M T W T F S S
Page No.
Date
YOUVA

-> 25/Jan/24

- HackerRank Problem Solving (Basic) 9ome
- Find min max array method that returns an array. Checks edge cases
- Triplet Sum (Three Sum).

Brute force Approach >>

```
for  i=0   i<n-2
  for
  if (ith == target) {
     for  j=i+1  j<n-1
        if  j+i <= target
            for  k=2j+1  k<n
                if  j+i+k == target
                    count++
                }
            else continue;
        else continue;
    return count;
```

only works for positive values

Brute force O(n^3)

- Find the unique number in given array where all other elem are twice.

Bit manipulation: XOR.

```
int ans = arr[0] // Edge Case not to handle as atleast 1 elem
for int i=1; i<arr.length; i++) {
    ans ^= arr[i]  ---> return ans.
```

O(n) Complexity

-> Return Second longest integer in array [Assume length is atleast 2) as it always exists.

edge case handler

```
public static int getSecondMax(int[] arr){
    int max = Integer.MIN_VALUE;
    int secondMax = Integer.MIN_VALUE;
    for int i=0; i < arr.length; i++ {
        if (arr[i] > max) {
            ~~max =~~
            secondMax = max
            max = arr[i];
        } else if ( arr[i] > secondMax){
            secondMax = arr[i];
        }
    } return secondMax;
```

-> Return the first value that is repeating in array, if all unique return -1

- Brute force $O(n^2)$ >>

```
for int i=0   i < arr.length-1
    for int j = i+1   j < arr.length
        if arr[i] == arr[j]
            return arr[i]
    }
}return -1
```

even if not (-1) still fine cause this will be false

-> In place : It means you must not create the new array but only swap or update the original array directly.

-> Swap two variables a and b without temporary variable.

```
a = 5          // only for premitive.
b = 3

a = a+b; 8
b = a-b; 5 (a)
a = a-b; 3 (b)
```

issue for large values, problem of overflow / underflow.

-> Reverse an Array] (in place)

```
n = arr.length-1
if n=0 -> return (edge case)
for i=0  i<=n/2
    swap(arr[i], arr[n-i])
```

-> Rotate an array by k index. point (k can be greater than n)

```
0  1  2  3  4
[10, 20, 30, 40, 50]
k = 2 ) [40, 50, 10, 20, 30]
k = 9 ) 9-1 % n = 9-1 % 5 = 4 [20, 30, 40, 50, 10]
                  -1.
```

1. Create new array
2. for(i) i = n-2 i<n i++
3. for(j) j = 0 j<n-2 j++

return newArr ... where k=2

-> Rotate Array without extra space!
(k > n) possible

7 - 4 = 3

-> [10, 20, 30, 40, 50, 60, 70]
k = 11 -> k = 11 % 7 = 4
[40, 50, 60, 70, 10, 20, 30]

n - k^th elem + index % n
7 - 4 + 2 % 7
5 % 7 -> 5 (index of original array)

n - k^th elem + index % n
7 - 4 + 4 % 7 = 0

-> Soln works and reduce space to one loop but still need extra array space!

- without Space Approach >>

(end exclusive)

1. Reverse entire array (0, n)
2. Reverse first part (0, k)
3. Reverse second part (k, n)

reverse func (int[] arr, int start, int end){
end--; // exclusion
while (start < end){
int temp = arr[start]
arr[start++] = arr[end]
arr[end--] = temp;
}

(or) if k = 0 return
k = 0

[Frequency Array]

→ Suppose you have an array of length n. You have to find an elem. (Linear Search)

But] Inputs are 10 million. Majority of them might repeat but they are 10 million queries!

Will you use linear search of O(n) complexity to traverse 10 million times ?

All values in array are 0 to 10^5 at max. So, Create an array of length 10^5. It is called frequency array.

```
for (int i = 0; i < arr.length; i++){
    freq[arr[i]]++;
}
```

Input -> 23

freq[23] > 0 then return true
else false.

Disadvantage: It cannot return index of the item though like linear search