3Sum Unique {Leetcode : 15}

15. 3Sum

Medium 16260 1559 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]]
```

{Return triplets}

-1, 0', 0'', 1, 2, -1, -4

{-1, 0', 1} This is duplicacy and it is not allowed.

{-1, 0'', 1}

After sorting, triplets should be unique.

Brute force:

```
for(   ){
    for(   ){
        for(   ){
        }
    }
}
```

$T.C. = O(N^3)$

Optimization: 3 Pointer Approach; Normal Loop + Two Pointer.

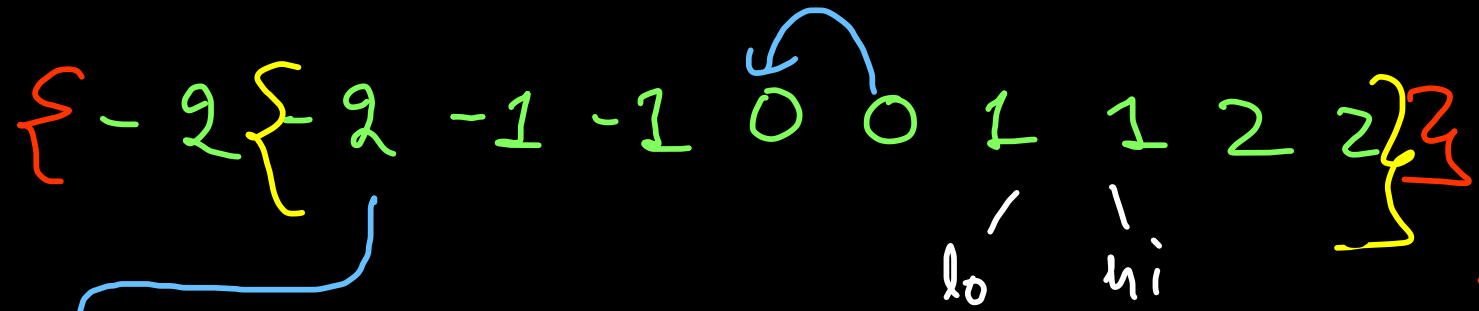
$k = x + y + z \Rightarrow \Rightarrow$

$x \Downarrow a$

$k = a + (y + z)$

$y + z = (k - a)$

$y + z$ — Target Sum Pair (2 Ptr)

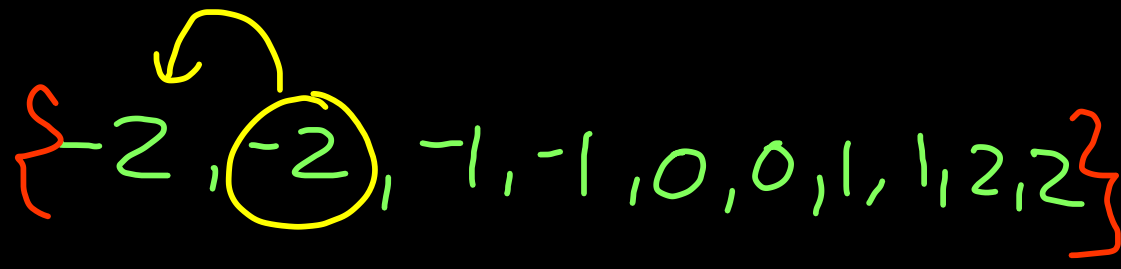


target = 0

target = 0 - (-2) = 2

In the two pointer loop condition that nums[i-1] != nums[i] should be applied to avoid duplicacy.

{-2, 0, 2} {-2, 1, 1}



nums[i] != nums[i-1] condition should be applied here as well.

3 Sum Code

{ LC }

```java
public List<List<Integer>> twoSumUnique(int[] nums, int target,int lo, int hi) {
    //array will already be sorted
    List<List<Integer>> ans = new ArrayList<>();
    int start = lo;
    while(lo < hi) {
        List<Integer> smallAns = new ArrayList<>();
        if(lo > start && nums[lo - 1] == nums[lo]) {
            lo++;
            continue;
        }

        int sum = nums[lo] + nums[hi];
        if(sum == target) {
            smallAns.add(nums[lo]);
            smallAns.add(nums[hi]);
            lo++;
            hi--;
        } else if(sum < target) {
            lo++;
        } else {
            hi--;
        }

        if(smallAns.size() > 0) ans.add(smallAns);
    }
    return ans;
}

public List<List<Integer>> threeSum(int[] nums) {

    Arrays.sort(nums);
    List<List<Integer>> ans = new ArrayList<>();
    for(int i=0;i<nums.length;i++) {

        int target = 0 - nums[i];
        if(i>0 && nums[i-1] == nums[i]) continue;

        List<List<Integer>> targetSumPairs = twoSumUnique(nums,target,i+1,nums.length-1);
        for(List<Integer> tsp : targetSumPairs) {
            tsp.add(0,nums[i]);
            ans.add(tsp);
        }
    }

    return ans;
}
```

lo > start & not lo > 0

because

$-1\{\underset{\uparrow}{1}\ 0\ 0\ 1\ 2\}$

when I am starting 2 ptr loop for this then I will skip this -1 because it is same as prev -1 however, this should have been included. So, check condn if lo > start so that it does not skip 1st ele.

$TC = O(N^2)$

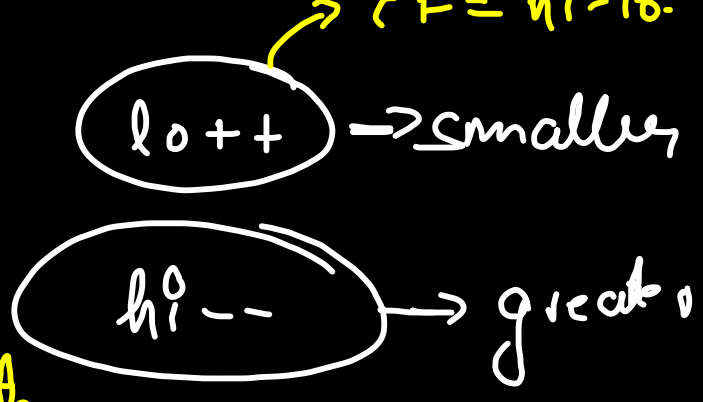
3 Sum Smaller (LintCode)

lo lo hi lo hi hi

{ -2, -2, 1, 2, 3 }

0 1 2 3 4

target = 2



3 pointers: Normal Loop + Two Pointers

{ 2 sum smaller }

** Duplicate Triplets are allowed in this Ques.

Code for 3 Sum Smaller (LintCode)

```java
public int twoSumSmaller(int[] nums, int target,int lo, int hi) {
    int ans = 0;
    while(lo < hi) {
        int sum = nums[lo]+nums[hi];
        if(sum < target) {
            ans += (hi-lo);
            System.out.println(nums[lo] + " " + nums[hi]);
            lo++;
        } else {
            hi--;
        }
    }
    return ans;
}

public int threeSumSmaller(int[] nums, int target) {
    // Write your code
    Arrays.sort(nums);

    int ans = 0;
    for(int i=0;i<nums.length;i++) {
        // if(i> 0 && nums[i-1] == nums[i]) continue;
        int numberOfPairs = twoSumSmaller(nums,target - nums[i],i+1,nums.length-1);
        ans += numberOfPairs;
    }

    return ans;
}
```

$TC = O(N^2)$

3 Sum Closest

3 pointer approach: Normal loop + 2 pointer closest

{Closest Sum Checking}

Duplicates may or may not be there. It does not matter in this question because we are not counting / displaying triplets.

3 Sum Closest Code {Leetcode}

```java
public int twoSumClosest(int[] nums,int target, int lo, int hi) {
    int minDiff = Integer.MAX_VALUE;
    int ans = 0;

    while(lo < hi) {
        int sum = nums[lo] + nums[hi];
        if(sum == target) {
            return target;
        } else if(sum > target) {
            int currDiff = sum - target;
            if(currDiff < minDiff) {
                minDiff = currDiff;
                ans = sum;
            }
            hi--;
        } else {
            int currDiff = target - sum;
            if(currDiff < minDiff) {
                minDiff = currDiff;
                ans = sum;
            }
            lo++;
        }
    }
    return ans;
}
```

```java
public int threeSumClosest(int[] nums, int target) {
    Arrays.sort(nums);
    int minDiff = Integer.MAX_VALUE;
    int ans = 0;

    for(int i=0;i<nums.length-2;i++) {
        int closestPairSum = twoSumClosest(nums,target-nums[i],i+1,nums.length-1);
        int currSum = closestPairSum + nums[i];
        int currDiff = Math.abs(target - currSum);
        if(currDiff < minDiff) {
            minDiff = currDiff;
            ans = currSum;
        }
    }

    return ans;
}
```

Count the Triplets {GfG}

{1, 2, 3, 4, 5, 6}

{{1, 2, 3, 4, 5}, 6}

a+b+c=0

a+b=c

⇒ a+b-c=0

Important point is that we have to fix the third variable, not the first. If you fix a 2 pointer will be applied on difference i.e. c-b.

Code for count the triplets.

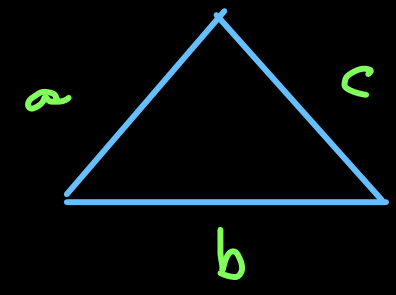
```java
int twoSum(int[] arr, int target, int lo, int hi) {
    int ans = 0;

    while(lo < hi) {
        int sum = arr[lo] + arr[hi];
        if(sum == target) {
            lo++;
            hi--;
            ans++;
        } else if(sum < target) {
            lo++;
        } else {
            hi--;
        }
    }

    return ans;
}

  int countTriplet(int arr[], int n) {
      // code here
      Arrays.sort(arr);
     int ans = 0;
     for(int i = n-1;i > 1;i--) {
        int twoSumPairs = twoSum(arr,arr[i],0,i-1);
        ans += twoSumPairs;
     }

     return ans;
  }
```

O(N)

$O(N^2)$

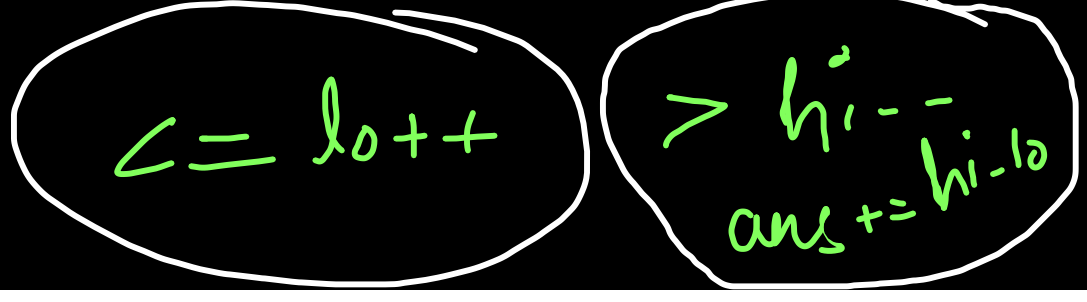
Valid Triangle Number {LC : 611}

{1, 2, 3, 4, 5, 6}



$$\begin{cases} a+b > c \\ b+c > a \\ a+c > b \end{cases}$$

So, it is same as prev question. Instead of a+b = c we have a+b > c.



If the array {a, b, c} is a sorted array, proving 1st will prove the rest 2 automatically.

Valid Triangle Number 문제

```java
public int twoSumGreater(int[] nums, int target,int lo, int hi) {
    // write your code here
    int ans = 0;
    while(lo < hi) {
        int sum = nums[lo] + nums[hi];
        if(sum > target) {
            ans += (hi-lo);
            hi--;
        } else {
            lo++;
        }
    }

    return ans;
}
```

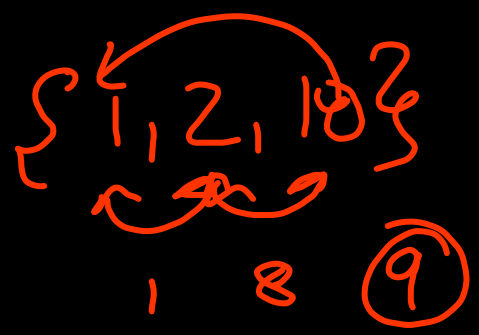
```java
public int triangleNumber(int[] nums) {
    Arrays.sort(nums);
    int ans = 0;

    for(int i=nums.length-1;i>1;i--) {
        ans += twoSumGreater(nums,nums[i],0,i-1);
    }

    return ans;
}
}
```

Array 3 Pointers {Interview Bit}

```
max(abs(A[i] - B[j]), abs(B[j] - C[k]), abs(C[k] - A[i]))
```

minimize this term

A : [1, 4, 10]
B : [2, 15, 20]
C : [10, 12]



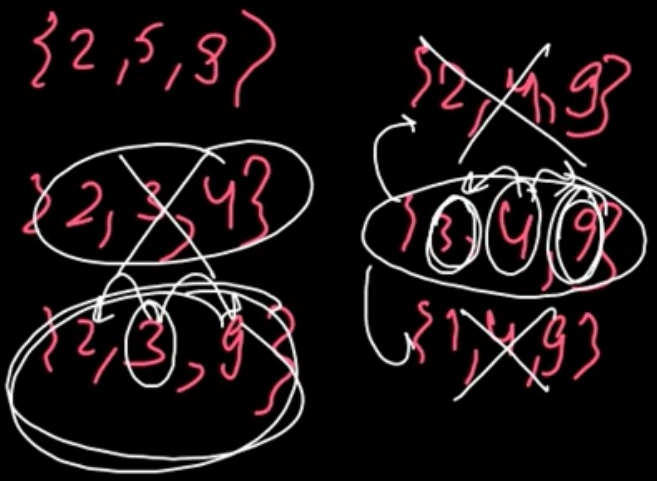
** Algo: Keep incrementing the min val ptr till any one array is finished.

Array 3 Pointers (IB) Code.

```java
// DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ ONLY
public int minimize(final int[] A, final int[] B, final int[] C) {
    int i = 0, j=0,k=0;
    int ans = Integer.MAX_VALUE;
    while(i<A.length && j<B.length && k<C.length) {
        int a = Math.abs(A[i] - B[j]), b = Math.abs(B[j] - C[k]),c = Math.abs(A[i] -C[k]);
        ans = Math.min(ans,Math.max(a,Math.max(b,c)));

        if(A[i] <= B[j] && A[i]<= C[k]) i++;
        else if(B[j] <= A[i] && B[j] <= C[k]) j++;
        else k++;
    }

    return ans;
}
```

Maximum Sum Triplet {IB}

```
[2, 5, 3, 1, 4, 9]
```



(i, j, k)

(i < j < k)

(A[i] < A[j] < A[k])

A[i] + A[j] + A[k] = max^m

HINT → Fix the middle ele

left → Floor in left part

right → max^m in right part

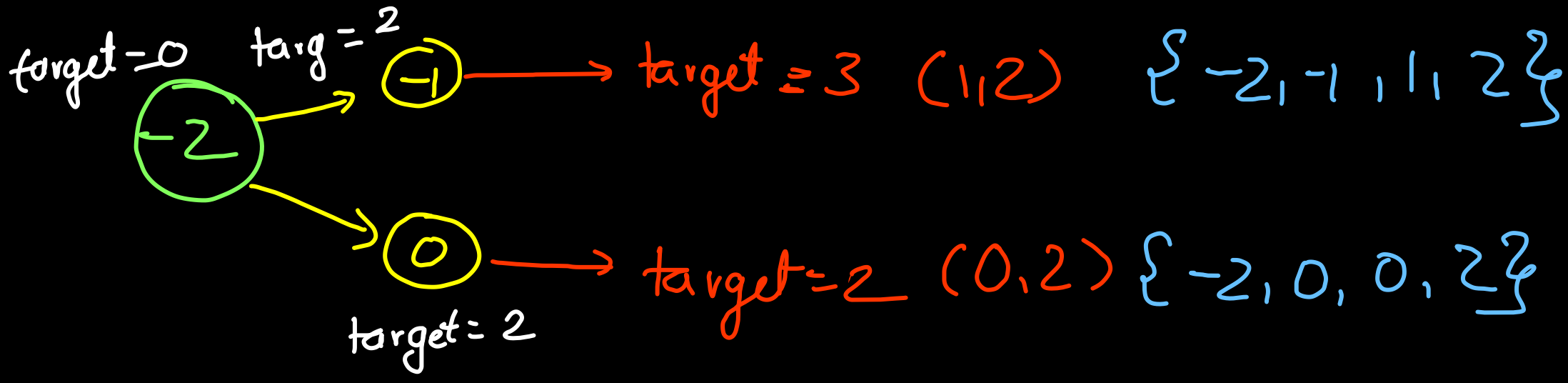
4 Sum { Leetcode }

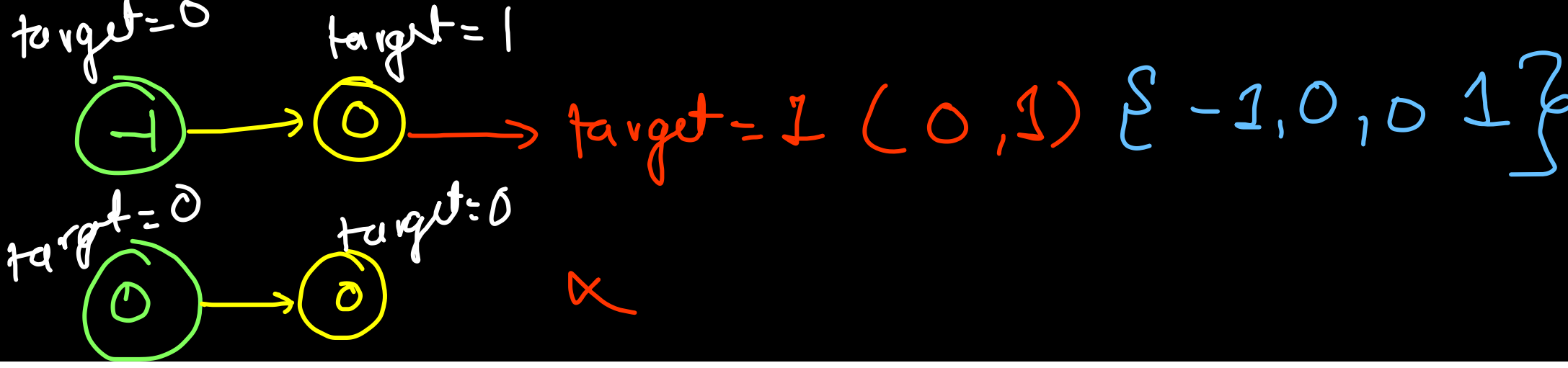
Brute force; 4 nested loops $O(n^4)$

Optimised; Nested Loop + 2 Pointer Approach

{ -2, { -1, 0, 0, 1, 2 } }

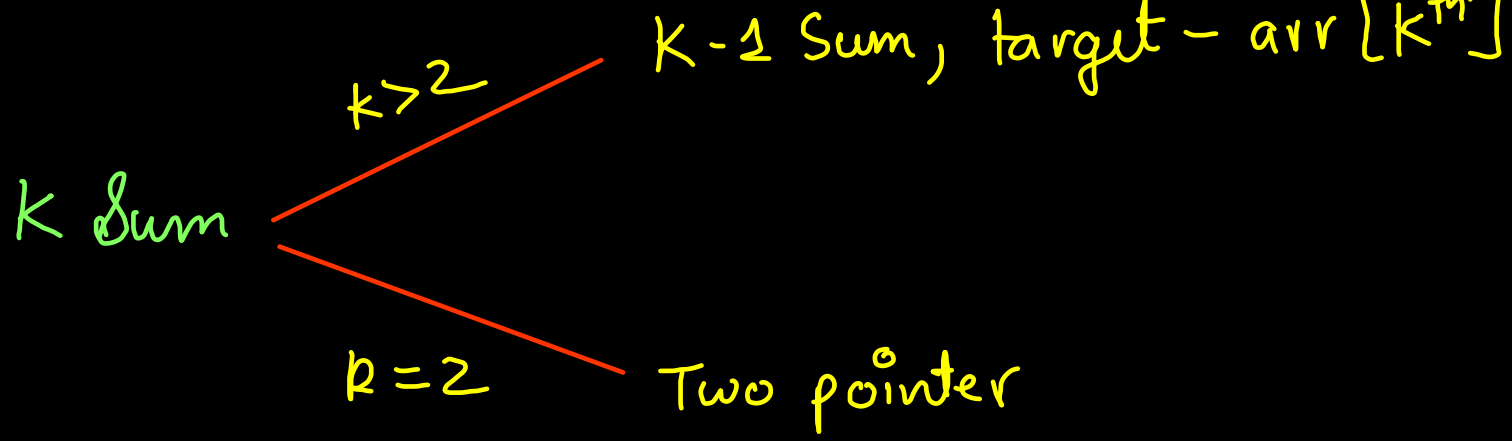
{-2, -1, 0, 0, 1, 2}


target=0
targ=2
-2
-1
target=3 (1,2) {-2,-1,1,2}
0
target=2 (0,2) {-2,0,0,2}
target=2


target=0
target=1
-1
0
target=1 (0,1) {-1,0,0,1}
target=0
target=0
0
0

K Sum

Recurrence Relation



$O(n^{k-1})$

$$O(k\,Sum) = \begin{cases} O((k-1)\,Sum * N & k>2 \\ O(N) & k=2 \end{cases}$$

K Sum (4 Sum Leetcode)

```java
    public List<List<Integer>> kSum(int[] nums, int start,int target,int k) {
        if(k == 2) {
            return twoSumUnique(nums,target,start);
        }

        List<List<Integer>> res = new ArrayList<>();
        for(int i=start; i<= nums.length-k;i++) {
            if(i>start && nums[i] == nums[i-1]) continue;
            List<List<Integer>> subRes = kSum(nums,i+1,target-nums[i],k-1);
            for(List<Integer> sub : subRes) {
                sub.add(0,nums[i]);
                res.add(sub);
            }
        }

        return res;
    }

    public List<List<Integer>> fourSum(int[] nums, int target) {
        Arrays.sort(nums);
        return kSum(nums,0,target,4);
    }
}
```

4 Sum II {leetcode}

Instead of giving a single array & finding quadruplets equal to the target sum, we are given 4 arrays this time - and we have to make quadruplets from them.

⇒ Approach 1 ; same as array 3 pointers question (4 Pointer Approach)

① Sort all arrays ;
② Take 4 pointers i, j, k, l
③ Keep updating the smallest ptr till 1 array is completed

This approach won't work. Many pairs will be skipped.

Approach-2

$$a + b + c + d = 0$$

A $\{1, 2, 2\}$

B $\{-2, -2, -1\}$

} Pairs (AB)

C $\{-1, -1, 2\}$

D $\{0, 1, 2\}$

} Pairs (CD)

HM <Sum, AL<Pair>>

↳ all pairs of that sum

-1	(1, -2)
0	(1, -1) (2, -2)
1	(2, -1)

-1	(-1, 0)
0	(-1, -1)
1	(-1, 2)
2	(2, 0)
3	(2, 1)
4	(2, 2)

Worst case:

$O(n^4)$

$\{1, -2, -1, 2\}$

$\{2, -1, -1, 2\}$

Code for 4 Sum II {leetcode}

```
class Solution {
    public int fourSumCount(int[] nums1, int[] nums2, int[] nums3, int[] nums4) {
        HashMap<Integer,Integer> AB =  new HashMap<>();
        //store the sum of pair vs count
        for(int val1: nums1)
            for(int val2: nums2)
                AB.put(val1 + val2,AB.getOrDefault(val1 + val2,0) + 1);

        int count = 0;
        for(int val3: nums3) {
            for(int val4: nums4) {
                int target = -val3 - val4;
                count += AB.getOrDefault(target,0);
            }
        }

        return count;

    }
}
```

4 Sum Leetcode {HashMap Approach} {4 Sum - II Appr}

```
public List<List<Integer>> fourSum(int[] nums, int target) {
    //sum against all the pairs with that sum
    HashMap<Integer,List<List<Integer>>> AB = new HashMap<>();

    int n = nums.length;
//we have to store the indices of the pair and not the values to check for duplicates later
    for(int i=0;i<n;i++) {
        for(int j=i+1;j<n;j++) {
            int sum = nums[i] + nums[j];
            List<Integer> pair = new ArrayList<>();
            pair.add(i);pair.add(j);
            if(AB.containsKey(sum) == false) {
                List<List<Integer>> temp = new ArrayList<>();
                AB.put(sum,temp);
            }
            AB.get(sum).add(pair);
        }
    }

    HashSet<List<Integer>> hs = new HashSet<>(); //to check for uniqueness

        for(int p=0;p<n;p++) {
            for(int q=p+1;q<n;q++) {
                int remTarget = target - nums[p] - nums[q];
                if(AB.containsKey(remTarget) == false) {
                    continue;
                }
                for(List<Integer> pair : AB.get(remTarget)) {
                    if(p != pair.get(0) && p!= pair.get(1) && q!=pair.get(0) && q!=pair.get(1)) {
                        List<Integer> quad = new ArrayList<>();
                        quad.add(nums[p]);
                        quad.add(nums[q]);
                        quad.add(nums[pair.get(0)]);
                        quad.add(nums[pair.get(1)]);
                        Collections.sort(quad);
                        hs.add(quad);
                    }
                }
            }
        }

        List<List<Integer>> res = new ArrayList<>();
        for(List<Integer> quad: hs) {
            res.add(quad);
        }

        return res;
    }
```

This code gives TLE on Leetcode because WC is $O(n^4)$ & the test cases are strict.

Count Tuples Product & LC Tuples with same product &

1726. Tuple with Same Product

Medium 385 17 Add to List Share

Given an array `nums` of **distinct** positive integers, return *the number of tuples* `(a, b, c, d)` *such that* `a * b = c * d` *where* `a`, `b`, `c`, *and* `d` *are elements of* `nums`, *and* `a != b != c != d`.

No of quadruplets * 8

{2, 3, 4, 6}

① (2, 3)
② (2, 4)
③ (2, 6)
④ (3, 4)
⑤ (3, 6)
⑥ (4, 6)

×8

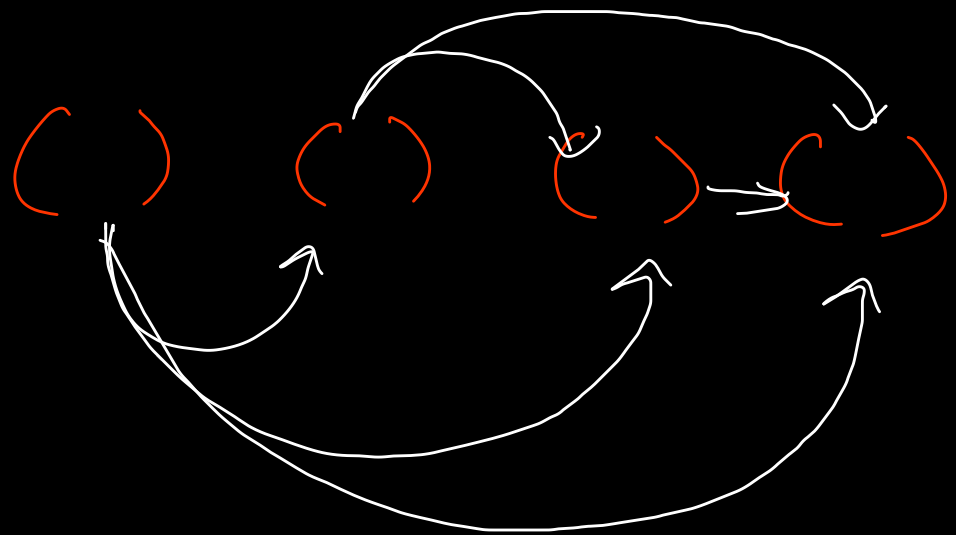
(2,6) (3,4)
(2,6) (4,3)
(6,2) (3,4)
(6,2) (4,3)
(3,4) (2,6)
(3,4) (6,2)
(4,3) (2,6)
(4,3) (6,2)

Count Tuples Code

```java
class Solution {
    public int tupleSameProduct(int[] nums) {
        HashMap<Integer,Integer> AB = new HashMap<>();

        for(int i=0;i<nums.length;i++) {
            for(int j=i+1;j<nums.length;j++) {
                AB.put(nums[i] * nums[j], AB.getOrDefault(nums[i] * nums[j],0) + 1);
            }
        }

        int count = 0;
        for(Integer key: AB.keySet()) {
            int val = AB.get(key);
            count = count + (val*(val-1))/2;
        }

        return count * 8;
    }
}
```

What is val*(val-1)/2 in Count Tuples?

⇒ Let us say we have 4 pairs with product k
Now quadruplets can be formed in the following way.



This is exactly how a nested loop works
i.e. $\frac{n(n-1)}{2}$ So,

val*(val-1)/2

Difference Pair

{ 1, 2, 3, 10, 11 }

lo ↑ hi ↑

target = 8

arr[p2] − arr[p1] = 8

↓ decreasing higher val

↑ increasing lower val

this means the same.

Hence 2 ptr in 2 opp direcs ⇒ should not be applied in diff pair

{ 1, 2, 3, 10, 11 }

p1 ↑ , p2 ↓

a[p2] − a[p1] = 1 ↑

badi value ko bada karo

{ 1, 2, 3, 10, 11 }

p1 ↑ , p2 ↓

a[p2] − a[p1] = 3 − 1 = 2 ↑

{ 1, 2, 3, 10, 11 }

p1 ↑ , p2 ↓

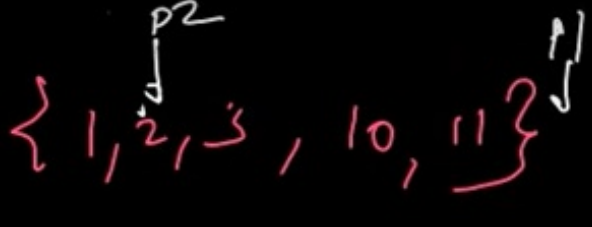
a[p2] − a[p1] = 10 − 1 = 9 ↓

Choti value ko bada karo

{ 1, 2, 3, 10, 11 }

p1 ↓ , b1 ↓ , p2 ↓

a[p2] − a[p1] = target



target = -20

2-1 = 1

2-2 = 0

2-3 = -1

2-10 = -8

2-11 = -9

At p1 can also reach arr.length before p2.

also target = 0

ptr1 != ptr2
pair found

ptr1 = ptr2
ptr2 ++
or ptr1 ++

```java
    */
    public int[] twoSum7(int[] nums, int target) {
        // write your code here
        int left = 0;
        int right= 1;

        while(left < nums.length && right<nums.length) {
            if(left == right) {
                if(target < 0) {
                    left++;
                } else {
                    right++;
                }
            }

            if(nums[right] - nums[left] == target) {
                int min = Math.min(nums[left],nums[right]);
                int max = Math.max(nums[left],nums[right]);
                return new int[]{min,max};
            } else if(nums[right] - nums[left] < target) {
                right++;
            } else {
                left++;
            }
        }

        return null;

    }
}
```

Code for Difference Pair

TC = O(N)

Count Diff Pairs {Unique} {LC: 532}

if diff < target ⇒ right++;

diff > target ⇒ left++;

diff == target ⇒ left++

because we will check arr[left] != arr[left+1] for duplicates.

Count Diff Pairs {Unique}

```java
    public int diffPair(int[] nums,int target) {
        int left = 0, right = 1;
        int count = 0;
        while(left< nums.length && right < nums.length) {

            if(left > 0 && nums[left] == nums[left-1]) {
                left++; continue;
            }

            if(left == right) {
                right++; continue;
            }

            int diff = nums[right] - nums[left];
            if(diff == target) {
                left++;
                count++;
            } else if(diff < target) {
                right++;
            } else {
                left++;
            }
        }

        return count;
    }

    public int findPairs(int[] nums, int k) {
        Arrays.sort(nums);
        return diffPair(nums,k);
    }
}
```

Count All Diff Pairs {LC : 2006}

2006 {All}

{1, 2, 2, 1}
0 1 2 3

1 → (2)
2 → (2)

target = (1)
2 * 2 = (4)

1	2
0	1
1	2
0	2
2	1
1	3
2	1
2	3

① Approach: Store each element and its frequency in a HashMap & multiply its freq with its comps' freq & add it to the total no of pairs.

Count All Diff Pairs {LC : 2006}

2006 {All}

{1, 2, 2, 1}
0 1 2 3

1 → (2)
2 → (2)

target = (1)
2 * 2 = (4)

1	2
0	1
1	2
0	2
2	1
1	3
2	1
2	3

① Approach: Store each element and its frequency in a HashMap & multiply its freq with its comps' freq & add it to the total no of pairs.

Count All Diff Pairs

```
    public int countKDifference(int[] nums, int k) {
        HashMap<Integer,Integer> fmap = new HashMap<>();

        for(int i=0;i<nums.length;i++) {
            fmap.put(nums[i],fmap.getOrDefault(nums[i],0) + 1);
        }

        int count = 0;
        for(int key: fmap.keySet()) {
            int freq = fmap.get(key);

            if(k == 0) {
                count = (k * (k - 1)) / 2;
            } else {
                int compFreq = fmap.getOrDefault(k + key,0);
                count  = count + freq * compFreq;
            }
        }

        return count;
    }
}
```

Maximum Width Ramp

(2, 0) => 2

(5, 1) => 4

(4, 3) => 1

0	1	2	3	4	5	6	7	8	9
2	0	14	38	10	9	27	12	20	18	14
38	38	38	27	27	27	20	20	18	14

i

j

=> Suffix Array

while (a[j] > a[i]) j++

(j - 1, i)

j is at a pos where a[j] < a[i]

Code for Max Width Ramp.

```java
class Solution {
    public int maxWidthRamp(int[] nums) {
        int [] suffix = new int[nums.length];

        int max = Integer.MIN_VALUE;

        for(int i=nums.length-1;i>=0;i--) {
            max = Math.max(nums[i],max);
            suffix[i] = max;
        }

        int i = 0;
        int j = 0;

        int maxDiff = Integer.MIN_VALUE;
        while(i < nums.length && j < nums.length) {

            while(j< nums.length && nums[i] <= suffix[j]) j++;
            int currDiff = (j-1)-i;
            maxDiff = Math.max(maxDiff,currDiff);
            i++;
        }

        return maxDiff;
    }
```

Static Sliding Window {Max Sum Subarray of size K} {GfG}

fixed sized window

100 200 300 400

0 1 2 3

$k = 2$

300

300 - 100 + 300 = 500

500 - 200 + 400 = 700

Take a window of size 2. When sliding it forward, remove the last ele of prev window & add the new element to the sum.

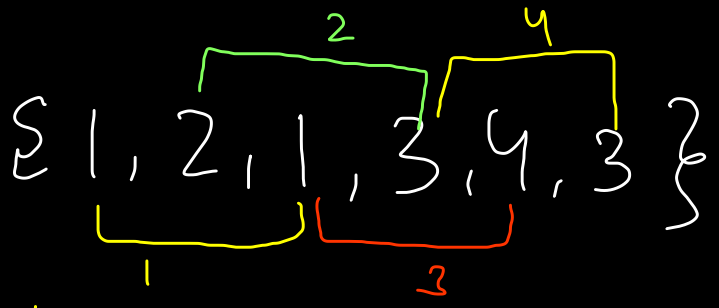
Code for Maximum Sum Subarray.

```java
static int maximumSumSubarray(int K, ArrayList<Integer> Arr,int N){
    // code here
    if(K > N) return 0;

    int sum = 0;
    for(int i=0;i<K;i++) {
        sum += Arr.get(i);
    }

    int maxSum = sum;
    for(int i=1;i<=N-K;i++) {
        sum = sum - Arr.get(i-1) + Arr.get(i+K-1);
        maxSum = Math.max(maxSum,sum);
    }

    return maxSum;
}
```

O(N) Time

O(1) Space

Distint Numbers in Window {IB}



k = 3

① 1-2, 2-1 } s = 2

② 2-1, 1-1, 3-1 } s = 3

③ 1-1, 3-1, 4-1 } s = 3

④ 3-2, 4-1 } s = 2

Total No of windows = N-k+1

** Take a HashMap for each window (freq Map). The size of the HashMap will be the no of unique eles in that window.

Code for distinct Nos in a Window.

```java
public class Solution {
    public int[] dNums(int[] A, int B) {
        HashMap<Integer,Integer> hm = new HashMap<>();
        int N = A.length;
        int K = B;

        for(int i=0;i<K;i++) {
            hm.put(A[i],hm.getOrDefault(A[i],0) + 1);
        }

        int[] ans = new int[N-K+1];
        ans[0] = hm.size();

        for(int i=1;i<=N-K;i++) {
            int oFreq = hm.get(A[i-1]);
            if(oFreq == 1) {
                hm.remove(A[i-1]);
            } else {
                hm.put(A[i-1],oFreq-1);
            }
            hm.put(A[i+K-1], hm.getOrDefault(A[i+K-1],0) + 1);
            ans[i] = hm.size();
        }

        return ans;
    }
}
```

Minimum Swaps & K Together

Minimum swaps and K together

Medium Accuracy: 47.98% Submissions: 52092
Points: 4

Given an array **arr** of **n** positive integers and a number **k**. One can apply a swap operation on the array any number of times, i.e choose any two index i and j (i < j) and swap arr[i] , arr[j] . Find the **minimum** number of swaps required to bring all the numbers less than or equal to **k** together, i.e. make them a contiguous subarray.

k = 6

2 9 8 7 3 9 4 5 7 8 1

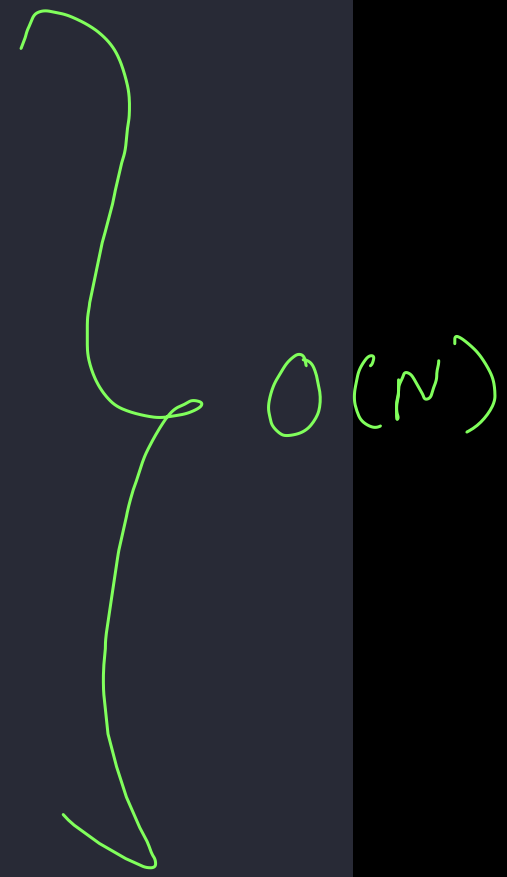
① Count total no of eles less than or equal to k. This will be our window size.

= 5 in this case

② Slide the fixed size window and every time count the no of eles less than equal to K in that window. The no of swaps req there will be Size of window − that count

Code for Min Swaps & K Together.

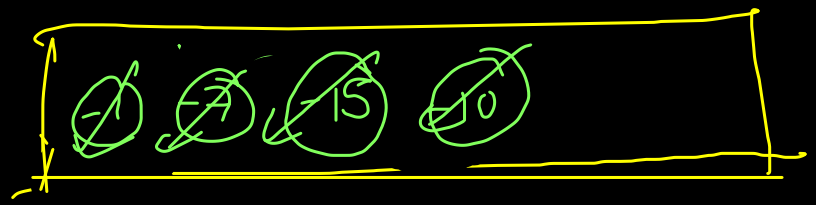
```java
// Function for finding maximum and value pair
public static int minSwap (int arr[], int n, int k) {
    //Complete the function
    int count = 0;
    for(int i=0;i<n;i++) {
        if(arr[i] <= k) count++;
    }

    int currNoOfEle = 0;
    for(int i=0;i<count;i++) {
        if(arr[i] <= k) currNoOfEle++;
    }

    int ans = count - currNoOfEle;

    for(int i=1;i<=n-count;i++) {
        if(arr[i-1] <= k) currNoOfEle--;
        if(arr[i + count -1] <= k) currNoOfEle++;
        int currAns = count - currNoOfEle;
        ans = Math.min(ans,currAns);
    }

    return ans;
}
```



First Negative Integer in Every Window of Size K [GfG]

0	1	2	3	4	5	6	7	8	9
12,	-1,	-7,	8,	-15,	-10,	30,	16,	28,	40

window = 4



Queue

-1, -1, -7, -15, -15, -10, 0

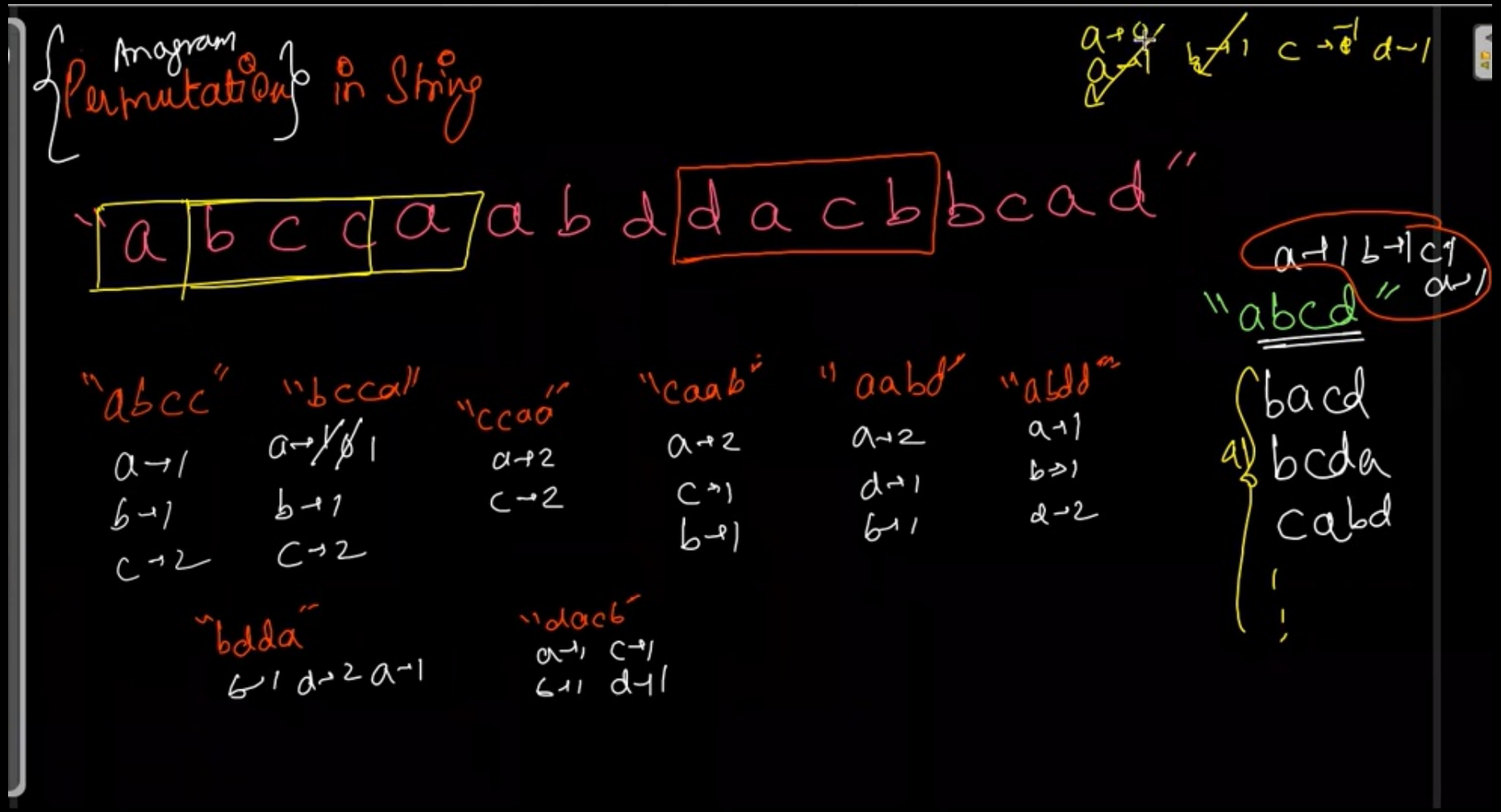
Code

```java
public long[] printFirstNegativeInteger(long A[], int N, int K)
{
    Queue<Integer> que = new ArrayDeque<>();
    for(int i=0;i<K;i++) {
        if(A[i] < 0) que.add(i);
    }

    long[] ans = new long[N-K+1];
    ans[0] = que.size() > 0 ? A[que.peek()] : 0;

    for(int i=1;i<=N-K;i++) {
        if(A[i-1] < 0) que.remove();
        if(A[i+K-1] < 0) que.add(i+K-1);
        if(que.size() == 0) ans[i] = 0;
        else ans[i] = A[que.peek()];
    }

    return ans;
}
```

O(N)

Permutation in String. LC 567

Anagram / Permutation in String



Code for Permutation in a String [LC 567]

```java
public boolean areEqual(int[] a, int [] b) {
    for(int i=0;i<26;i++) {
        if(a[i] != b[i]) return false;
    }
    return true;
}

public boolean checkInclusion(String s1, String s2) {
    int[] reqFreq = new int[26];
    int windowSize = s1.length();

    for(int i=0;i<s1.length();i++)
        reqFreq[s1.charAt(i) - 'a']++;

    int []currFreq = new int[26];
    for(int i=0;i<s2.length();i++) {
        currFreq[s2.charAt(i)- 'a']++;

        if(i >= windowSize) {
            currFreq[s2.charAt(i - windowSize) - 'a']--;
        }

        if(areEqual(reqFreq,currFreq) == true) {
            return true;
        }
    }

    return false;
}
```

$O(26 * N)$

Sliding Window Maximum

2 | 3 1 4 5 | 3 6 7

window = 4

4 5 5 6 7

1st window

4

2nd window

4,5

=> Deque

** peeche se bop karvana hai unsab ko jo humse chote hai jab hum insert no rahe hai.

2 3 1 4 | 5 3 6 7 |

3rd window

4,5,5

5th window

4,5,5,6,7

4th window

4,5,5,6

removefirst → out of window

removeLast → val < addval

addLast → always

deque ↗ chronological order
↘ decreasing order

Code for Sliding Window Maximum

```
ass Solution {
  public int[] maxSlidingWindow(int[] nums, int k) {
      int[] res = new int[nums.length - k + 1];
      Deque<Integer> q = new ArrayDeque<>();

      int window = 0;
      for(int i=0;i<nums.length;i++) {
          if(q.size() > 0 && q.getFirst() <= i-k) {
              q.removeFirst();
          }
          while(q.size() > 0 && nums[q.getLast()] < nums[i]) {
              q.removeLast();
          }

          q.addLast(i);
          if(i >= k - 1) {
              res[window++] = nums[q.getFirst()];
          }
      }

      return res;
  }
```

O(N)

Dynamic Window {Max Consecutive Ones : LC 485}

485. Max Consecutive Ones

Easy 2383 394 Add to List Share

Given a binary array `nums`, return *the maximum number of consecutive* `1`*'s in the array.*

{1, 1, 0, 1, 1, 1}

c = 0 1 2 0 1 2 3

mc = 0 2 3

```
public int findMaxConsecutiveOnes(int[] nums) {
    int count = 0;
    int maxCount = 0;

    int i = 0;
    while(i < nums.length) {
        count = 0;
        while(i<nums.length && nums[i] == 1) {
            count++;i++;
        }
        maxCount = Math.max(count,maxCount);
        i++;
    }

    return maxCount;
}
```

Maximum Consecutive Ones - III {LC: 1004}

```
= [0,0,1,1,0,0,1,1,1,0,1,1,0,0,0,1,1,1,1]
```

r

l

k = 3

count 0s = 1 2 3 4 3 4 3 4 5

left → exclude

right → include

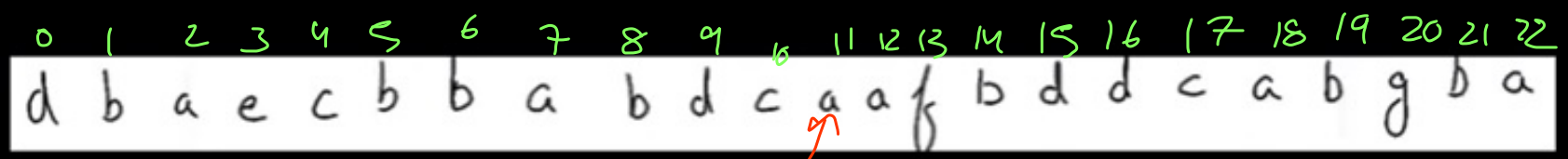
maxLen = 1 2 3 4 5 6 7 8 9 10 11

(r - l + 1)

Code for Max Consecutive Ones – III

```
public int longestOnes(int[] nums, int k) {
    int maxLen = 0, countOfZeroes = 0,left = 0;

    for(int right=0;right<nums.length;right++) {
        if(nums[right] == 0) {
            countOfZeroes++;
        }

        //make subarray valid by excluding left elements
        while(countOfZeroes > k) {
            if(nums[left] == 0) countOfZeroes--;
            left++;
        }

        maxLen = Math.max(maxLen,right - left + 1);
    }

    return maxLen;
}
```

Minimum Window Substring {LC Hard}



pattern

a b b c d c

"dbaecbbabdc" ⇒ cbbabdc

"bbabdcaafbddc"

a-1
b-2
c-2
d-1

d-1 2 1 2 3 2

b-1 2 3 4 minLen = 6 11 10 9 8 7

a-1 2 1 2 3 2 3 4

c-1 2 1 2 1

d-1 a-1

count = 0 1 2 3 4

of matching chars

3 4 3 4

```
ublic String minWindow(String s, String t) {
    if (t.equals("") == true || s.equals("") == true
            || s.length() < t.length())
        return "";

    HashMap<Character, Integer> req = new HashMap<>();
    for (int i = 0; i < t.length(); i++) {
        char ch = t.charAt(i);
        req.put(ch, req.getOrDefault(ch, 0) + 1);
    }

    HashMap<Character, Integer> curr = new HashMap<>();
    int matchCount = 0, l = 0;
    int idx = 0, len = Integer.MAX_VALUE;
```

```
    for (int r = 0; r < s.length(); r++) {
        char ch = s.charAt(r);
        int freq = curr.getOrDefault(ch, 0) + 1;
        curr.put(ch, freq);

        if (freq == req.getOrDefault(ch, 0)) {
            matchCount++;
        }

        while (matchCount >= req.size()) {
            if (matchCount >= req.size() && r - l < len) {
                idx = l;
                len = r - l + 1;
            }

            char chl = s.charAt(l);
            int freql = curr.get(chl) - 1;
            curr.put(chl, freql);

            if (freql + 1 == req.getOrDefault(chl, 0)) {
                matchCount--;
            }
            l++;
        }
    }

    if (len == Integer.MAX_VALUE)
        return "";
    return s.substring(idx, idx + len);
}
```

Longest Substring Without Repeating Characters {LC: 3}

a b b a c b c d b a d b d b b d c b

(Q3) (leetcode)

"Longest substring without repeating characters"

longest
Valid → Include
Invalid → Exclude

max = 4 (5)

Hashmap

a → 1
b → 1
c → 1
d → 1
e → 1

Code for Longest Substring Without Repeating Chars.

```java
public int lengthOfLongestSubstring(String s) {
    int left = 0, maxLen = 0;
    HashMap<Character,Integer> freq = new HashMap<>();
    for(int right=0;right < s.length(); right++) {
        char ch = s.charAt(right);
        freq.put(ch,freq.getOrDefault(ch,0) + 1);

        while(freq.get(ch) > 1) {
            char chl = s.charAt(left);
            freq.put(chl,freq.get(chl)-1);
            left++;
        }

        maxLen = Math.max(maxLen,right-left + 1);
    }

    return maxLen;
}
```

Count Substring Without Repeating Character

r

a	b	b	a	c	b	c	d	b	a	d	b	d	b	b	d	c	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

l

count += (r - l + 1)

maxLen = 0 2 3 4

a - r 0 1 2 0

d - r 2 4 5 6 0 1

b - r 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

c - r 2 4 6 0 1

Code for count Substring W/o Repeating character

```java
public static int solution(String s) {
    // write your code here
    int left = 0, count = 0;
    HashMap<Character,Integer> freq = new HashMap<>();
    for(int right=0;right < s.length(); right++) {
        char ch = s.charAt(right);
        freq.put(ch,freq.getOrDefault(ch,0) + 1);

        while(freq.get(ch) > 1) {
            char chl = s.charAt(left);
            freq.put(chl,freq.get(chl)-1);
            left++;
        }

        count += (right - left + 1);
    }

    return count;
}
```

Longest Substring with atmost K Unique Characters

count

d d a c b b a c c d e d a c e b b

0 1 2 3 4 5 6 2 8 9 10 11 12 13 14 15 16

l r

k = 3 (atmost 3 unique chars allowed)

a → 1 2 3 4 5 6 7

b → 1 2 3 4 5

c → 1 2 3 4 5 6 7

d → 1 2 3 4 5 6 7 8

e → 1 2 3

maxlen = 0 1 2 3 4 5 6 7

count = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65

** Previous ques are same with k = 1

```java
public static int solution(String str, int k) {
    // write your code here
    HashMap<Character,Integer> fmap = new HashMap<>();
    int maxLen = 0,left = 0;

    for(int right=0;right < str.length();right++) {
        char ch = str.charAt(right);
        fmap.put(ch,fmap.getOrDefault(ch,0) + 1);

        while(fmap.size() > k) {
            char chl = str.charAt(left);
            int oFreq = fmap.get(chl);
            if(oFreq == 1) fmap.remove(chl);
            else fmap.put(chl,oFreq - 1);
            left++;
        }

        maxLen = Math.max(maxLen,right - left + 1);
    }
    return maxLen;
}
```

maxLength

MaxLen is updated when we have <= K unique characters

```java
public static int solution(String str, int k) {
    // write your code here
    HashMap<Character,Integer> fmap = new HashMap<>();
    int count = 0,left = 0;

    for(int right=0;right < str.length();right++) {
        char ch = str.charAt(right);
        fmap.put(ch,fmap.getOrDefault(ch,0) + 1);

        while(fmap.size() > k) {
            char chl = str.charAt(left);
            int oFreq = fmap.get(chl);
            if(oFreq == 1) fmap.remove(chl);
            else fmap.put(chl,oFreq - 1);
            left++;
        }

        count += (right - left + 1);
    }
    return count;
}
```

Count

Longest } Exactly K unique characters
count }

We have to update maxLen when we have exactly K unique characters.

```java
public int longestkSubstr(String str, int k) {
    // code here
    HashMap<Character,Integer> fmap = new HashMap<>();
    int maxLen = 0,left = 0;

    for(int right=0;right < str.length();right++) {
        char ch = str.charAt(right);
        fmap.put(ch,fmap.getOrDefault(ch,0) + 1);

        while(fmap.size() > k) {
            char chl = str.charAt(left);
            int oFreq = fmap.get(chl);
            if(oFreq == 1) fmap.remove(chl);
            else fmap.put(chl,oFreq - 1);
            left++;
        }

        if(fmap.size() == k) maxLen = Math.max(maxLen,right - left + 1);

    }
    if(maxLen == 0) return -1;
    return maxLen;
}
```

⇒ Longest K Unique Chars Substring (GFG)

Count Exact K Unique Subarrays with K diff Integers {LC: 992}

d	d	a	c	b	b	a	c	c	d	e	d	a	c	e	b	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

⇒ count += (right - left + 1) will not work as it gives ans wor for atmost K

⇒ count++; will not work as we can have more than 1 substring

⇒ So, we will use the formula

Exactly K = Atmost(K) − Atmost(K-1)

Code for Subarrays with K Diff Integers.

```
class Solution {
    public int subarraysWithAtmostKDistinct(int[] nums, int k) {
        HashMap<Integer,Integer> fmap = new HashMap<>();
        int count = 0,left = 0;

        for(int right=0;right < nums.length;right++) {
            int ch = nums[right];
            fmap.put(ch,fmap.getOrDefault(ch,0) + 1);

            while(fmap.size() > k) {
                int chl = nums[left];
                int oFreq = fmap.get(chl);
                if(oFreq == 1) fmap.remove(chl);
                else fmap.put(chl,oFreq - 1);
                left++;
            }

            count += (right - left + 1);
        }
        return count;
    }
    public int subarraysWithKDistinct(int[] nums, int k) {
        return subarraysWithAtmostKDistinct(nums,k) - subarraysWithAtmostKDistinct(nums,k-1);
    }
}
```

Count K Odds { LC Count No of Nice Subarrays }

{ Return the count of those subarrays that have exactly K odd eles }

Count of exactly K odds = Count of Atmost K odds − Count of Atmost K-1 odds

Do not Insert Ever nos.

Count No of Nice Subarrays {LC 1248}

```java
public int numberOfSubarraysWithAtmost(int[] nums, int k) {
    int left = 0, ans = 0, odd = 0;

    for(int right = 0; right<nums.length;right++) {
        if(nums[right] % 2 == 1) odd++;

        while(odd > k) {
           if(nums[left] % 2 == 1) {
               odd--;
           }

           left++;
        }

        ans += (right - left + 1);
    }

    return ans;
}

public int numberOfSubarrays(int[] nums, int k) {
   return numberOfSubarraysWithAtmost(nums,k) - numberOfSubarraysWithAtmost(nums,k-1);
}
```

Longest Repeating Character Replacement {LC 424}

d d a c b b a c c d e d a c e b b

(l pointer under "e", r pointer above the last "b")

Dry Run for char 'A'

K = 3

O(26N)

Apply 2 pointer 26 times.

maxLen = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5 ⟹ maxLen = 5 for 'A'

no of chs changed = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~3~~ ~~4~~ ~~3~~ ~~4~~ ~~3~~ ~~4~~ ~~3~~ ~~4~~ ~~3~~ ~~4~~ ~~3~~ ~~4~~ ~~3~~ ~~4~~ ~~3~~ ~~4~~ ~~3~~ 4 3

Code for Longest Repeating Character Replacement.

```java
public int helper(String nums, int k, char ch) {
    int maxLen = 0, replacement = 0, left = 0;

    for(int right = 0;right < nums.length();right++) {
        if(nums.charAt(right) != ch) replacement++;

        while(replacement > k) {
            if(nums.charAt(left) != ch) replacement--;
            left++;
        }

        maxLen = Math.max(maxLen, (right - left + 1));
    }

    return maxLen;
}

public int characterReplacement(String s, int k) {
    int ans = 0;
    for(int i=0;i<26;i++) {
        ans  = Math.max(ans, helper(s,k,(char)(i+'A')));
    }
    return ans;
}
```

Minimum Size Subarray Sum { LC : 209 }

Smallest Subarray with Sum $\geq$ target

target = 7

r

2 3 1 2 4 3

l

Sum = 0 2 5 6 8 6 10 7

6 9

minLen = 0 4

3 2

r++ => Invalid subarray (sum < target)

l++ -> smallest valid subarray.

Code for Minimum Size Subarray Sum

```
class Solution {
    public int minSubArrayLen(int k, int[] nums) {
        int left = 0, minLen = Integer.MAX_VALUE, sum = 0;

        for(int right=0;right<nums.length;right++) {
            sum += nums[right];

            while(sum >= k) {
                minLen = Math.min(minLen,right -left + 1);
                sum = sum- nums[left];
                left++;
            }

        }

        if(minLen == Integer.MAX_VALUE) return 0;
        return minLen;
    }
```

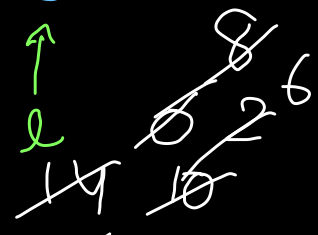
Maximum Sum SubArray $\leq K$

{ Gfg Max sum of subarray less than or equal to x }

{ Variation of Kadane's Algo }

target = 7

2 8 6 4 10 2 6



valid

right ++

sum <= target

Sum = ~~2~~ ~~10~~ 8 ~~6~~ 6 ~~10~~ 4

maxSum = ~~2~~ 6

invalid

left ++

sum > target

Code

```java
long findMaxSubarraySum(long arr[], int N,int X)
{
    // Your code goes here
    int left = 0;
    long maxSum = 0, sum = 0;

    for(int right = 0;right<N;right ++) {
        sum += arr[right];

        while(sum > X) {
            sum -= arr[left];
            left++;
        }

        maxSum = Math.max(maxSum,sum);
    }

    return maxSum;
}
```

Subarray Product less than K

{ Leetcode }

{ 10, 5, 2, 6, 3, 1 }

0 1 2 3 4 5

r ↓

l ↑

Product = 100

count = 0 1 2 5 6 11 15

product = 10 50 100 60 180 36

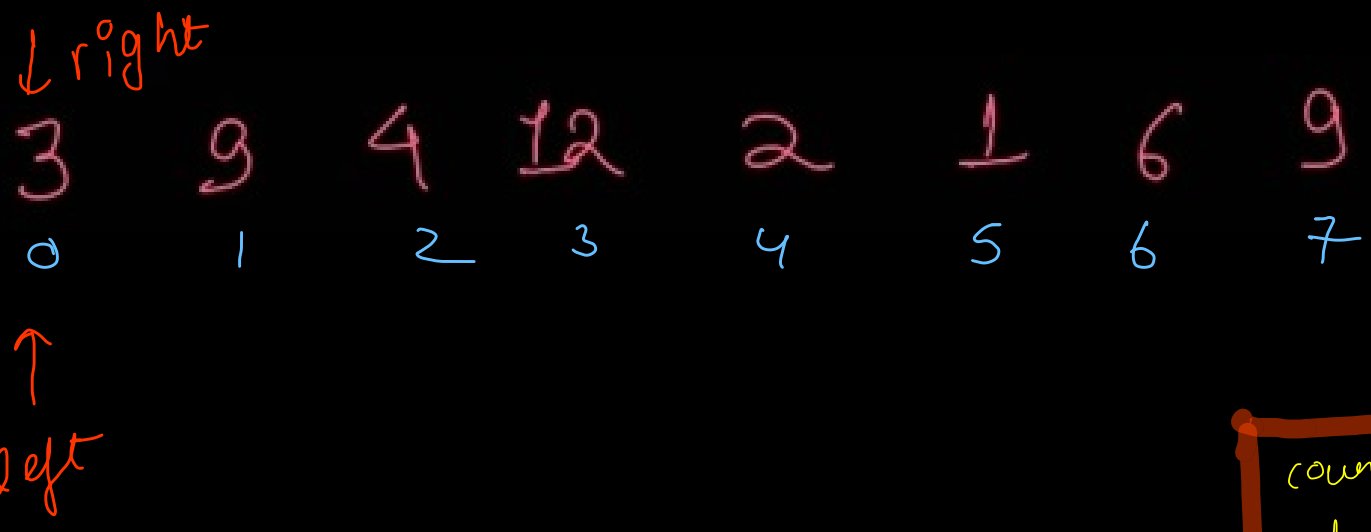
right ++ → Product < K

left ++ → Product ≥ K

Code

```
class Solution {
    public int numSubarrayProductLessThanK(int[] nums, int k) {
        int left =  0;
        int count = 0;
        int product = 1;

        for(int right=0;right<nums.length;right++) {
            product = product * nums[right];

            while(left < nums.length && product >= k) {
                product = product/nums[left];
                left++;
            }

            count += (right - left + 1);
        }

        if(count < 0) return 0;
        return count;
    }
```

Number of Subarrays with Bounded Maximum



count of subarrays with max in range [3,8] =

count of subarrays with max > 2 − with max greater than 8

invalid → Inclusion

valid → Exclusion