallestDifference;
}
```

Problem - Triplets with Smaller Sum

Problem Statement

Given an array `arr` of unsorted numbers and a target sum, **count all triplets** in it such that `arr[i] + arr[j] + arr[k] < target` where `i`, `j`, and `k` are three different indices. Write a function to return the count of such triplets.

Example 1:

```
Input: [-1, 0, 2, 3], target=3
Output: 2
Explanation: There are two triplets whose sum is less than the targe
t: [-1, 0, 3], [-1, 0, 2]
```

Example 2:

```
Input: [-1, 4, 2, 1, 3], target=5
Output: 4
Explanation: There are four triplets whose sum is less than the target:
   [-1, 1, 4], [-1, 1, 3], [-1, 1, 2], [-1, 2, 3]
```

Try it yourself

Try solving this question here:

$\forall T\varphi \rightarrow i + j + K < target$.

$targetDiff = target - (i + j + K)$.

```
int tripletsSmallerSum(vector<int> arr, int target){
    sort(arr.begin(), arr.end());
    int count=0;
    for(int i = 0; i < arr.size(); i++){
        int left= i+1;
        int right = arr.size()-1;
        while( left < right){
```

```
        if (arr[left] + arr[right] < target) {
            // found the triplets
            // since arr[right] >= arr[left], therefore, we can
               replace arr[right] by any number between left and
               right to a get sum less than the target sum.
            count += right - left;
            left++;
        } else {
            right--; // we need a pair with a smaller sum.
        }
    }
    return count;
}
```

Problem - Subarray with Product less than a Target.

Problem Statement

Given an array with positive numbers and a target number, find all of its contiguous subarrays whose **product is less than the target number.**

Example 1:

```
Input: [2, 5, 3, 10], target=30
Output: [2], [5], [2, 5], [3], [5, 3], [10]
Explanation: There are six contiguous subarrays whose product is less than the target.
```

Example 2:

```
Input: [8, 2, 6, 5], target=50
Output: [8], [2], [8, 2], [6], [2, 6], [5], [6, 5]
Explanation: There are seven contiguous subarrays whose product is less than the target.
```

Try it yourself

Try solving this question here:

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

    vector<vector<int>> findSubarrays(vector<int>& arr, int target){
    vector<vector<int>> result;
    int product = 1;
    int left = 0;
```

```
for( int right = 0; right < arr.size(); right++){
    product *= arr[right];
    while( product >= target && left < arr.size()){
        product /= arr[left++];
    }
    // if we just want return count
    count += right - left - 1;   // how to know when R-L & R-L+1
    return count;
```

when Inclusive [a, b] = Range → b - a + 1

when b Exclusive [a, b) = Range → b - a

```
    deque <int> templist;
    for ( int i = right; i >= left; i--){
        templist.push_front(arr[i]);
        vector<int> resultvec;
        move( tempList.begin(), tempList.end(), back_inserter(resultVec))
        result.push_back(resultVec);
    }
}
return result;
}
```

one way to copy/mov ↙ arr/queue.

Dutch National Flag

Problem Statement

Given an array containing 0s, 1s and 2s, sort the array in-place. You should treat numbers of the array as objects, hence, we can't count 0s, 1s, and 2s to recreate the array.

The flag of the Netherlands consists of three colors: red, white and blue; and since our input array also consists of three different numbers that is why it is called Dutch National Flag problem.

Example 1:

```
Input: [1, 0, 2, 1, 0]
Output: [0 0 1 1 2]
```

Example 2:

```
Input: [2, 2, 0, 1, 2, 0]
Output: [0 0 1 2 2 2 ]
```

```cpp
#include <bits/stdc++.h>
using namespace std;
    void sort(vector<int> &arr){
        int low = 0;
        int high = arr.size()-1;
        int i = 0;
        while(i <= high){
            if(arr[i] == 0){
                swap(arr, i, low);
                i++;
                low++;
            }else if(arr[i] == 1){
                i++;
            }else{
                swap(arr, i, high);
                high--;
            }
        }
    }
```

this wont work.

```cpp
for(int i = 0; i < arr.size();){
```

Problem - 4 Sum

Quadruple Sum to Target (medium)

Given an array of unsorted numbers and a target number, find all **unique quadruplets** in it, whose **sum is equal to the target number.**

Example 1:

```
Input: [4, 1, 2, -1, 1, -3], target=1
Output: [-3, -1, 1, 4], [-3, 1, 1, 2]
Explanation: Both the quadruplets add up to the target.
```

Example 2:

```
Input: [2, 0, -1, 1, -2, 2], target=2
Output: [-2, 0, 2, 2], [-1, 0, 1, 2]
Explanation: Both the quadruplets add up to the target.
```

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

vector<vector<int>> searchQuadruplets(vector<int>& arr, int target)
    sort(arr.begin(), arr.end());
    vector<vector<int>> quadruplets;
    for(int i=0; i<arr.size(); i++){
        if(i>0 && arr[i]==arr[i-1]){
            continue; // skip Duplicates.
        }
        for(int j=i+1; j<arr.size(); j++){
            if(j>i+1 && arr[j]==arr[j-1]){
                continue;
            }
            int left = j+1;
            int right = arr.size()-1;
            while(left<right){   // use long long if overflow.
                int sum = arr[i]+arr[j]+arr[left]+arr[right];
                if(sum==target){
                    quadruplet.push_back({arr[i],arr[j],arr[left],arr[right]})
                    left++;
                    right--;
```

```
                while(left<right && arr[left]==arr[left+1]){
                    left++;
                }
                while(left<right && arr[right]==arr[right-1]){
                    right--;
                }
            }else if(sum<target){
                left++;
            }else{
                right--;
            }
        }
    }
    return quadruplets;
};
```

Problems → Comparing Strings Containing Backspaces

e

Given two strings containing backspaces (identified by the character '#'), check if the two strings are equal.

Example 1:

```
Input: str1="xy#z", str2="xzz#"
Output: true
Explanation: After applying backspaces the strings become "xz" and "xz" respectively.
```

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

    bool compare(string &str1, string &str2){
        int index1=str1.length()-1;
        int index2=str2.length()-1;
```

```
while( index1 >= 0 || index2 >= 0){
    int i1 = getNextValidCharIndex(str1, index1);
    int i2 = getNextValidCharIndex(str2, index2);
    if( i1 < 0 && i2 < 0){ // reached the end of both the strings.
        return true;
    }
    if(i1 < 0 || i2 < 0){// reached the end of one of the strings.
        return false;
    }
    if(str1[i1] != str2[i2]){ // check if the characters are equal.
        return false;
    }
    index1 = i1 - 1;
    index2 = i2 - 1;
}
return true;
}
int getNextValidCharIndex( string &str, int index){
int backspaceCount = 0;
while( index >= 0){
    if( str[index] == '#'){
        backspaceCount++;
    } else if ( backspaceCount > 0){
        backspaceCount--;
    } else {
```

```
            break;
        }
        index--;
    }
    return index;
}
```

Problem → Minimum Window Sort (medium).

Minimum Window Sort (medium)

Given an array, find the length of the smallest subarray in it which when sorted will sort the whole array.

Example 1:

```
Input: [1, 2, 5, 3, 7, 10, 9, 12]
Output: 5
Explanation: We need to sort only the subarray [5, 3, 7, 10, 9] to make the whole array sorted
```

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

int sort(vector<int>& arr){
    int low = 0;
    int high = arr.size()-1;
    // find the first number out of sorting order from the beggining.
    while(low<arr.size()-1 && arr[low]<=arr[low+1]){
        low++;
    }
    if(low==arr.size()-1){ // if the array is sorted
        return 0;
    }
    // find the first number out of sorting order from the end
    while(high>0 && arr[high]>=arr[high-1]){
        high--;
    }
```

```
// find the maximum & minimum of the subarray.
int subarrayMax = numeric_limits<int>::min();
int subarrayMin = numeric_limits<int>::max();
for(int k = low, k= high, k++){
    subarrayMax = max(subarrayMax, arr[k]);
    subarrayMin = min(subarrayMin, arr[k]);
}
// extend the subarray to include which is bigger than
  the minimum of the subarray.
while(low > 0 && arr[low-1] > subarrayMin){
      low--;
}
// extend the subarray from the include any number
  which is smaller than the maximum of the subArray
while(high < arr.size()-1 && arr[high+1]< subarrayMax){
      high++;
}
return high - low + 1;
}
```