

Two pointer technique

Lecture ①

Two sum / Target sum pairs

- Unsorted
- Sorted
- Count
- Count Unique
- Two Matrices
- Design
- Closest
- Absolute
- Smaller
- Greater

3 sum

Target sum Triplets

→ Unique

→ Smaller

→ Closest

Count Valid Triplets

Valid Triangles

Max Sum Triplet

Minimize Differences

lecture ②

4 sum

→ 4 sum (I & II)

→ Count Tuples

→ Sum

→ Product

Difference Pairs

→ Target Diff Pair

→ All

→ Unique

→ longest Diff
Pair (I & II)

K sum

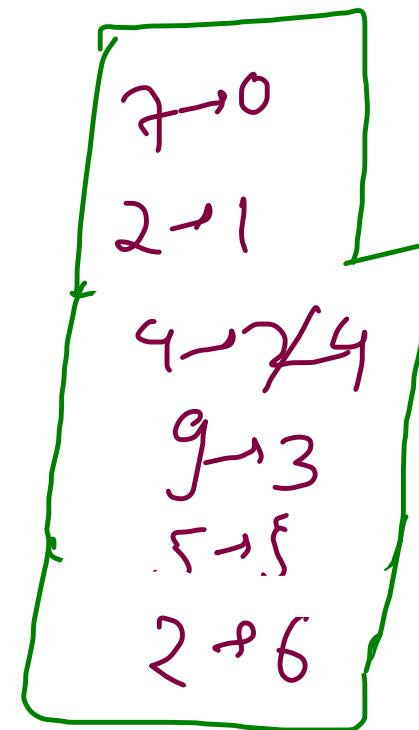
~~O(1)~~ Two Sum - Unsorted

target \Rightarrow 10

7 2 4 9 4 5 2 5
0 1 2 3 4 5 6 7

HashMap

Single Traversal



$O(n)$ Extra Space
 $O(n)$ Time

$\rightarrow O(n^2)$

Brute force

$\rightarrow O(r) T, O(n) S$

HashMap

$\rightarrow O(n \log n) T$

$O(1) S$

Two Pointers

```
public int[] twoSum(int[] nums, int target) {  
    HashMap<Integer, Integer> comp = new HashMap<>();  
  
    for(int i=0; i<nums.length; i++){  
        if(comp.containsKey(target - nums[i]) == true){  
            return new int[]{comp.get(target - nums[i]), i};  
        }  
        comp.put(nums[i], i);  
    }  
    return null;  
}
```

~~$\Theta(16^n)$~~
Two sum \rightarrow sorted

On time
 $O(1)$ extra space

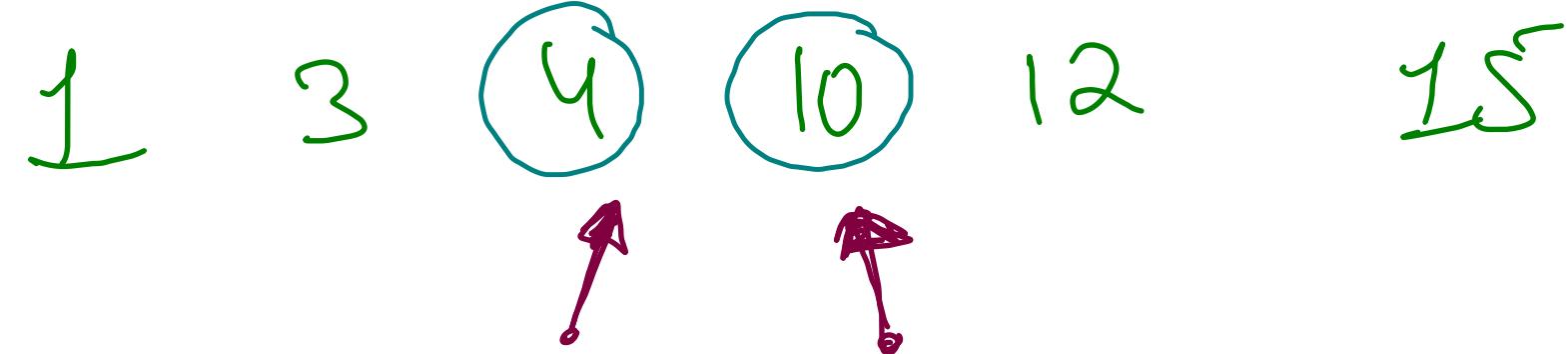
$$1 + 5 > 14$$

$$1 + 12 < 14$$

$$3 + 12 > 14$$

$$3 + 10 < 14$$

target = 14

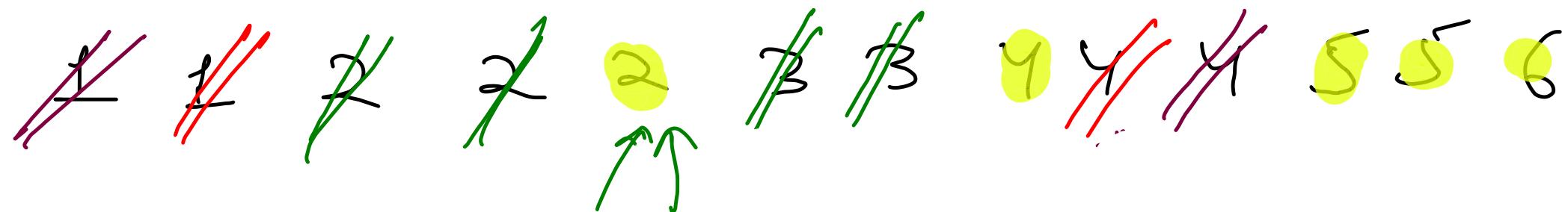


```
int left = 0, right = nums.length - 1;
while(left < right){
    int sum = nums[left] + nums[right];
    if(sum == target){
        return new int[]{left + 1, right + 1};
    } else if(sum > target){
        right--;
    } else {
        left++;
    }
}
return null;
```

~~Q1679~~

Target sum Pairs → Remove

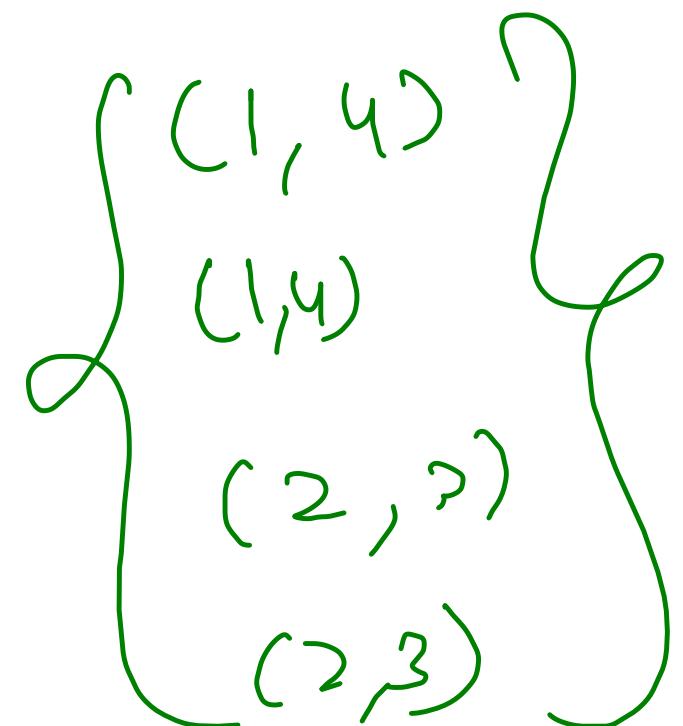
target = 5



Count of pairs = 0 / 1 / 2 / 3 / 4

$O(n \log n)$ Time

$O(1)$ Extra Space



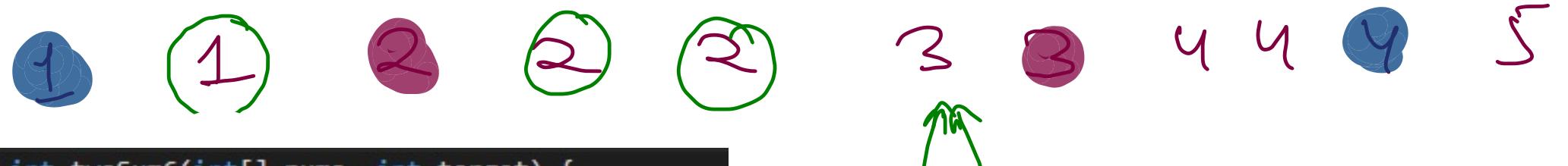
```
public int maxOperations(int[] nums, int target) {  
    Arrays.sort(nums); ← N log N  
    int left = 0, right = nums.length - 1;  
    int count = 0;  
    while(left < right){  
        int sum = nums[left] + nums[right];  
        if(sum == target){  
            count++;  
            left++; right--;  
        } else if(sum > target){  
            right--;  
        } else {  
            left++;  
        }  
    }  
    return count;  
}
```

$O(N)$

~~list code~~
~~C(87)~~

Unique Target Sum Pairs

target = 



```
public int twoSum6(int[] nums, int target) {
    Arrays.sort(nums);
    int left = 0, right = nums.length - 1;
    int count = 0;
    while(left < right){
        if(left > 0 && nums[left - 1] == nums[left]){
            left++; continue;
        }

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

    return count;
}
```

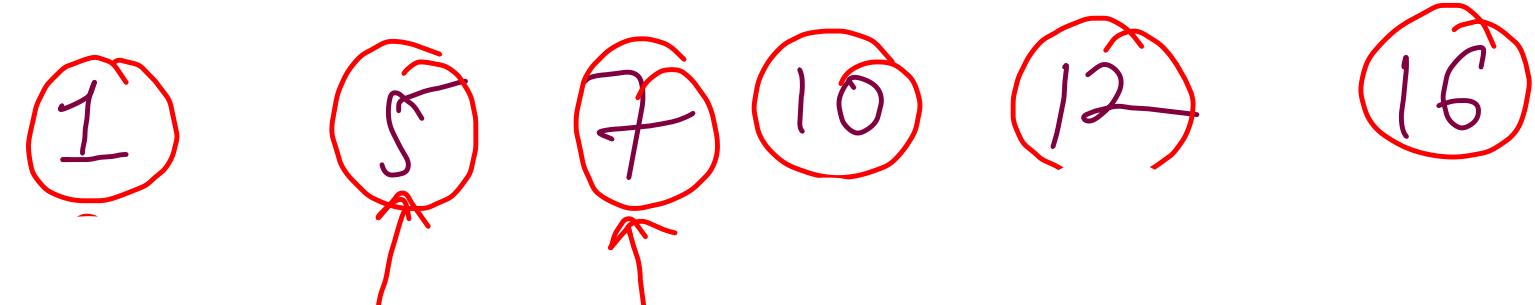
{1, 4}

{2, 3}

unique = 2

~~Binary~~ Two Sum - Closest

533



```

public int twoSumClosest(int[] nums, int target) {
    Arrays.sort(nums); → O(N log N)
    int left = 0, right = nums.length - 1;
    int abs = Integer.MAX_VALUE;
    while(left < right){
        int sum = nums[left] + nums[right];
        if(sum == target){
            return 0;
        } else if(sum > target){
            abs = Math.min(abs, sum - target);
            right--;
        } else {
            abs = Math.min(abs, target - sum);
            left++;
        }
    }
    return abs;
}

```

target = 14

ans = ~~+6~~ ~~+8~~ +1

$$|+6| = |7 - 14| = 3$$

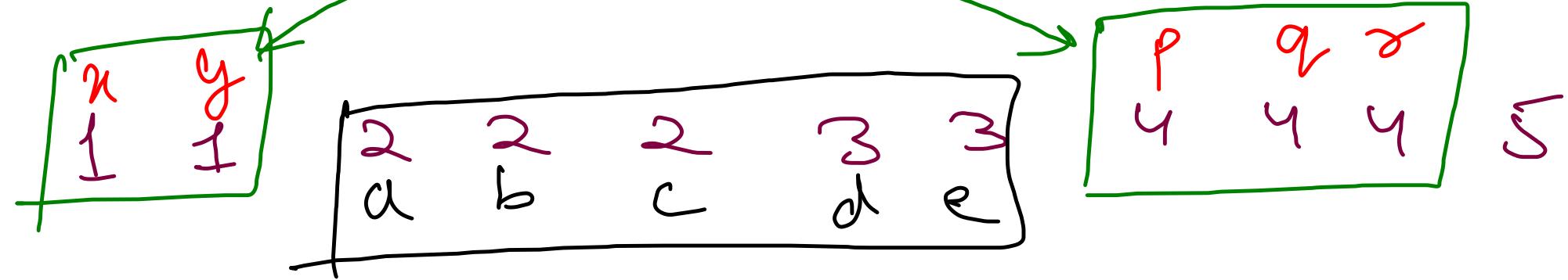
$$|+8| = |12 - 14| = 2$$

$$|+12| = |7 - 14| = 3$$

$$|+10| = |5 - 14| = 9$$

$$|+7| = |12 - 14| = 2$$

~~CFG~~
Target Sum Edit 11



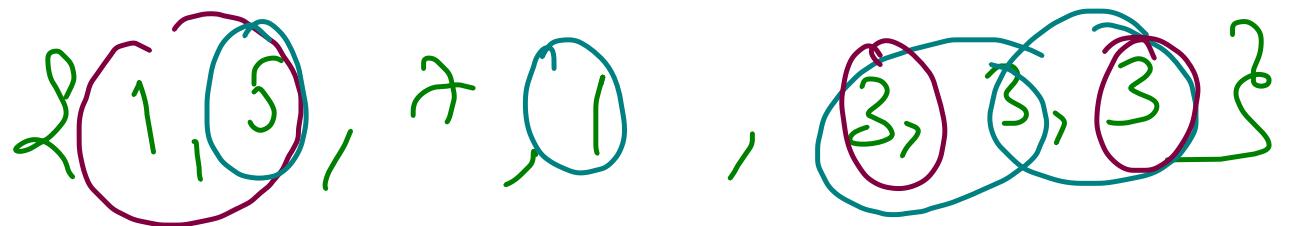
$(x,y)(x,y)(x,y)$ (a,d) (b,d) (c,d)

$(y,p)(y,q)(x,r)$ (a,e) (b,e) (c,e)

target = ~~y~~
y

$$1f * 4f = 2 * 3$$

$$2f * 3f = 3 * 2$$



target = 6

```

HashMap<Integer, Integer> hm = new HashMap<>(); {->O(N)}  

int count = 0;  

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

    int comp = target - arr[i];  

    int freq = hm.getOrDefault(comp, 0);  

    count += freq;  

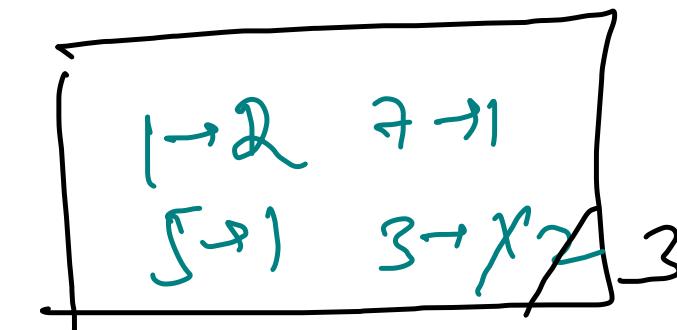
    hm.put(arr[i], hm.getOrDefault(arr[i], 0) + 1);
}  

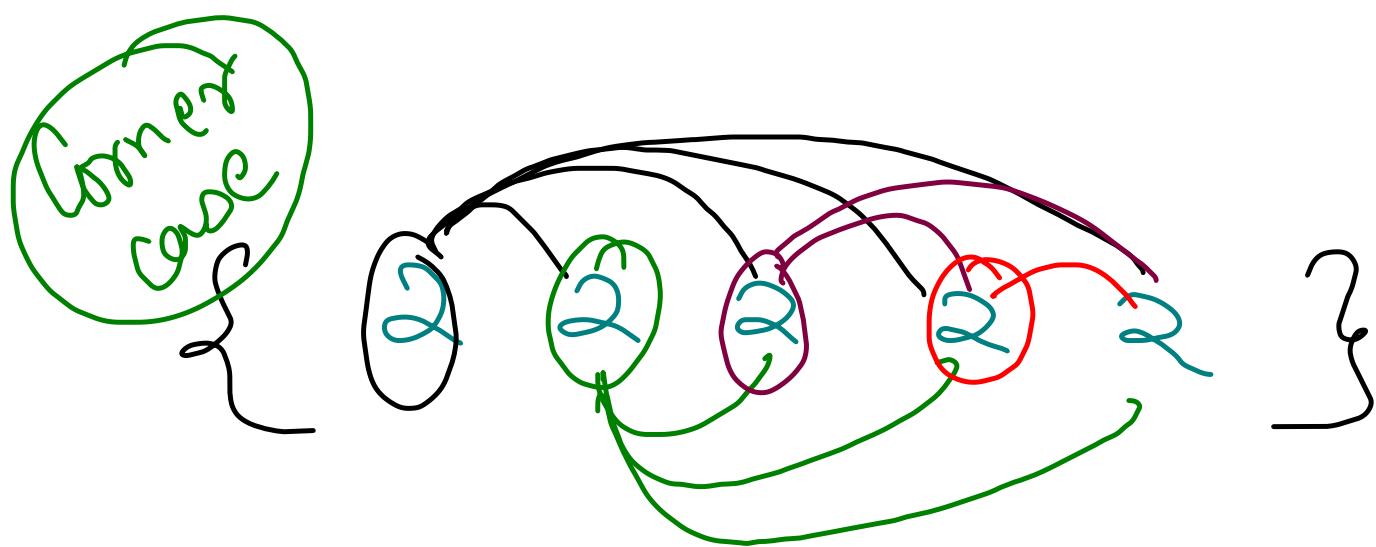
return count;

```

O(N) time



Count = ~~0~~ / ~~2~~ / ~~3~~ ~~5~~



complement
== de

target = 4

$$f[2] = 5$$

$$(n-1) + (n-2) + (n-3) + \dots + 1 =$$

$$\frac{n * (n-1)}{2}$$

Target sum bar
2 preceding

0	1	2
1	5	6
3	4	5
8	10	11
6	7	8
15	16	18

1 5 6 8 10 11 15 16 18

2D to 1D

idx → 1 (4/3)
idx → 1 (4/3)
Collision = $\Theta(n^2/cols)$

2	4	7
9	10	12
13	16	20

target = 21

2D to 1D

2 4 ≠ 9 10 12 13 16 20

16 20

{1, 20}

{5, 13}

{8, 13}

{11, 10}

609
intcode

2 sum - smaller or equal

1 3 4 10 12 15

target $\leq \textcircled{14}$
 $= 14$
 < 14

$$1 + 15 > 14$$

$$1 + 12 < 14 \Rightarrow \textcircled{4}$$

$$3 + 12 > 14$$

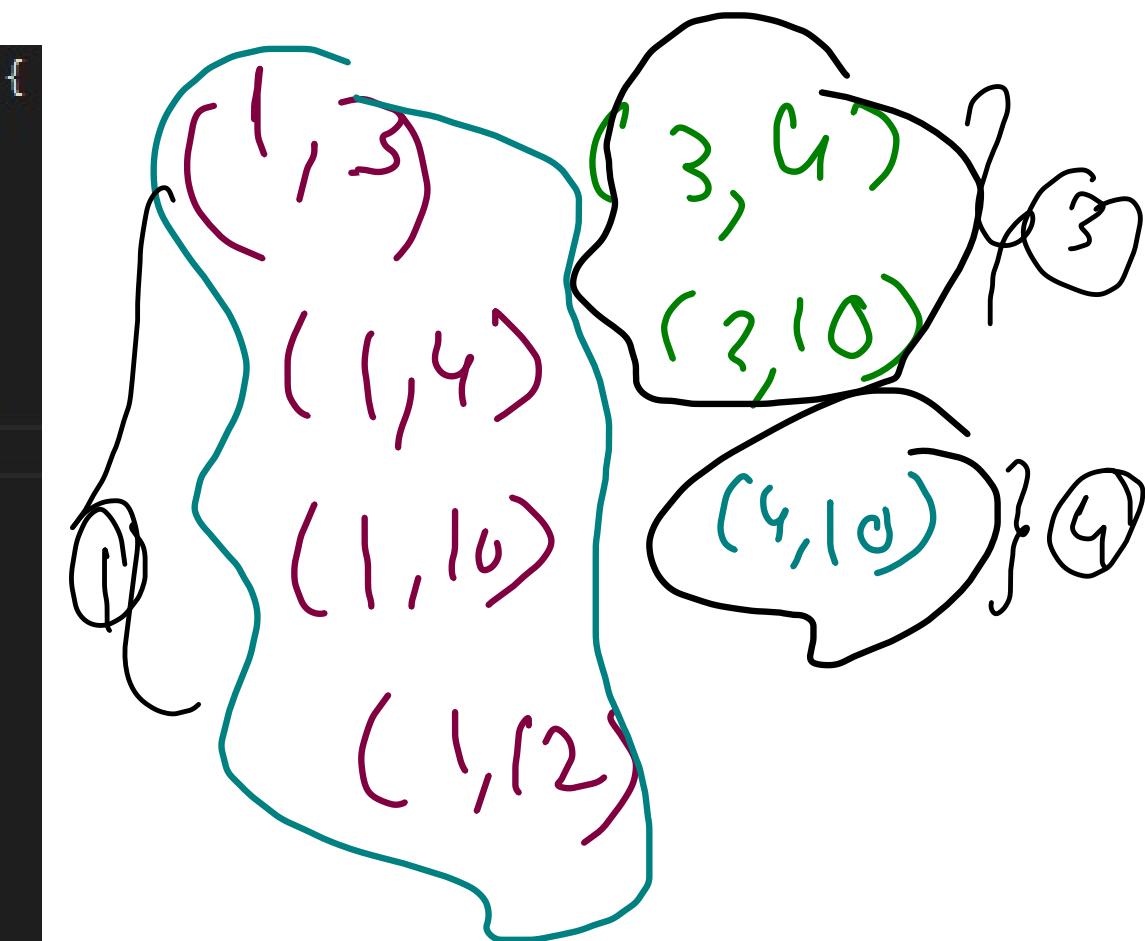
$$3 + 10 < 14 \Rightarrow \textcircled{2}$$

$$4 + 10 = 14 \Rightarrow \textcircled{1}$$

```
public int twoSum5(int[] nums, int target) {
    Arrays.sort(nums);
    int left = 0, right = nums.length - 1;

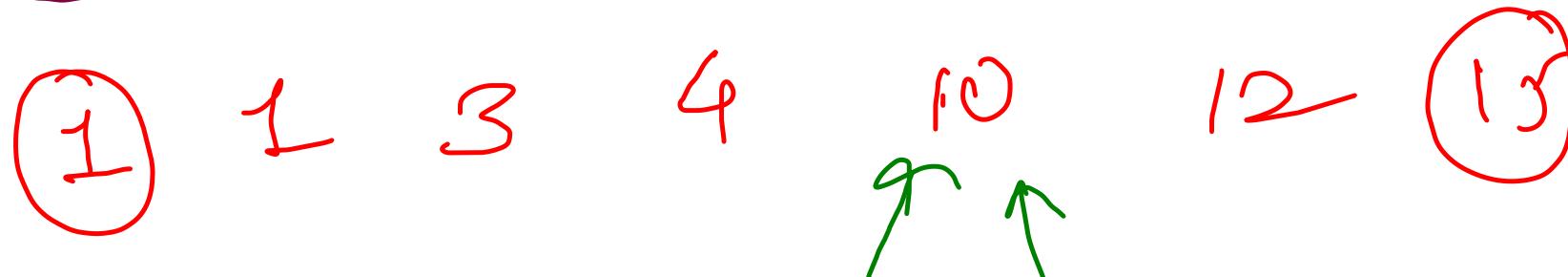
    int count = 0;
    while(left < right){
        int sum = nums[left] + nums[right];

        if(sum <= target){
            count += right - left;
            left++;
        } else {
            right--;
        }
    }
    return count;
}
```



443
hint code

2 Sum - Greater



```
Arrays.sort(nums);
int left = 0, right = nums.length - 1;

int count = 0;
while(left < right){
    int sum = nums[left] + nums[right];

    if(sum <= target){
        left++;
    } else {
        count += right - left;
        right--;
    }
}

return count;
```

target > 14

(1, 15) }
(3, 15) } 15
(4, 15) }
(10, 15) }
(12, 15) } 12
(3, 12) }
(4, 12) }
(10, 12) }

Two sum - Design / 8beam

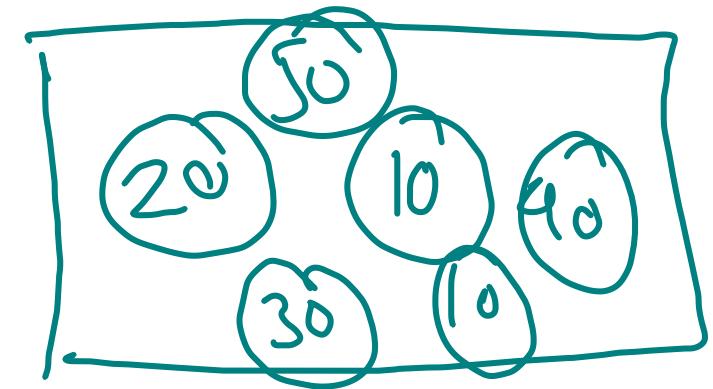
Online

Algorithm

Design and implement a TwoSum class. It should support the following operations: add and find.

add - Add the number to an internal data structure.

find - Find if there exists any pair of numbers which sum is equal to the value.



① add(20)

⑥ find(50) = True

② find(40) = False

⑦ find(40) = True

③ add(10)

⑧ add(10) ⑩ find(90) = True

④ add(30)

⑨ add(50)

⑤ add(40)

1

ArrayList

{ add }

① Add last } $\Rightarrow O(n \log n)$
② Sort

{ find }

Two pointers } $\Rightarrow O(N)$

2

ArrayList

add → add last $\Rightarrow O(1)$

find → if sorted → two pointers $\Rightarrow O(N)$

else → sort } $\Rightarrow O(n \log n)$
two pointers

```
public class TwoSum {  
    ArrayList<Integer> data;  
    boolean isSorted;  
  
    public TwoSum(){  
        data = new ArrayList<>();  
        isSorted = true;  
    }  
  
    public void add(int number) {  
        data.add(number);  
        isSorted = false;  
    }  
  
    public boolean find(int value) {  
        if(isSorted == false){  
            Collections.sort(data);  
            isSorted = true;  
        }  
  
        int left = 0, right = data.size() - 1;  
        while(left < right){  
            int sum = data.get(left) + data.get(right);  
            if(sum == value) return true;  
            if(sum < value) left++;  
            else right--;  
        }  
        return false;  
    }  
}
```

③ Ordered - Map

Self Balancing BST

add → add in order $\rightarrow O(\log n)$

find → Two pointers $\rightarrow O(N)$

Java code
↳ TreeMap
next(), hasNext()

map<int, int> arr;

void add(int number) {
 arr[number]++;
}

bool find(int value) {
 auto left = arr.begin();
 auto right = arr.end();
 right--;

while(left != right)
 {
 int sum = left->first + right->first;
 if(sum == value)
 return true;
 if(sum < value) left++;
 else right--;
 }

if(left->second > 1 && 2 * (left->first) == value)
 return true;
 return false;
}

C++ code

④ Hashmap of frequency

Add → frequency update

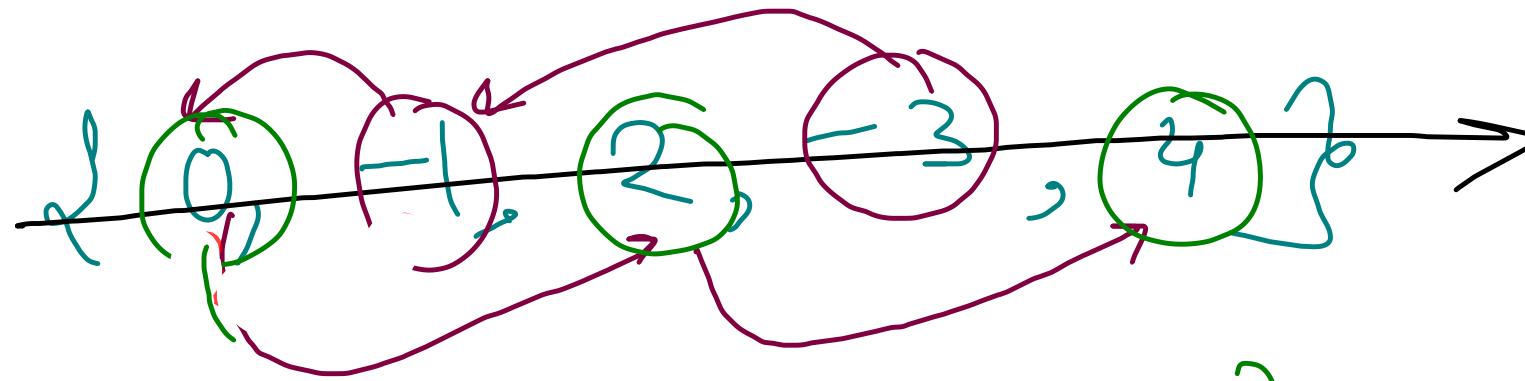
$O(1)$ avg case
 $O(N)$ worst-case
(collisions)

find → checking complement
by iterating on hashmap $\rightarrow O(N)$

Corner case → equal complement
 \downarrow
 $freq \geq 2$

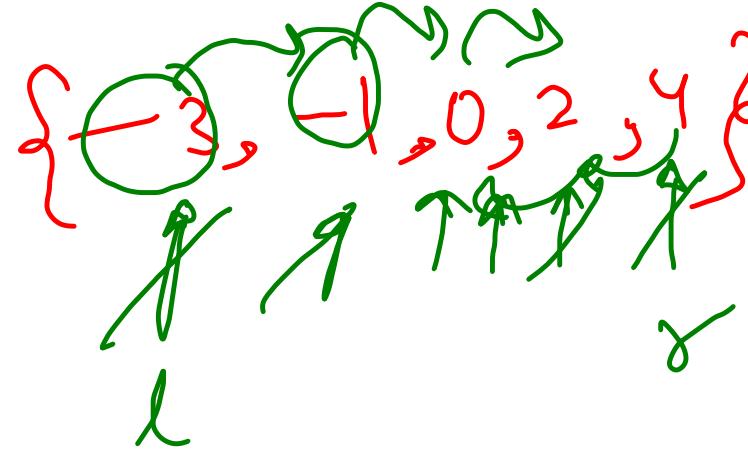
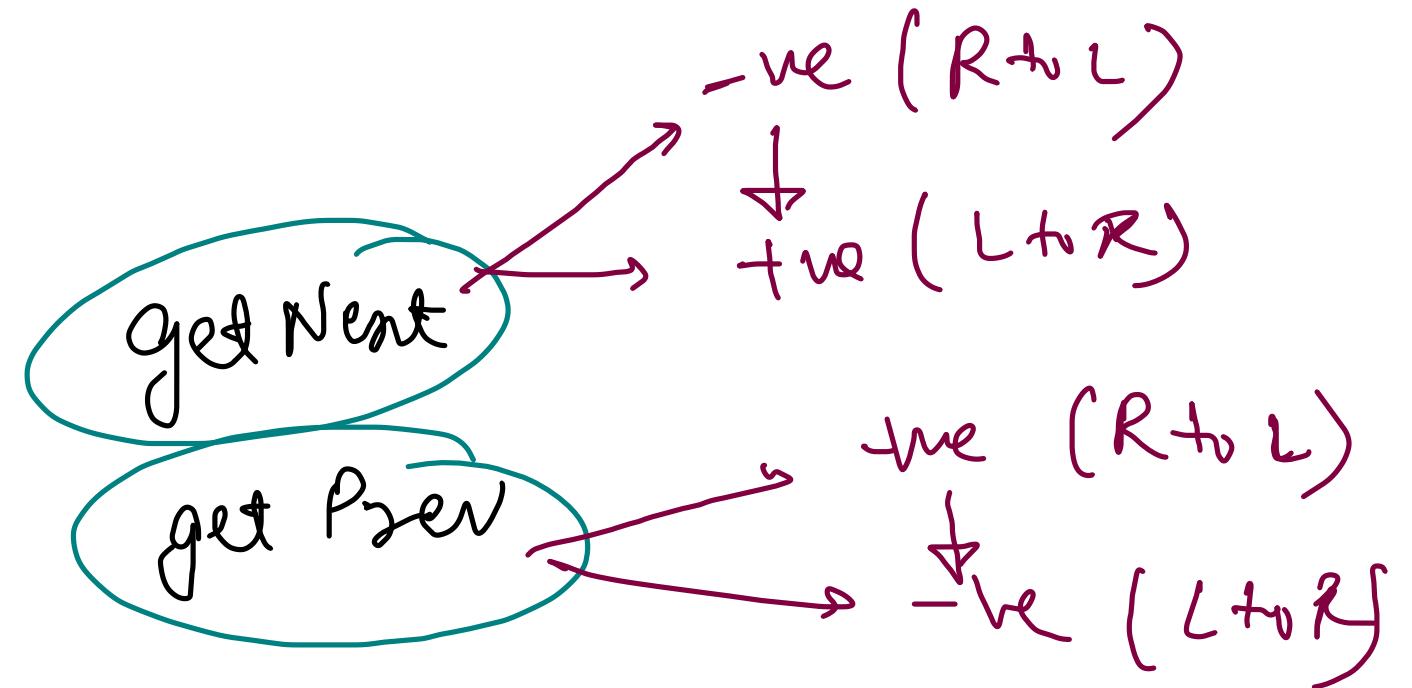
```
public class TwoSum {  
    HashMap<Integer, Integer> freq;  
  
    public TwoSum(){  
        freq = new HashMap<>();  
    }  
  
    public void add(int number) {  
        freq.put(number, freq.getOrDefault(number, 0) + 1);  
    }  
  
    public boolean find(int value) {  
        for(Integer key: freq.keySet()){  
            int comp = value - key;  
            int freq_comp = freq.getOrDefault(comp, 0);  
  
            if(value - key == key){  
                if(freq_comp >= 2) return true;  
            } else {  
                if(freq_comp >= 1) return true;  
            }  
        }  
        return false;  
    }  
}
```

Two Sum - Absolute (int code (8+9)) HW



int left = ?, right = ?

{3, 9} {1, 2}



3 sum

Target sum Triplets

- Unique
- Smaller
- Closest

Count Valid Triplets

Valid Triangles

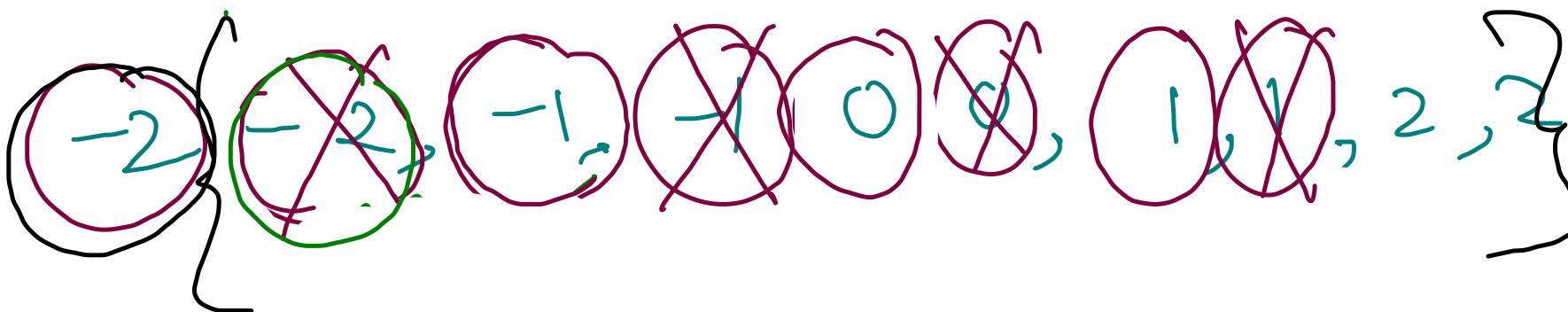
Max Sum Triplet

Minimize Differences

lecture ②

K sum

~~015~~ ~~3 sum - Unique~~



target = 0
 $x + y + z = 0$
-2

① Brute force : $\rightarrow O(n^3)$

② 3 pointers : \rightarrow Normal loop + Two pointer
 $O(n^2)$

{ -2, 0, 2 }

{ -1, -1, 2 }

{ -2, 1, 1 }

{ 1, 0, 1 }

{ Skip Repeated Elements }

in normal loop

{ first }

in two pointers { yes }

```

public List<List<Integer>> twoSum(int[] nums, int left, int right, int target){
    List<List<Integer>> ans = new ArrayList<>();
    int start = left;

    while(left < right){
        if(left > start && nums[left - 1] == nums[left]){
            left++; continue;
        }

        int sum = nums[left] + nums[right];

        if(sum == target){
            List<Integer> pair = new ArrayList<>();
            pair.add(nums[left]);
            pair.add(nums[right]);
            ans.add(pair);

            left++; right--;
        } else if(sum < target){
            left++;
        } else {
            right--;
        }
    }

    return ans;
}

```

```

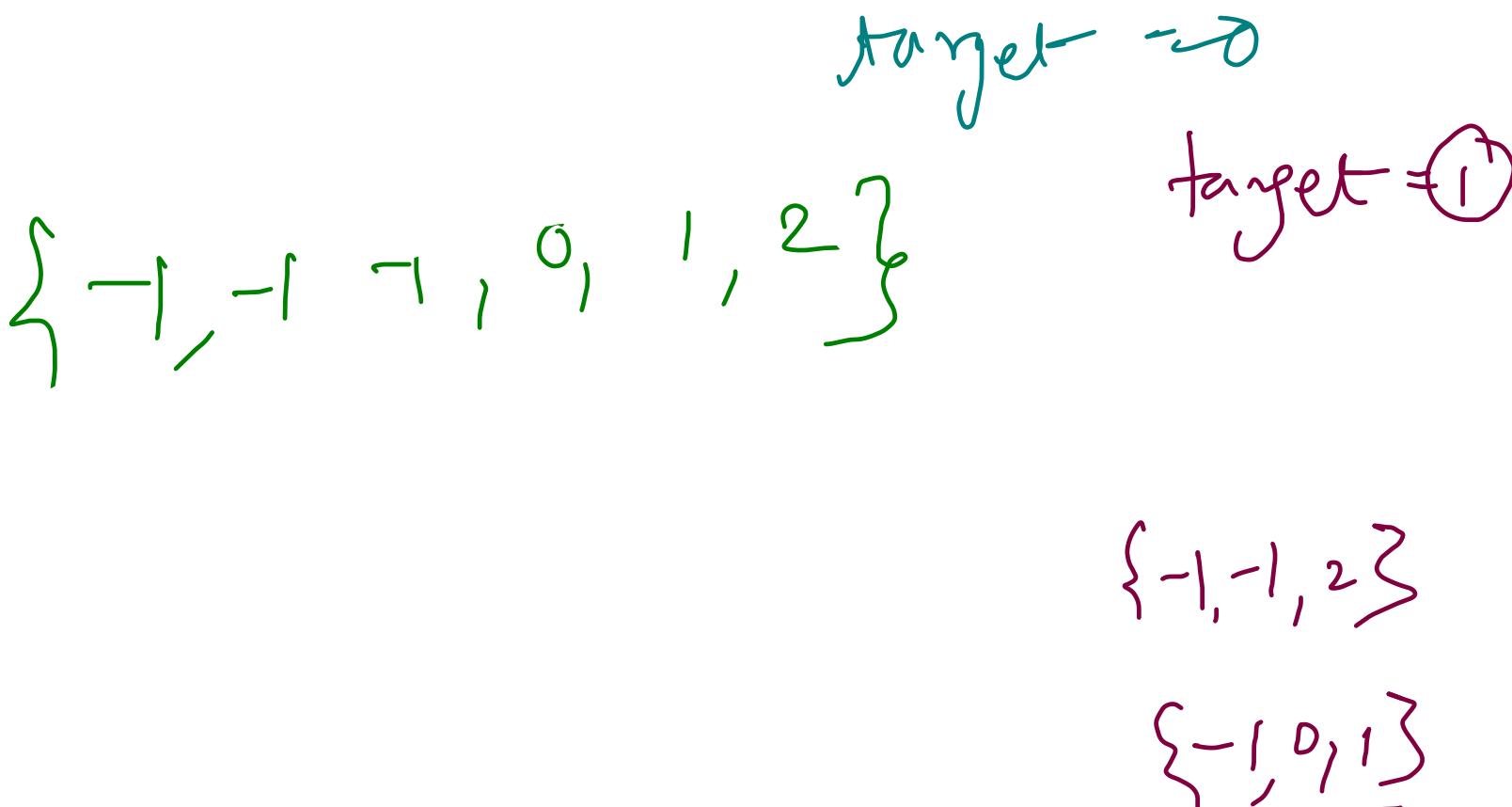
public List<List<Integer>> threeSum(int[] nums) {
    Arrays.sort(nums);
    List<List<Integer>> ans = new ArrayList<>();

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

        List<List<Integer>> pairs = twoSum(nums, first + 1, nums.length - 1, -nums[first]);
        for(List<Integer> triplet: pairs){
            triplet.add(0, nums[first]);
            ans.add(triplet);
        }
    }

    return ans;
}

```



OG18 (Unsorted)
 \rightarrow sum smaller

$-2 \rightarrow 1, 2, 3$

Normal loop

+
Two smaller \rightarrow smaller

sum ≤ 2

$$x+y+3 \leq 2$$

-2

$$\boxed{y+3 \leq 4}$$

$$(-2)(0, 3)$$

$$(-2)(0, 2)$$

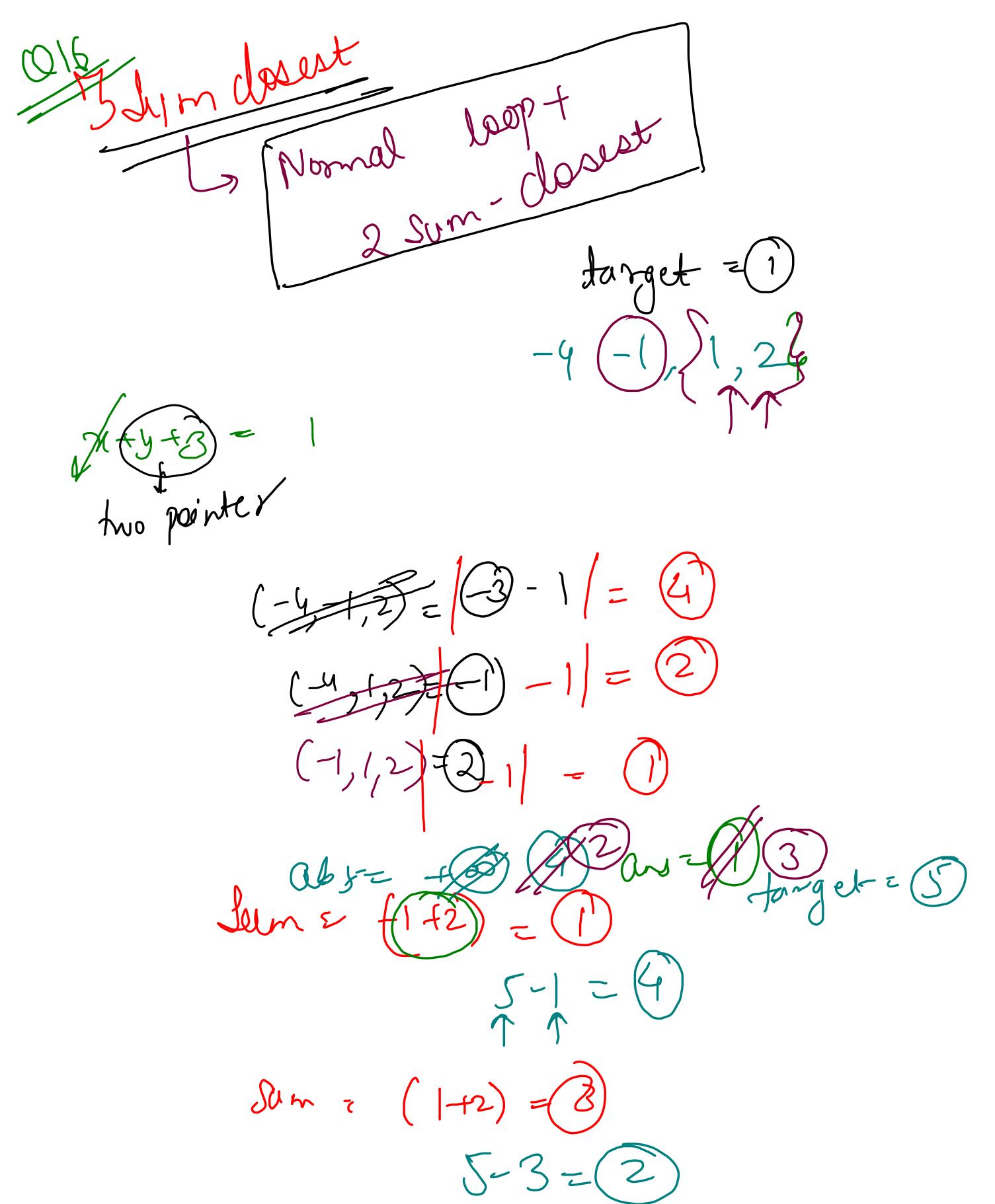
$$(-2)(0, 1)$$

$$(-2)(1, 2)$$

```
public int twoSumSmaller(int[] nums, int left, int target) {  
    int right = nums.length - 1;  
  
    int count = 0;  
    while(left < right){  
        int sum = nums[left] + nums[right];  
  
        if(sum < target){  
            count += right - left;  
            left++;  
        } else {  
            right--;  
        }  
    }  
  
    return count;  
}
```

$$O(N^2)$$

```
public int threeSumSmaller(int[] nums, int target) {  
    Arrays.sort(nums);  
    int ans = 0;  
    for(int f=0; f<nums.length; f++){  
        ans += twoSumSmaller(nums, f + 1, target - nums[f]);  
    }  
    return ans;  
}
```



```

public int twoSumClosest(int[] nums, int left, int target) {
    int right = nums.length - 1;
    int abs = Integer.MAX_VALUE;
    int ans = Integer.MAX_VALUE;

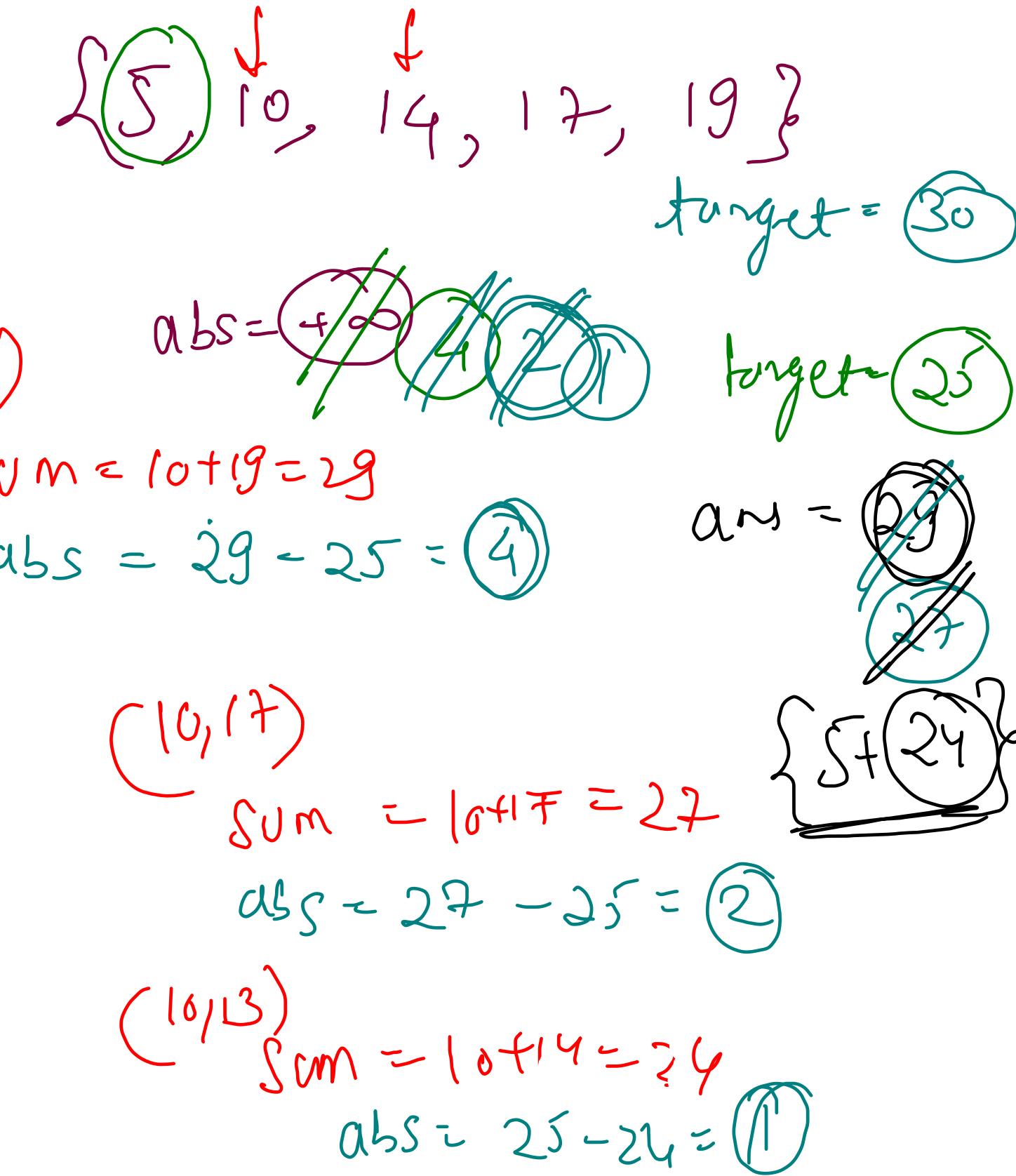
    while(left < right){
        int sum = nums[left] + nums[right];
        if(sum == target){
            return target;
        } else if(sum > target){
            if(sum - target < abs){
                abs = sum - target;
                ans = sum;
            }
            right--;
        } else {
            if(target - sum < abs){
                abs = target - sum;
                ans = sum;
            }
            left++;
        }
    }
    return ans;
}

```

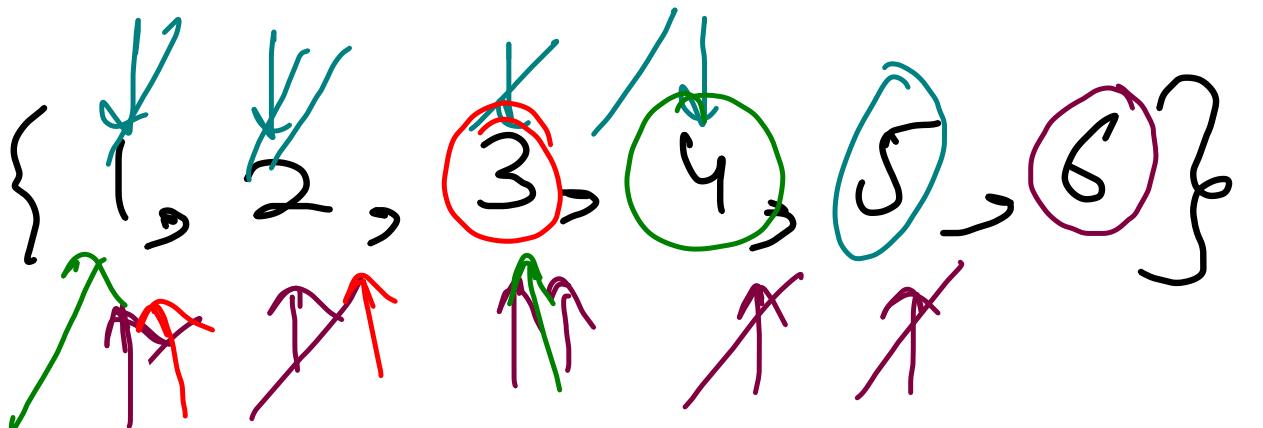
```

public int threeSumClosest(int[] nums, int target) {
    Arrays.sort(nums);
    int ans = 0;
    int abs = Integer.MAX_VALUE;
    for(int i=0; i<nums.length-2; i++){
        int curr = twoSumClosest(nums, i + 1, target - nums[i]) + nums[i];
        if(Math.abs(curr - target) < abs){
            abs = Math.abs(curr - target);
            ans = curr;
        }
    }
    return ans;
}

```



~~GFG~~ Count the triplets



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

$$\# \underbrace{a+b=c}_{\text{count triplets}} = \textcircled{c}$$

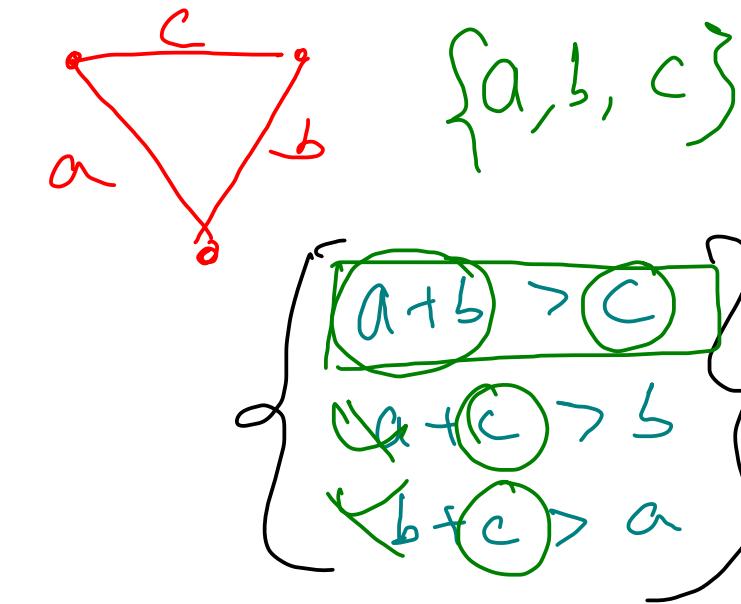
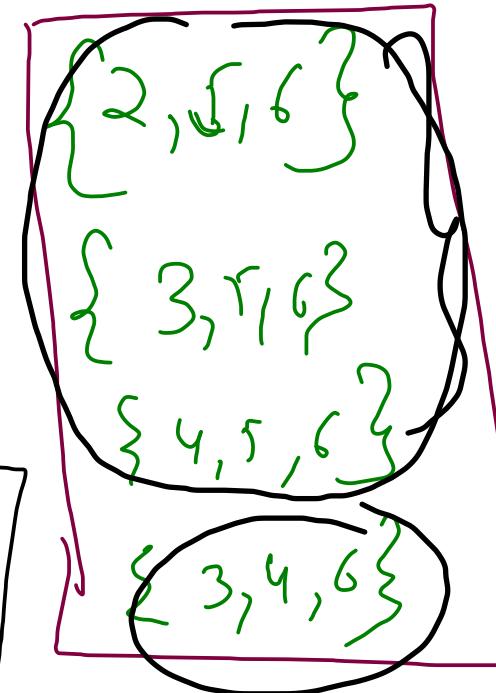
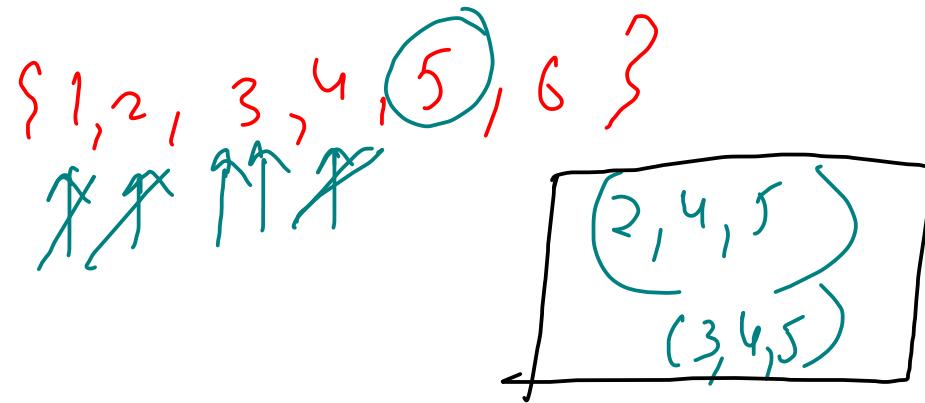
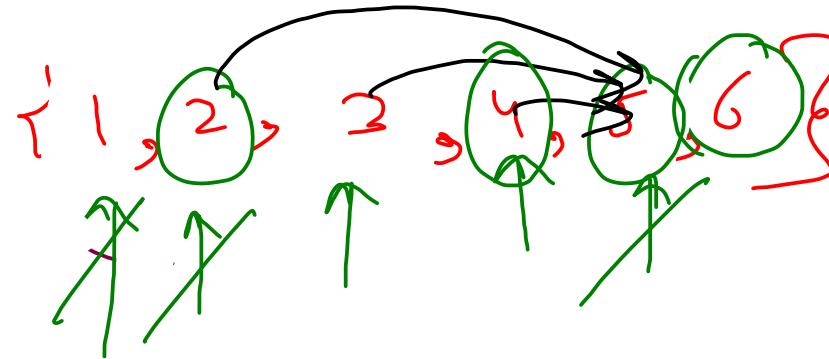
Find third ele { from
right to left }
& two pointers on
(a+b)

```
int twoSum(int[] arr, int left, int right, int target){  
    int count = 0;  
    while(left < right){  
        int sum = arr[left] + arr[right];  
  
        if(sum == target){  
            count++;  
            left++; right--;  
        } else if(sum < target){  
            left++;  
        } else {  
            right--;  
        }  
    }  
    return count;  
}  
int countTriplet(int arr[], int n) {  
    Arrays.sort(arr);  
    int count = 0;  
    for(int c=n-1; c>1; c--){  
        count += twoSum(arr, 0, c - 1, arr[c]);  
    }  
    return count;  
}
```

$O(n)$

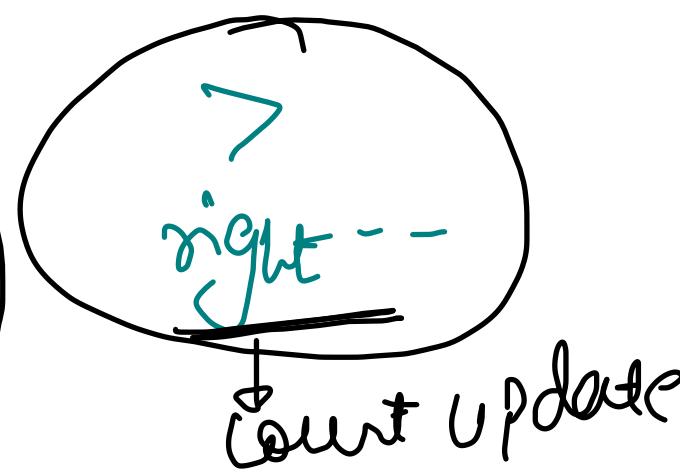
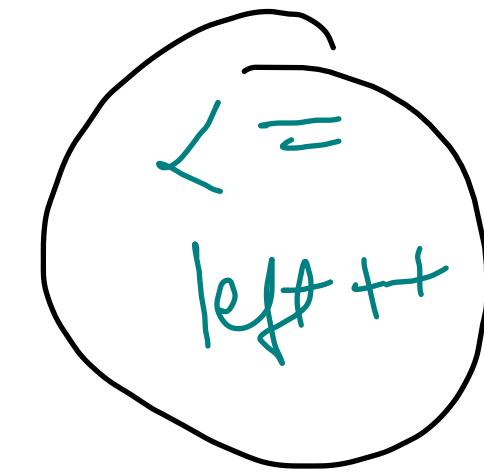
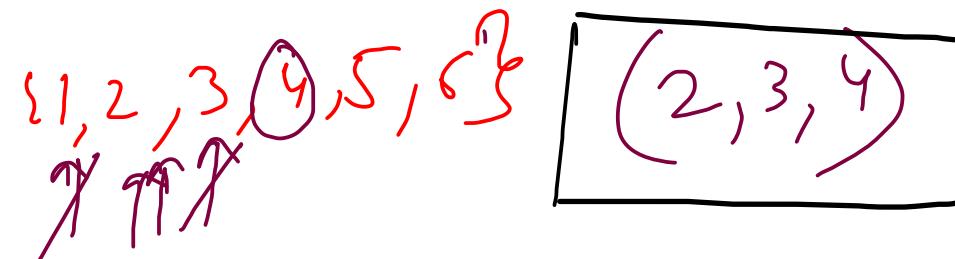
$O(n^2)$

Q611 Count Valid Triangles



$$a+b > c$$

Two Sum-Generator



```
public int twoSumGreater(int[] nums, int right, int target) {
    int left = 0;

    int count = 0;
    while(left < right){
        int sum = nums[left] + nums[right];

        if(sum <= target){
            left++;
        } else {
            count += right - left;
            right--;
        }
    }

    return count;
}
```

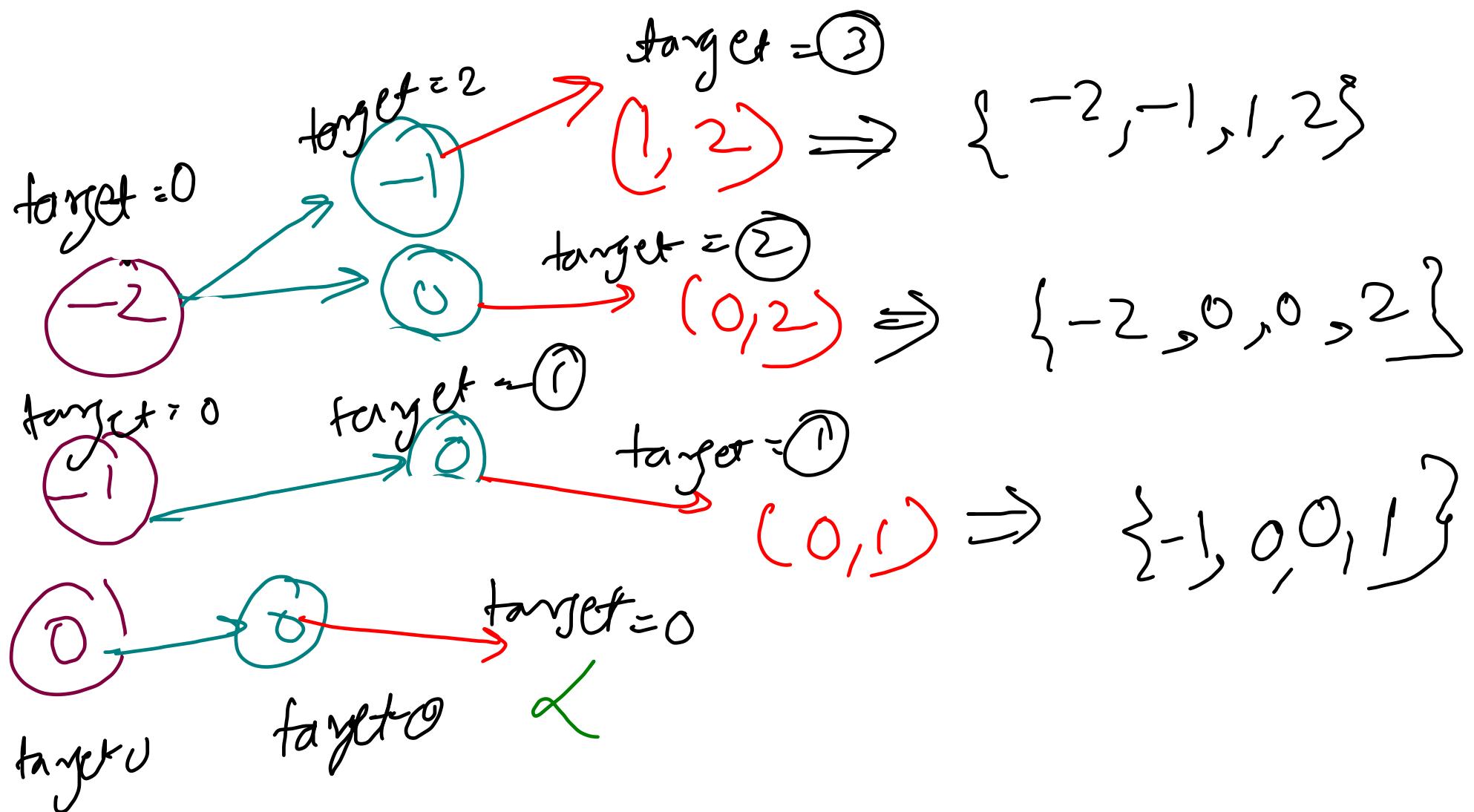
```
public int triangleNumber(int[] nums) {
    Arrays.sort(nums);
    int ans = 0;

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

    return ans;
}
```

$$\underline{\underline{4 \text{ sum}}} = \underline{\underline{\text{loop} + 3 \text{ sum}}} = \underline{\underline{\text{Nested loop} + \text{Two pointers}}} \quad a + b + c + d = 0$$

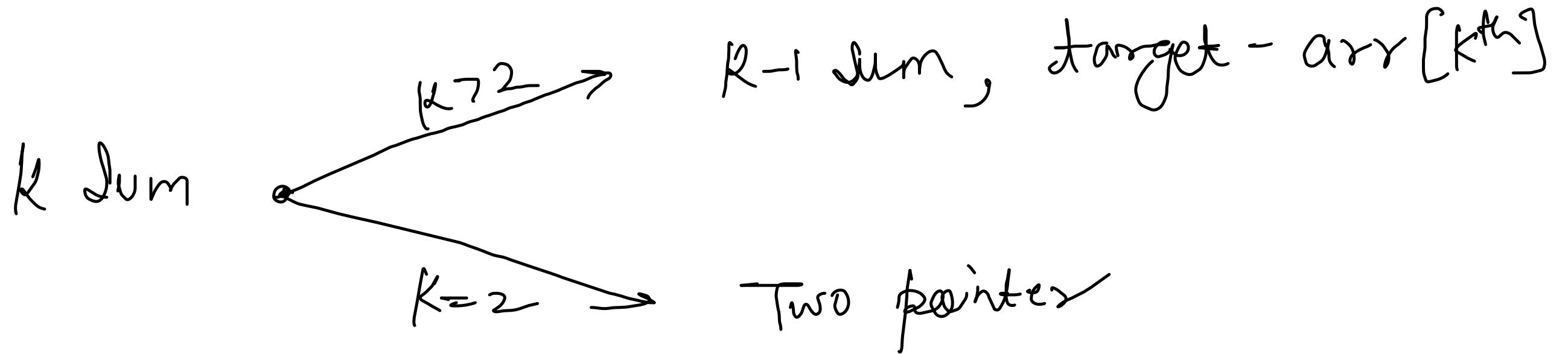
$\{-2, -1, \textcircled{0}, \textcircled{0}, 1, 2\}$



① Brute force $\Rightarrow O(n^4)$

② Nested loop + two pointers
 $\Rightarrow O(n^3)$

~~K-Sum~~ Recurrence Relation



$$O(k \text{sum}) = \begin{cases} O((k-1)\text{sum}) * n & k > 2 \\ O(n) & k = 2 \end{cases}$$
$$= O(n^{k-1})$$

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

(B) Array 3 pointers

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

$\max(\text{abs}(A[i] - B[j]), \text{abs}(B[j] - C[k]), \text{abs}(C[k] - A[i]))$

↑
minimum

{1, 2, 10}

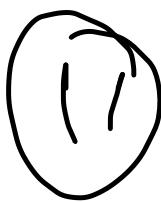
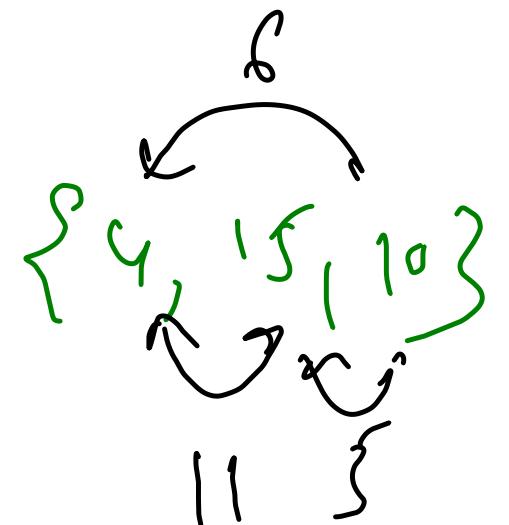
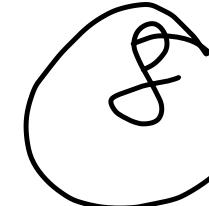
(1,2) (2,10) (1,10)

{1, 8, 9}
9

{4, 2, 10}

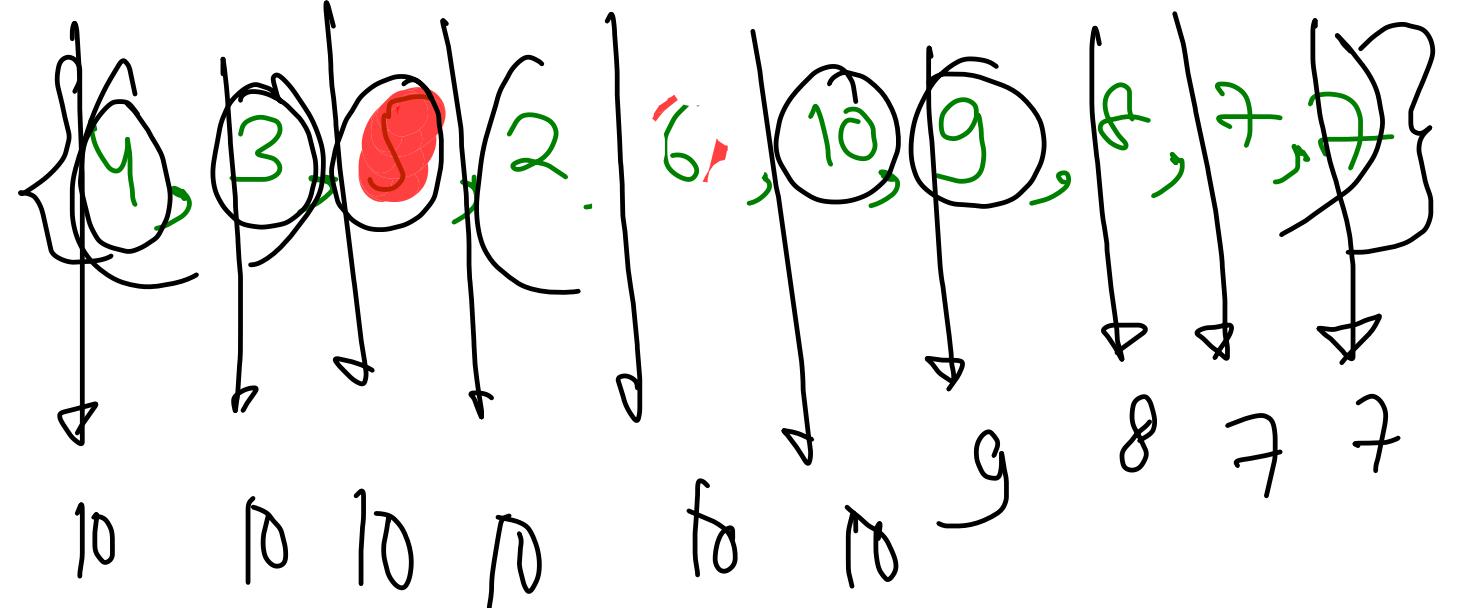
(4,2) (2,10) (4,10)

2, 8, 6



```
int Solution::minimize(const vector<int> &A, const vector<int> &B, const vector<int> &C) {
    int i = 0, j = 0, k = 0;
    int ans = INT_MAX;
    while(i < A.size() && j < B.size() && k < C.size())
    {
        int a = abs(A[i] - B[j]), b = abs(B[j] - C[k]), c = abs(C[k] - A[i]);
        ans = min(ans, max(a, max(b, c)));
        if(A[i] <= B[j] && A[i] <= C[k])
            i++;
        else if(B[j] <= C[k] && B[j] <= A[i])
            j++;
        else
            k++;
    }
    return ans;
}
```

$$O(n_1 + n_2 + n_3)$$



(i, j, k)

($i < j < k$)

$A[i] < A[j] < A[k]$

$$A[i] + A[j] + A[k] = \underline{\underline{man^m}}$$

HINT

left → Floor in left part
 Fix the middle idle
 right → man^{ma} right part

```

int Solution::solve(vector<int> &arr) {
    set<int> left;
    vector<int> right(arr.size(), 0);
    for(int i=arr.size()-1; i>=0; i--)
        right[i] = max(((i == arr.size()-1) ? 0 : right[i+1]), arr[i]);
    left.insert(INT_MIN);
    int ans = 0;
    for(int i=1; i<arr.size()-1; i++)
    {
        left.insert(arr[i-1]);
        if(arr[i] < right[i+1])
        {
            auto idx = left.lower_bound(arr[i]);
            idx--;
            ans = max(ans, right[i+1] + (*idx) + arr[i]);
        }
    }
    return ans;
}

```

 $\Theta(N)$ $\Theta(N \log N)$ $\Theta(N \log N)$ $\Theta(N \log N)$

Lecture - ③

4 sum

→ 4 sum (I & II)

→ Count Tuples

→ Sum

→ Product

Difference Pairs

→ Target Diff Pair

→ All

→ Unique

→ longest Diff
Pair (I & II)

4 Sum - II {Sorting not needed}

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

H < Int, A < A < L >>

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

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

} Pairs (AB)

-1 $\rightarrow (-2, 1)$ N^2 pairs

0 $\rightarrow (1, -1) (2, -2)$

1 $\rightarrow (2, -1)$

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

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

} Pairs (CD)

-1 $\rightarrow (-1, 0)$

0 $\rightarrow (-1, 1)$ N^2 pairs

1 $\rightarrow (-1, 2)$

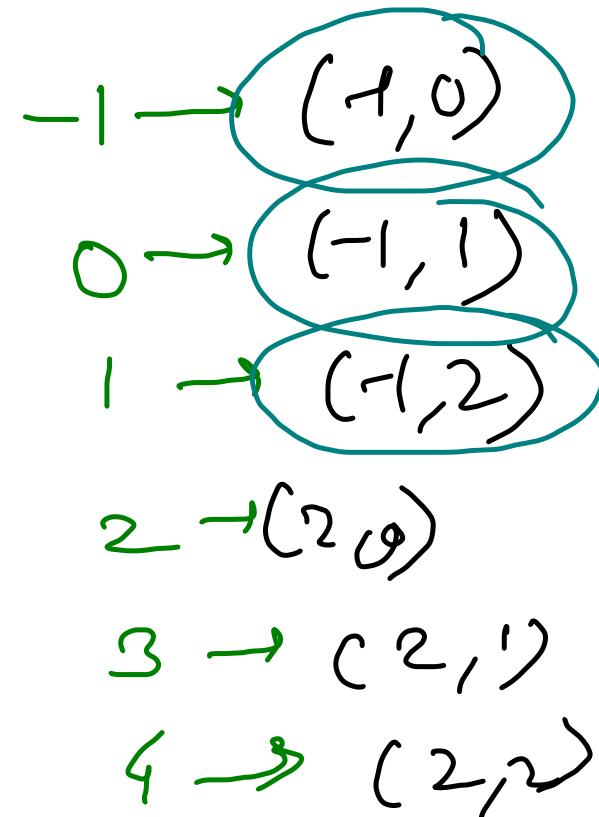
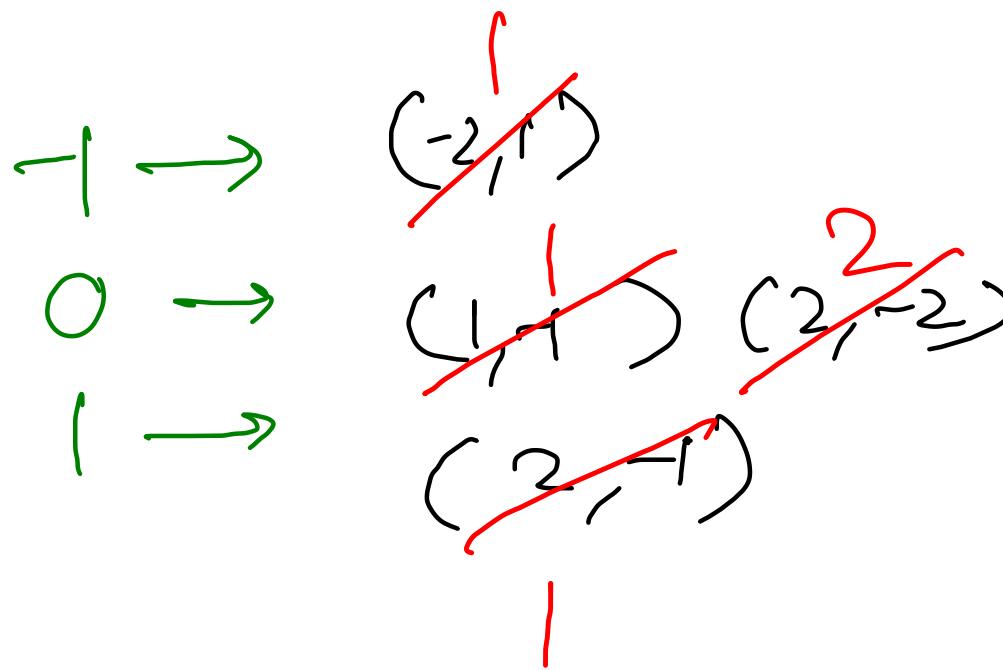
2 $\rightarrow (2, 0)$

3 $\rightarrow (2, 1)$

4 $\rightarrow (2, 2)$

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

$O(N^2) + O(N^2)$ +
 $O(\text{Resultant size})$



Count = 6

```

public int fourSumCount(int[] nums1, int[] nums2, int[] nums3, int[] nums4) {
    HashMap<Integer, Integer> AB = new HashMap<>();
    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)
            count += AB.getOrDefault(-val3 - val4, 0);
    return count;
}

```

$O(n^2)$

$O(n^2)$

$O(n^2)$

target = 160

$$\{0, \cancel{20}, \cancel{60}, \cancel{70}\}$$

$$\begin{cases} 30 \\ 40 \\ 70 \\ 10, 20 \\ 10, 60 \\ 10, 60 \end{cases}$$

$$\begin{cases} 40 \\ 30 \\ 20, 20 \\ 20, 70 \\ 20, 60 \end{cases}$$

$$\begin{cases} 150 \\ 100, 60 \\ 50 \\ 60, 70 \\ 60, 20 \\ 4, 5 \end{cases}$$

$$\begin{cases} 80 \\ 10, 20 \\ 20, 60 \\ 20, 60 \\ 20, 60 \end{cases}$$

$$\begin{cases} 10, 20 \\ 60, 70 \end{cases}$$

$$\begin{cases} 10, 60, 20, 70 \end{cases}$$

$$\begin{cases} 0, 70, 20, 60 \end{cases}$$

$$\begin{cases} 20, 20, 60, 60 \end{cases}$$

$$\begin{cases} 20, 70, 10, 60 \end{cases}$$

$$\begin{cases} 60, 60, 20, 20 \end{cases}$$

~~Count Tuples Product~~

{ 2, 3, 4, 6 }

6

(2, 3)

8

(2, 4)

12

(2, 6) (3, 4)

18

(3, 6)

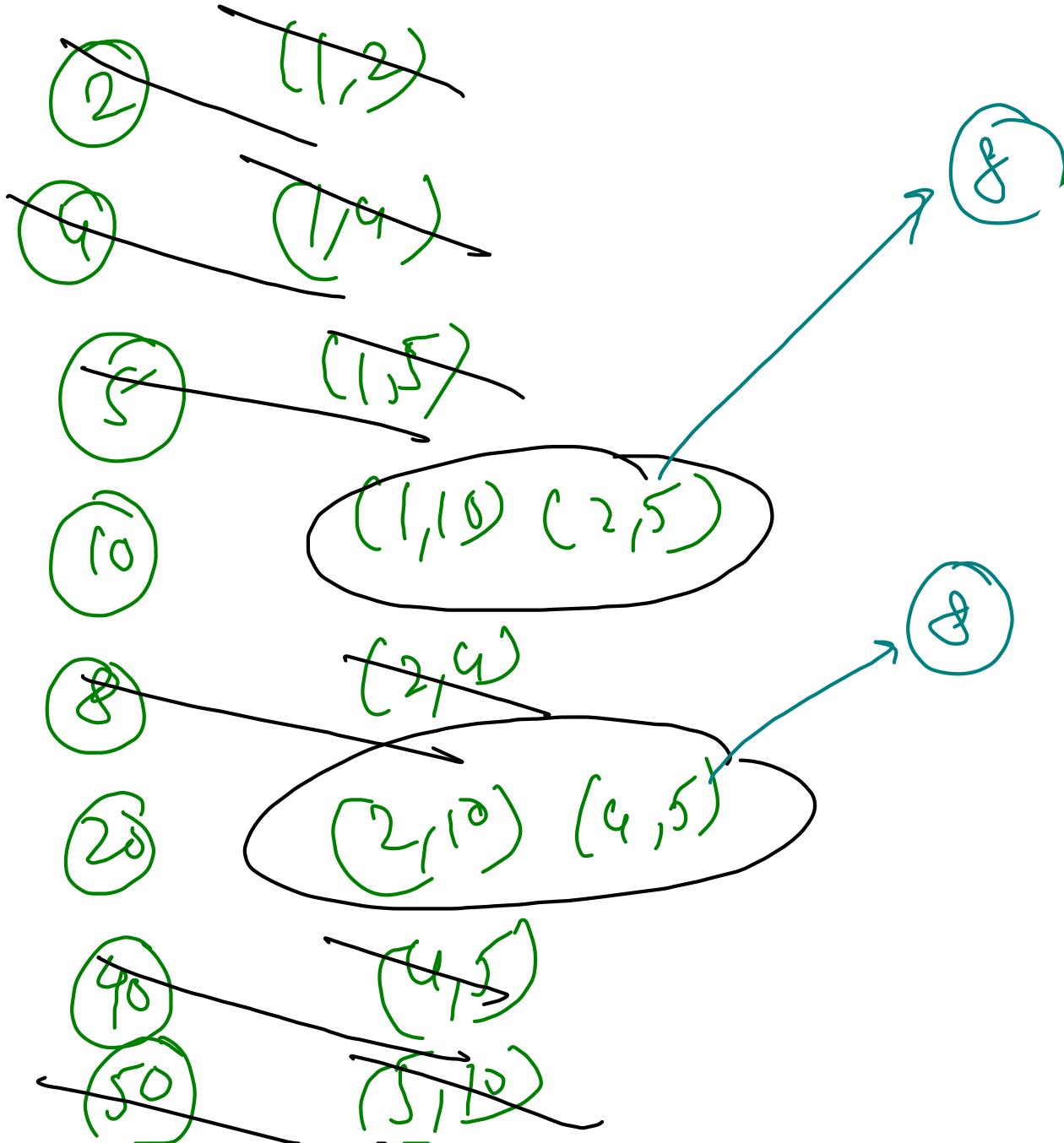
24

(4, 6)

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

```
public int tupleSameProduct(int[] nums) {  
    HashMap<Integer, Integer> AB = new HashMap<>();  
  
    int n = nums.length;  
    for(int i=0; i<n; i++){  
        for(int j=i+1; j<n; j++){  
            int product = nums[i] * nums[j];  
            AB.put(product, AB.getOrDefault(product, 0) + 1);  
        }  
    }  
  
    int count = 0;  
    for(Integer key: AB.keySet()){  
        int val = AB.get(key);  
        count = count + (val * (val - 1)) / 2;  
    }  
  
    return 8 * count;  
}
```

$\{1, 3, 4, 5, 10\}$

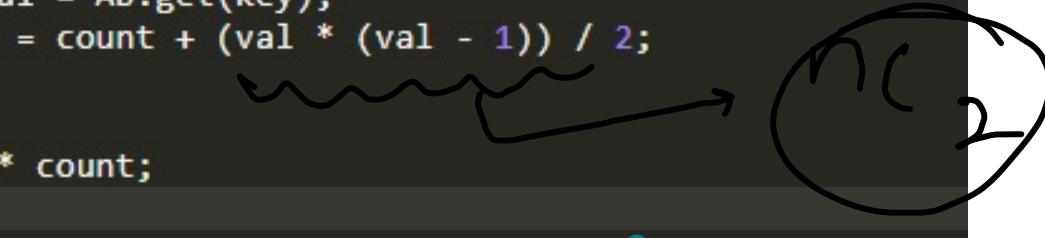


```
public int tupleSameProduct(int[] nums) {
    HashMap<Integer, Integer> AB = new HashMap<>();

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

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

    return 8 * count;
}
```



$(1, 10)$ $(2, 5)$
 $(5, 2)$

$(10, 1)$ $(2, 1)$
 $(5, 2)$

$(2, 1)$ $(1, 10)$
 $(10, 1)$

$(r, 2)$ $(1, 10)$
 $(10, 1)$

Difference Pair target = 7

Direct \Rightarrow same

{ 1, 2, 3, 10, 11 }

$$\begin{aligned} n[p2] - n[p1] \\ = 2 - 1 = 1 \uparrow \\ p2 + f \end{aligned}$$

\downarrow $\text{arr}[p2] - \text{arr}[p1] \uparrow = \text{target}$

$$\begin{aligned} 10 - 0 &= 10 \\ 10 - 1 &= 9 \\ 11 - 2 &= 9 \end{aligned}$$

$$\begin{aligned} \Rightarrow 3 - 1 &= 2 \uparrow \\ p2 + f \\ \Rightarrow 10 - 1 &= 9 \downarrow \\ = 1 \quad 10 - 2 &= 8 \uparrow \\ p1 + f \\ \Rightarrow 10 - 3 &= 7 \end{aligned}$$

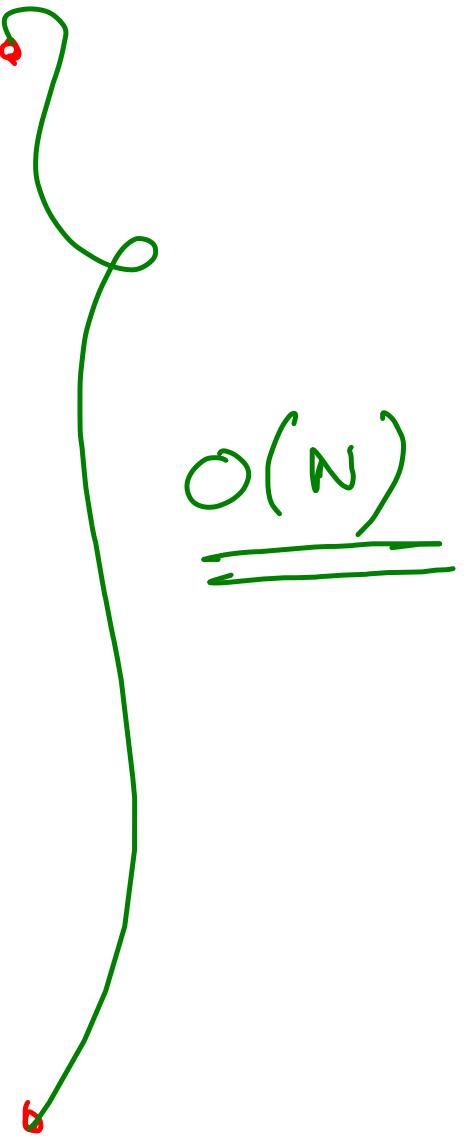
```

int left = 0, right = 1;
while(left != nums.length && right != nums.length){
    if(left == right){
        if(target < 0){
            left++;
        } else {
            right++;
        }
        continue;
    }

    int diff = nums[right] - nums[left];

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

```



$O(N)$

$\{1, 2, 3, 10, 15\}$
 L ↑ R
 target = 5

M, T, W

$\{1, 2, 3, 10, 11\}$

target = -20

2-1 = 1 ↓

2-2 = 0 ↓

2-3 = -1 ↓

2-10 = -8 ↓

2-11 = -9 ↓

p1 == nums.length

target = 0
else {
 p1 == p2
 ptr2++

$p1 \neq p2$

Pair Found

~~Q532~~ ~~Count Unique Pairs~~

{ 1, 1, 3, 3, 4, 4, 5, 5 } target = 2
l r

```
public int diffPair(int[] nums, int target){  
    int left = 0, right = 1, count = 0;  
    while(left < nums.length && right < nums.length){  
        if(left > 0 && nums[left - 1] == nums[left]){  
            left++; continue;  
        }  
        if(left == right){  
            right++; continue;  
        }  
  
        int diff = nums[right] - nums[left];  
  
        if(diff == target){  
            count++; left++;  
        } else if(diff < target){  
            right++;  
        } else {  
            left++;  
        }  
    }  
  
    return count;  
}
```

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

{ 1, 3 }
{ 5, 3 }

2006 {All}

K Difference Pair

{
1, 2, 2, 1, 3, 4, 3, 4, 3
0 1 2 3

1 → 2
2 → 2
2 → 2
3 → 3
3 → 3
4 → 2

target = 1

~~2 * 2~~ = 4

2 + 3 = 6

~~3 * 2~~ = 6
18

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

corner case

target = 0
 $x - y = 0$
 $x = y$

$$n \{ = \frac{n(n-1)}{2}$$

```
public int countKDifference(int[] nums, int k) {
    HashMap<Integer, Integer> freq = new HashMap<>();
    for(int i=0; i<nums.length; i++){
        freq.put(nums[i], freq.getOrDefault(nums[i], 0) + 1);
    }

    int count = 0;
    for(Integer key: freq.keySet()){
        int freq1 = freq.get(key);

        if(k == 0){
            count = count + (k * (k - 1)) / 2;
        } else {
            int freq2 = freq.getOrDefault(k + key, 0);
            count = count + freq1 * freq2;
        }
    }

    return count;
}
```

```
----  
int maxWidthRamp(vector<int>& arr)  
{  
    int n = arr.size();  
    int rmax[n];  
    rmax[n-1] = arr[n-1];  
    for(int i=n-2; i>=0; i--)  
        rmax[i] = max(arr[i], rmax[i+1]);  
  
    int i = 0, j = 0;  
    int ans = 0;  
    while(i < n && j < n)  
    {  
        if(arr[i] <= rmax[j])  
        {  
            ans = max(ans, j - i);  
            j++;  
        }  
        else i++;  
    }  
    return ans;  
}
```

Max width ramp

January & February Schedule

Thursday → 8 PM to 12 AM → DSA { level 2 }

Saturday → 9 AM to 5 PM → DSA { level 2 }

→ 9 PM to 12 AM → System Design (HLD + LLD)

Sunday → 9 AM to 5 PM → DSA { level 2 }

→ 9 PM to 12 AM → System Design (HLD + LLD)

(STATIC WINDOW)

Lecture ④

{ Thursday }

→ Max sum Subarray of size K.

→ Distinct No in Windows

→ Min swaps K together

→ First -ve in Windows

→ Anagram Substrings

→ Sliding Window Maxm

(DYNAMIC WINDOW)

Lecture ⑤

{ Saturday Morning }

→ Max Consecutive ones (I, II & III)

→ Minimum Window Substring (I & II)

→ Substring with

- W/o Repeating
- Almost K unique
- Exact K unique

→ Subarrays sum in Range

(Two POINTER)
Classic

Lecture ⑥

{ Sunday Morning }

* → Trapping Rain Water

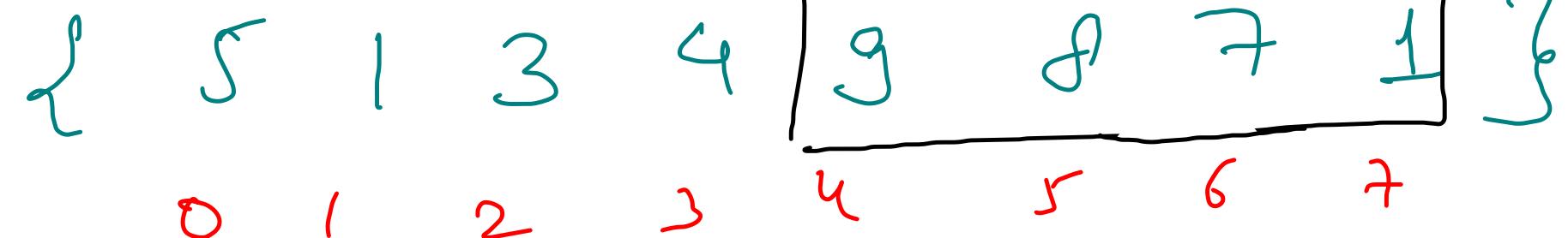
→ Max Area Container

→ Boats to Save People

→ Min strings Deleting Ends

→ Distinct Absolute Array Elements

Static Sliding Window



$$\text{sum} = \cancel{5+1} + \cancel{3+4} + 9 + 8 + 7 + 1$$

$$\text{maxSum} = 28$$

{manSum window}

$$\text{window} = 4$$

Fixed window size

Static

$$N = 8$$

$$k = 4$$

```

static int maximumSumSubarray(int win, ArrayList<Integer> arr, int N){
    int currSum = 0;
    for(int i=0; i<win; i++){
        currSum += arr.get(i);
    }

    int maxSum = currSum;
    for(int i=win; i<N; i++){
        int inc = arr.get(i);
        int exc = arr.get(i - win);
        currSum = currSum + inc - exc;

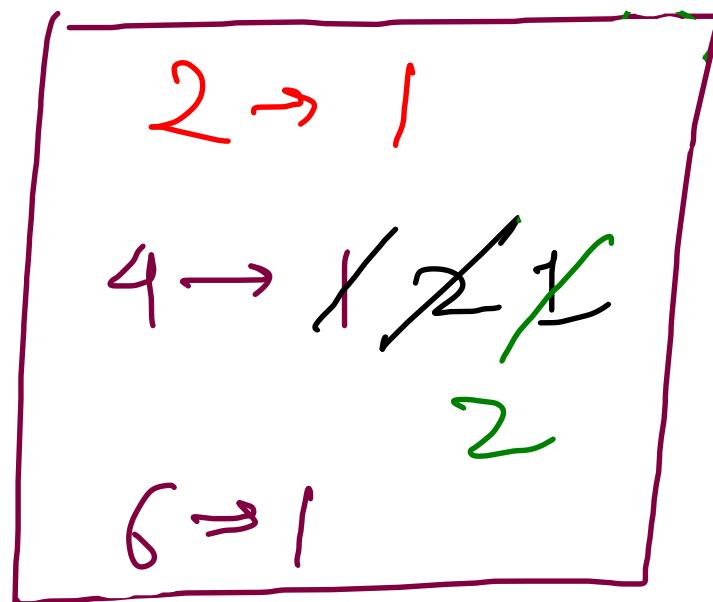
        maxSum = Math.max(maxSum, currSum);
    }
    return maxSum;
}

```

O(N) Time

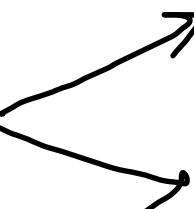
O(1) Space

window = 4



{ 3, 3, 4, 3, 3, 3, 3 }

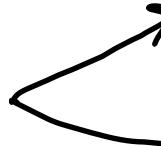
Include



present → update (freq++) O(1)

not present → insert (key, 1) O(1)

Exclude



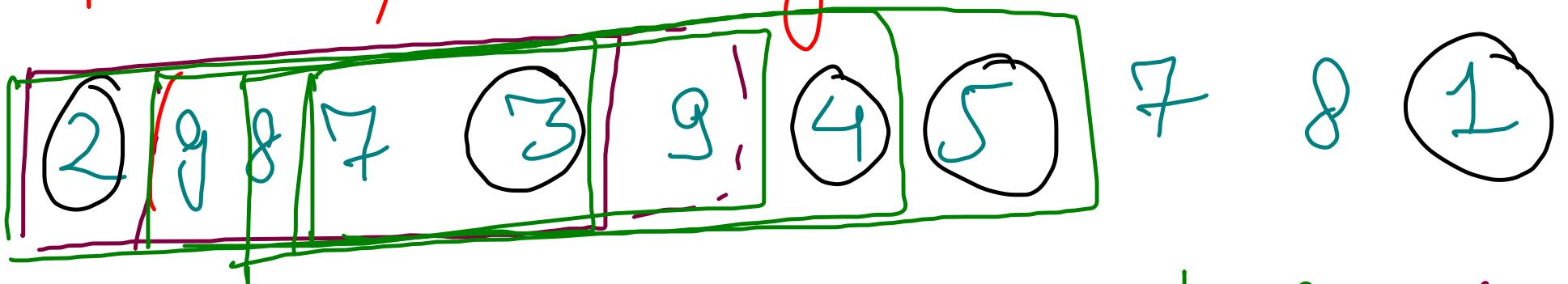
freq > 1 { freq - } O(1)

freq = 1 { remove } O(1)

```
public int[] dNums(int[] A, int B) {  
    int[] res = new int[A.length - B + 1];  
  
    HashMap<Integer, Integer> freq = new HashMap<>();  
    for(int i=0; i<B; i++){  
        freq.put(A[i], freq.getOrDefault(A[i], 0) + 1);  
    }  
  
    int window = 0;  
    res[window++] = freq.size();  
  
    for(int i=B; i<A.length; i++){  
        freq.put(A[i], freq.getOrDefault(A[i], 0) + 1);  
  
        if(freq.get(A[i - B]) == 1){  
            freq.remove(A[i - B]);  
        } else {  
            freq.put(A[i - B], freq.get(A[i - B]) - 1);  
        }  
  
        res[window++] = freq.size();  
    }  
  
    return res;  
}
```

$O(N)$

Min swaps to k together



$$k = 6$$

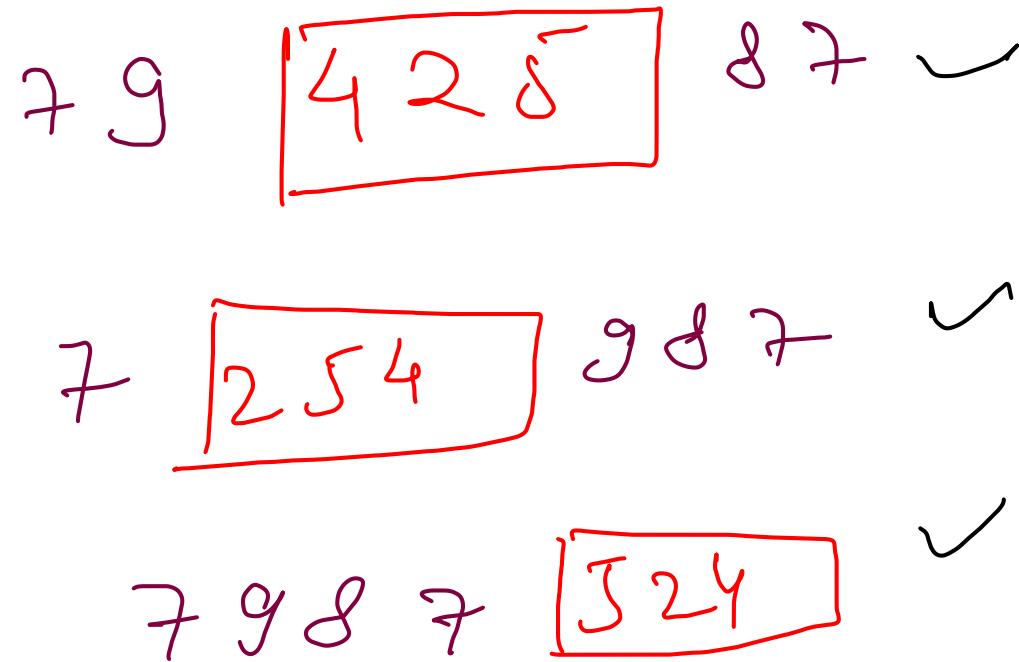


→ Calculate window's size $O(N)$



→ Sliding window } $O(N)$
total - ek $\leq k$

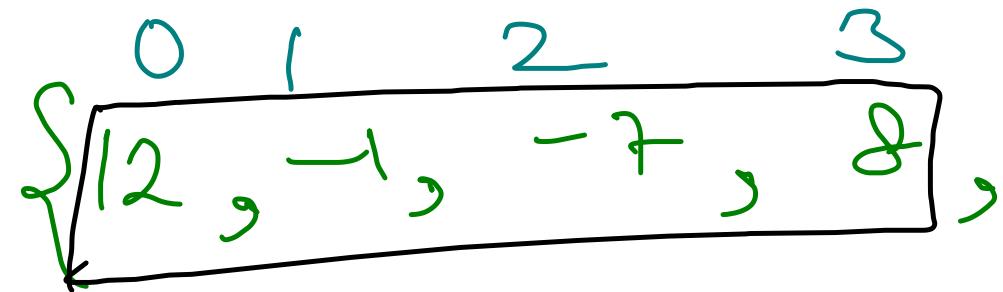
$$j-2 = 3$$



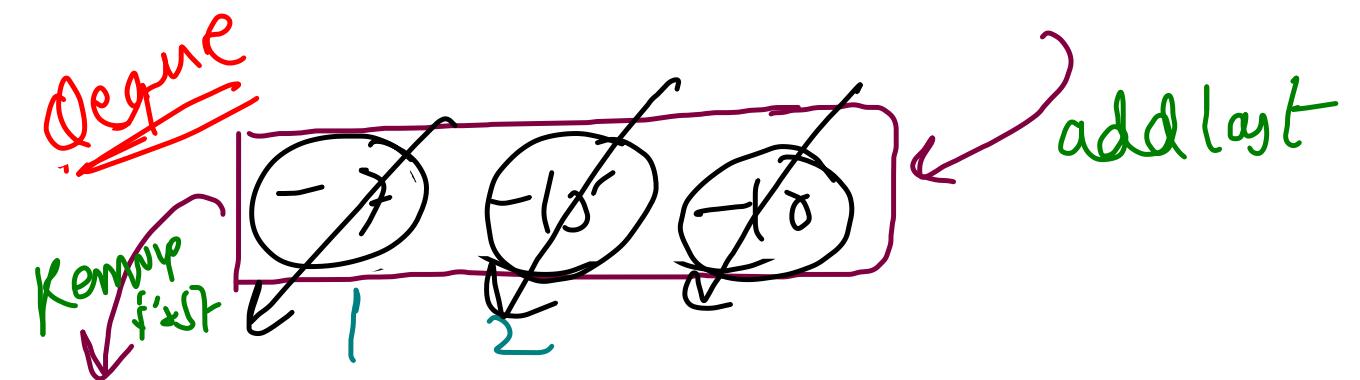
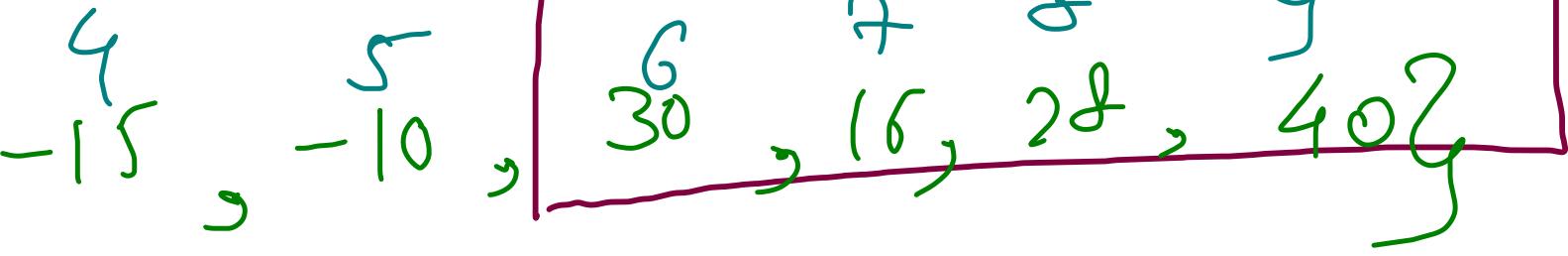
```
public static int minSwap (int arr[], int n, int k) {  
    int window = 0;  
    for(int i=0; i<n; i++){  
        if(arr[i] <= k){  
            window++;  
        }  
    }  
  
    int moreThanK = 0;  
    for(int i=0; i<window; i++){  
        if(arr[i] > k){  
            moreThanK++;  
        }  
    }  
  
    int minSwaps = moreThanK;  
    for(int i=window; i<n; i++){  
        if(arr[i] > k)  
            moreThanK++;  
        if(arr[i - window] > k)  
            moreThanK--;  
  
        minSwaps = Math.min(minSwaps, moreThanK);  
    }  
    return minSwaps;  
}
```

$\Theta(n)$

First -ve in Every window of size K



window = 4



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

Exclude \rightarrow -ve \rightarrow remove first
+ve \rightarrow do nothing

Include \rightarrow -ve \rightarrow add last
+ve \rightarrow do nothing

and when \rightarrow que empty $\rightarrow 0$
else $q \cdot \text{getFirst}()$

```

public long[] printFirstNegativeInteger(long A[], int N, int K)
{
    Deque<Integer> q = new ArrayDeque<>();
    for(int i=0; i<K; i++){
        if(A[i] < 0){
            q.addLast(i);
        }
    }

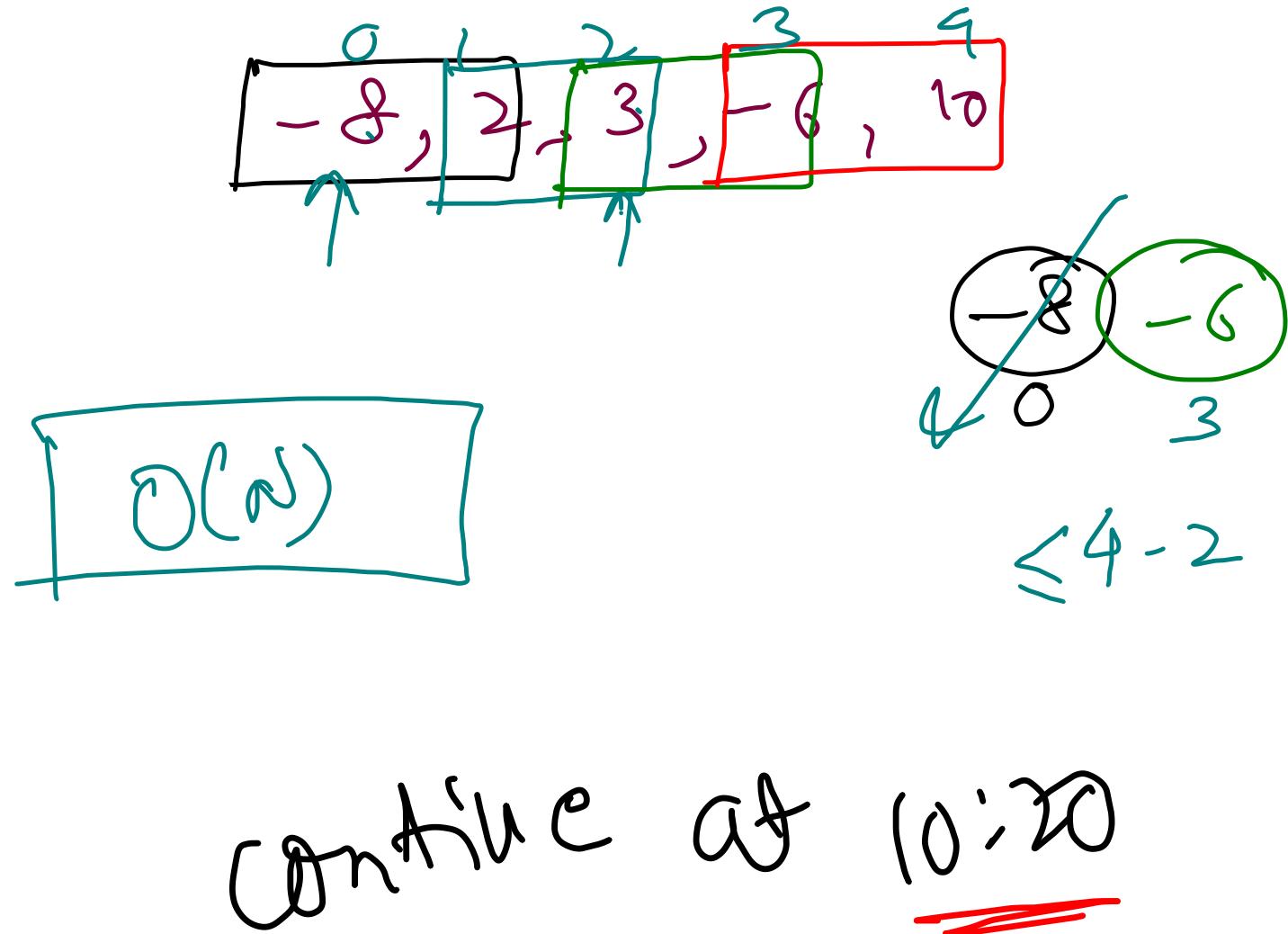
    long[] res = new long[N - K + 1];
    int window = 0;
    res[window++] = (q.size() == 0) ? 0 : A[q.getFirst()];

    for(int i=K; i<N; i++){
        if(q.size() > 0 && q.getFirst() <= i - K){
            q.removeFirst();
        }

        if(A[i] < 0){
            q.addLast(i);
        }
        res[window++] = (q.size() == 0) ? 0 : A[q.getFirst()];
    }

    return res;
}

```



{
Program
Permutation} in String

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

$a \rightarrow 1 \ b \rightarrow 1 \ c \rightarrow 1$

"abcd"

"abcc"

$a \rightarrow 1$

$b \rightarrow 1$

$c \rightarrow 2$

"bccal"

$a \rightarrow 1 \ b \rightarrow 1$

$b \rightarrow 1$

$c \rightarrow 2$

"ccao"

$a \rightarrow 2$

$c \rightarrow 2$

"caab"

$a \rightarrow 2$

$c \rightarrow 1$

$b \rightarrow 1$

"aabd"

$a \rightarrow 2$

$d \rightarrow 1$

$b \rightarrow 1$

"abdd"

$a \rightarrow 1$

$b \rightarrow 1$

$d \rightarrow 2$

"bdda"

$b \rightarrow 1 \ d \rightarrow 2 \ a \rightarrow 1$

"dacb"

$a \rightarrow 1 \ c \rightarrow 1$
 $b \rightarrow 1 \ d \rightarrow 1$

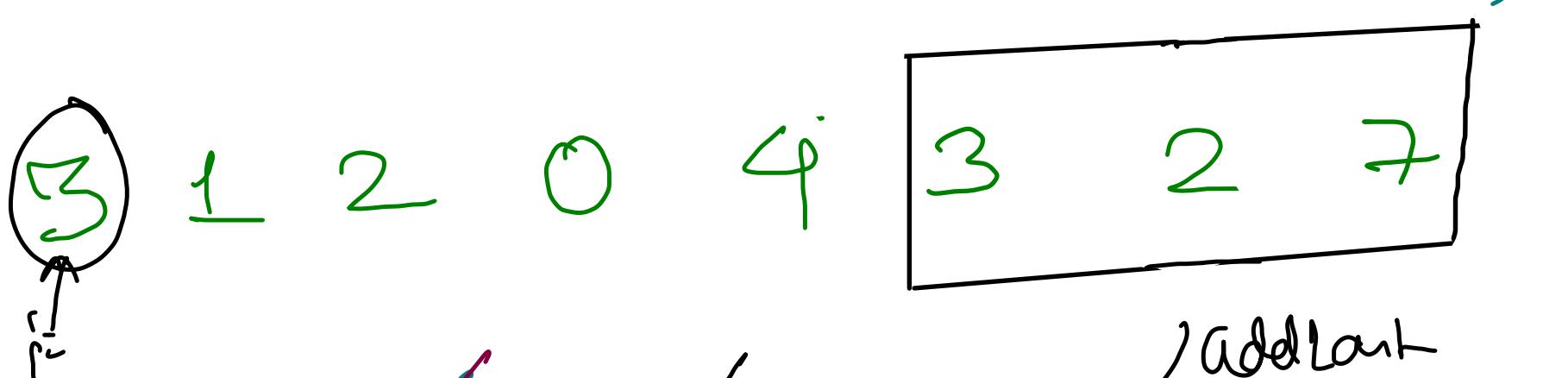
$4!$

abcd
bcda
cabd
...
!

```
public boolean isEqual(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 window = s1.length();  
  
    for(int i=0; i<window; 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 >= window){  
            currFreq[s2.charAt(i - window) - 'a']--;  
        }  
  
        if(isEqual(reqFreq, currFreq) == true){  
            return true;  
        }  
    }  
    return false;  
}
```

} $O(26 \times N)$

Sliding Window Maximum



Next Greater / Stack $\rightarrow O(n)$
Deque (Sliding window)

$$\text{window} = \{3\}$$

\rightarrow Deque

removeFirst \rightarrow out of window
removeLast \rightarrow val < addval
addLast \rightarrow always

deque $\begin{cases} \xrightarrow{\text{chronological order}} \\ \xrightarrow{\text{decreasing order}} \end{cases}$
 $\max^n \rightarrow$ que front

```
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;
}
```

} $O(N)$ 1 push for each ele
 | pop for each ele

(DYNAMIC window)

Lecture 5

{ Sunday Morning }

→ Max consecutive
ones (I, II & III)

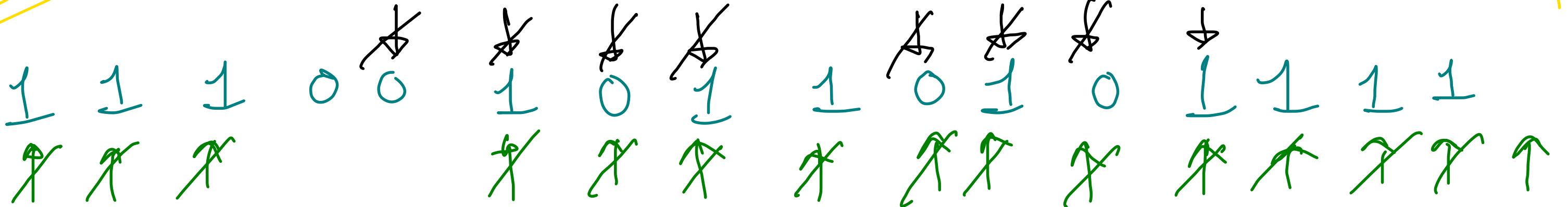
→ Minimum Window
substring (I & II)

→ Substring with
w/o Repeating

LC
485

Max-Consecutive Ones - ①

$R = 0$



```
public int findMaxConsecutiveOnes(int[] nums) {  
  
    int max = 0;  
    for(int i=0; i<nums.length; i++){  
        if(nums[i] == 0){  
            continue;  
        }  
  
        int count = 0;  
        while(i < nums.length && nums[i] == 1){  
            count++;  
            i++;  
        }  
  
        max = Math.max(max, count);  
    }  
  
    return max;  
}
```

$O(n)$

~~LC Q1004~~

Max Consecutive Ones - III

{ longest subarray with atmost k 0's }

[0,0,1,1,0,0,1,1,1,0,1,1,0,0,0,1,1,1,1].

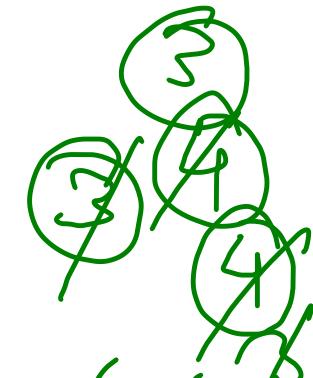
```
public int longestOnes(int[] nums, int k) {
    int maxLen = 0, countOfZeros = 0, left = 0;
    for(int right = 0; right < nums.length; right++){
        if(nums[right] == 0) countOfZeros++;

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

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

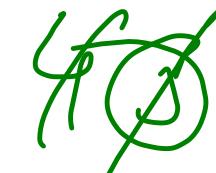
    return maxLen;
}
```

$k = 3$



curr zeros = 0 1 2

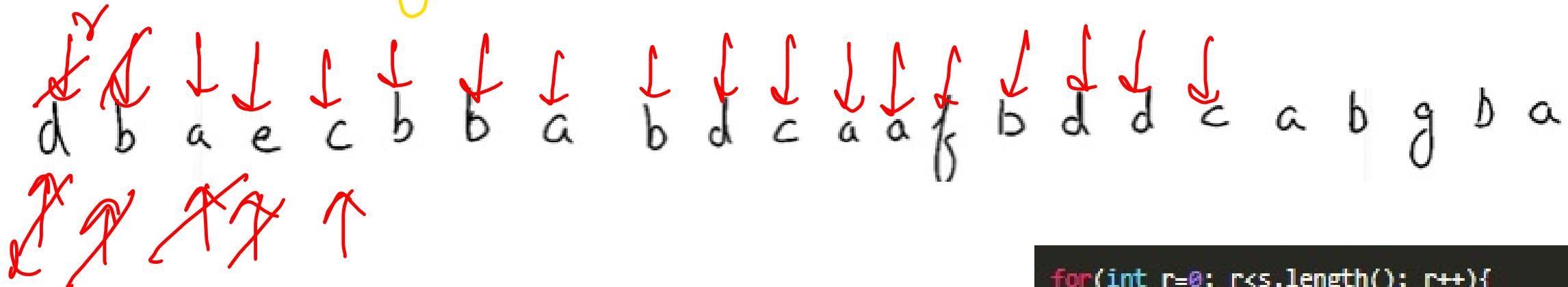
maxlen = 0 1 2 3



(Q76 heetode)

Minimum Window Substring

{ IMPORTANT }



$$d=1 \quad b=3 \quad a=2 \quad c=1/2$$

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

$MC = 0$
 $\cancel{3} \cancel{4} \cancel{5} \cancel{6}$

$right++ \Rightarrow$ making invalid substring val \neq
 $left++ \Rightarrow$ making valid substring of min length

```
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 - 1 < len){
            idx = l; len = r - 1 + 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++;
    }
}
```

pattern

a b b c d c

a → 1

b → 2

c → 2

d → 1

$min = \infty$

$\cancel{\infty}$
 $\cancel{1}$
 $\cancel{2}$

$\cancel{3}$
 $\cancel{4}$
 $\cancel{5}$
 $\cancel{6}$
 $\cancel{7}$

(Q3) Leetcode)

"longest substring without repeating characters"

$k=1$

longer 1
Valid \rightarrow include
invalid \rightarrow exclude

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

HashMap

a \rightarrow X \emptyset 1
b \rightarrow X X 1

c \rightarrow 1

max

count = $\emptyset \times \beta \times \text{length} - \text{right-left} + 1$

a, ab, b¹, b², ba, a, bac, ac, c

d \rightarrow
c \rightarrow

~~Man Vergh~~

```
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;
}
```

$O(N)$

~~Count valid~~

```
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;
}
```

Lecture 6

3PM to 6PM

{ Sunday Evening }

① → Substring with

- Almost K Unique
- Exact K Unique

② → R odds → K Repeating
→ Repeating Replacement

③ {	<u>Smallest Subarray with Sum $\geq X$</u>
④ {	<u>Maximum Sum Subarray $< X$</u>
⑤ {	<u>Count Subarrays with Sum $< X$</u>
	<u>Count Subarrays with Product $< X$</u>
	<u>Count Subarrays with Maximum $> X$</u>
	<u>Count Subarrays with Max in Range</u>

longest

```
HashMap<Character, Integer> freq = new HashMap<>();
int maxLen = 0, left = 0;

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

    while(freq.size() > k){
        char chl = str.charAt(left);
        freq.put(chl, freq.getOrDefault(chl, 0) - 1);

        if(freq.get(chl) == 0){
            freq.remove(chl);
        }
        left++;
    }

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

return maxLen;
```

Count

```
HashMap<Character, Integer> freq = new HashMap<>();
int count = 0, left = 0;

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

    while(freq.size() > k){
        char chl = str.charAt(left);
        freq.put(chl, freq.getOrDefault(chl, 0) - 1);

        if(freq.get(chl) == 0){
            freq.remove(chl);
        }
        left++;
    }

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

return count.
```

Enact k → longest

```
HashMap<Character, Integer> freq = new HashMap<>();
int maxLen = 0, left = 0;

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

    while(freq.size() > k){
        char chl = str.charAt(left);
        freq.put(chl, freq.getOrDefault(chl, 0) - 1);

        if(freq.get(chl) == 0){
            freq.remove(chl);
        }
        left++;
    }

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

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

Enact k → Count

Same logic will
not
work
for Count



longest \rightarrow substring with at most k unique characters
Count

γ
d d a c b b a c c d e d a c e b b
 β

$$k = 3$$

$$\max \text{Lon} = 77^\circ 34' 45''$$

Invalid → left ++
(endbrace)

$$\text{cont} = \text{X}\beta\delta\alpha\beta\alpha^2$$

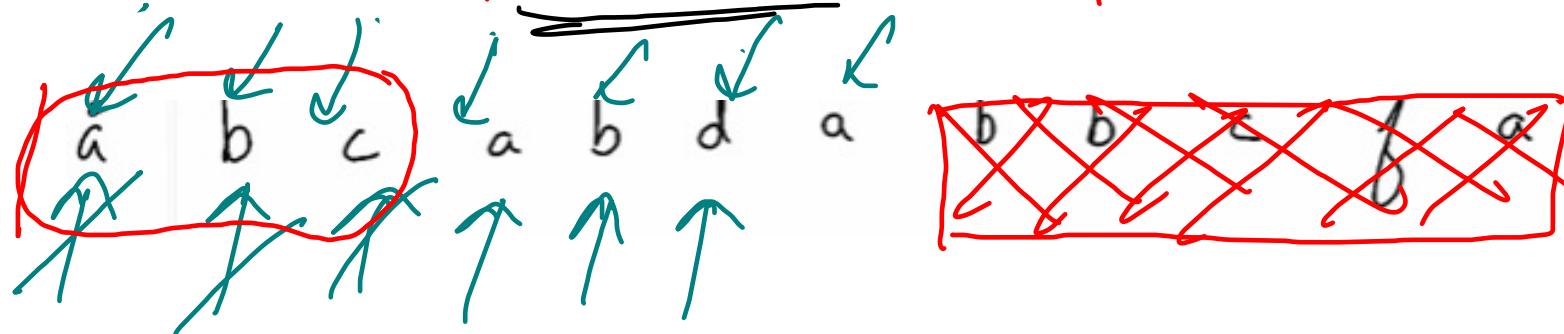
Valid → right +
(include)

d^1, d^2, dd, dda, da, a
 $ddac, dac, ac, c, acb, cb, bs, acbb,$
 $ccb, bb, 1$

- d → X^oX
- a → X2
- c → 1
- b → X2

→ Almost K unique - Almost (k-1) Unique

Count → Extract k unique characters



$$k = \textcircled{3} - \textcircled{2}$$

~~abca~~

10

a_3^L, c, a
 a_b, b, c

a^3 , ab , bc , c

$a^b c, b^c a$

also

a, b, c

a^5, b^6, c^a

$$\text{Exactly 3} = \text{Almost 3} - \text{Almost 2}$$

24-12

12

A hand-drawn dollar sign inside a circle, with a large checkmark drawn over it.

6

~~10~~ ~~15~~ ~~19~~

24

A diagram consisting of two circles. The top circle has a clockwise arrow inside it. The bottom circle has a clockwise arrow inside it. There are also two curved arrows pointing upwards from the circles towards the top left corner of the page.

12

[Handwritten signature]

100%

[Signature]





d-e

a-1

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

1248

Count No of Nice Subarrays

=> Count of subarrays with Exactly k odd no's

$$\text{Exactly } k = \frac{\text{Almost } k}{\text{odd nos}} - \frac{\text{Almost } (k-1)}{\text{odd nos}}$$

Even No's: Do not insert

```
long long helper(vector<int> &nums, int k)
{
    long long left = 0, right = 0;
    long long res = 0, odd = 0;
    for(right = 0; right < nums.size(); right++)
    {
        if(nums[right] % 2 == 1) odd++;
        if(odd <= k)
        {
            res += right - left + 1;
            continue;
        }
        while(left <= right && odd > k)
        {
            if(nums[left] % 2 == 1)
            {
                left++;
                odd--;
                res += right - left + 1;
                break;
            }
            left++;
        }
    }
    return res;
}
int numberOfSubarrays(vector<int>& nums, int k) {
    return helper(nums, k) - helper(nums, k-1);
}
```

Longest Repeating character Replacement

$$B = 3$$

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

26 times

two pointer

$$O(26N) = O(N)$$

(A) \rightarrow longest substring with atmost $k (= A)$ chars

(B) \rightarrow longest substn with atmost $k (= B)$ char

'as', 'bs', 'cs', 'ds' - - - - -

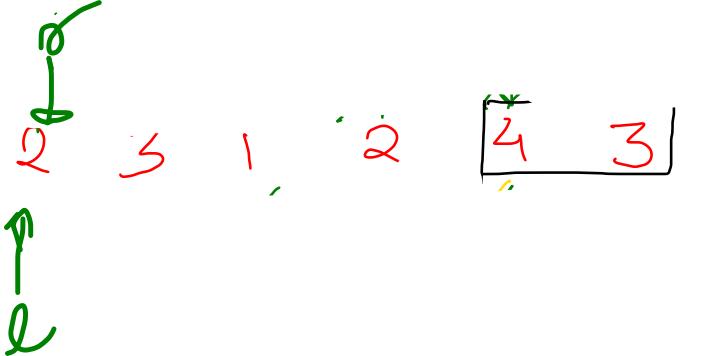
Return the length of the longest substring containing the same letter you can get after performing the above operations.

~~Non consecutive ones~~

```
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++;  
  
        // make subarray valid by excluding left elements  
        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;  
}
```

~~O(26N)~~

209
Find the smallest subarray with $\text{sum} \geq \text{target}$
target = 7



$\minLen = \cancel{0} \cancel{4} \cancel{2}$

$\text{right}++ \rightarrow \text{invalid subarray}$
($\text{sum} < \text{target}$)

$\text{left}++ \rightarrow \text{smallest valid subarray}$

$$\begin{aligned} \text{sum} = & 0 + 2 + 3 + 1 \\ & + 2 \\ & - 2 + 4 \\ & - 3 - 1 \\ & + 3 - 2 \\ & - 4 \end{aligned}$$

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

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

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

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

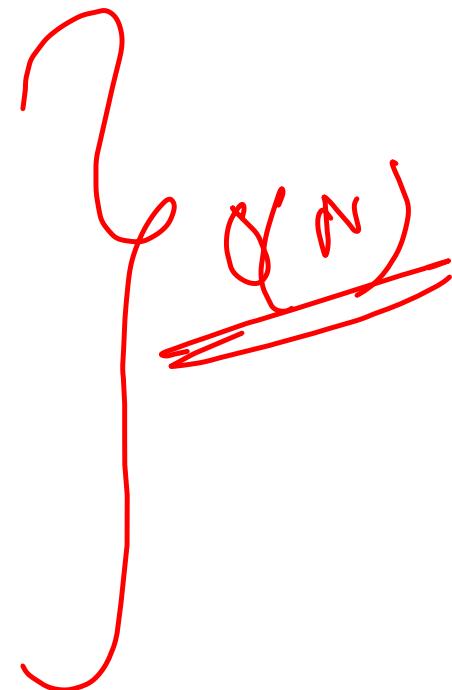
$O(N)$

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

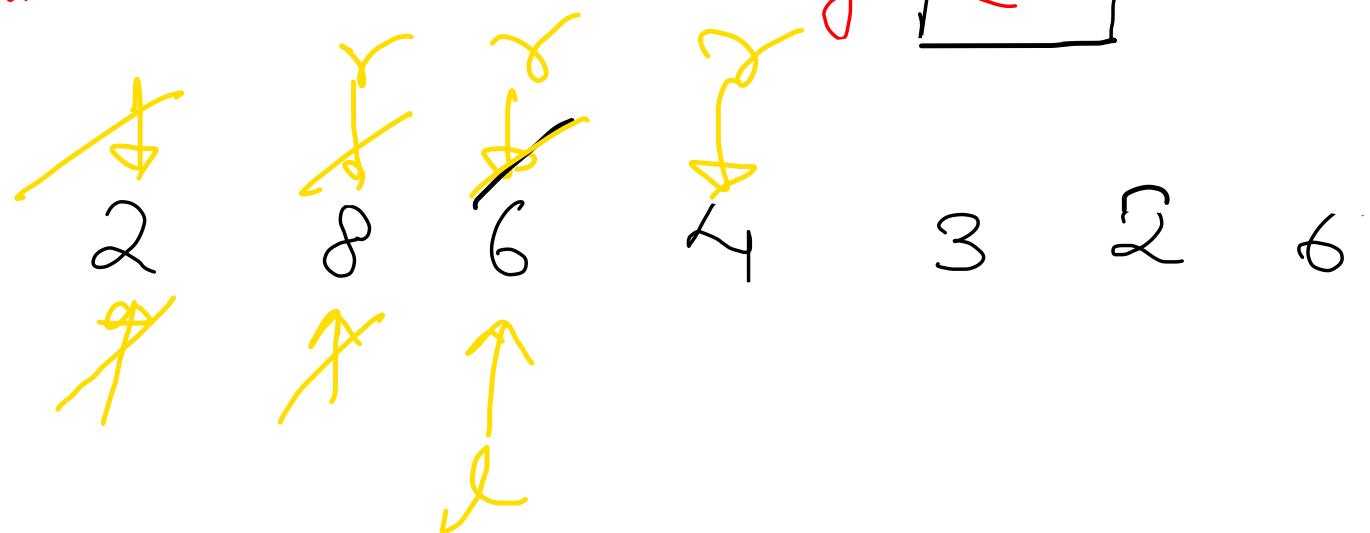
    for(int right = 0; right < nums.length; right++){
        sum += nums[right];

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

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



Maximum sum subarray  Kadane's



$$\text{sum} = \cancel{2} + \cancel{8} + \cancel{6}$$

$$\text{maxSum} = \cancel{2} \cancel{8} \cancel{6}$$

valid (Included)
right++
 $\text{sum} \leq \text{target}$

invalid (Excluded)
left++
 $\text{sum} > \text{target}$

target = 7

sat
λ
δn

DSA
 $S - 30 + 2:30$
 $= 8 hrs$

```
int left = 0;
long maxSum = 0, sum = 0;

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

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

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

return maxSum;
```

