TWO POINTER TECHNIQUE

lecture (1)

Two Sum / Target Sum Pairs

→ Unsorted
→ Sorted
→ Count
→ Count Unique
→ Two Matrices

→ Design
→ Closest
→ Absolute
→ Smaller
→ Greater

Lecture (2)

3 Sum

Target Sum Triplets

- → Unique
- → Smaller
- → Closest

Count Valid Triplets

Valid Triangles

Max Sum Triplet

Minimize Differences

4 Sum

→ 4 Sum (I & II)

→ Count Tuples

- → Sum
- → Product

Difference Pairs

→ Target Diff Pair

- → All
- → Unique

→ Longest Diff Pair (I & II)

K Sum

Q) Two Sum - Unsorted

target => 10

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

Hashmap

7→0
2→1
4→2,4
9→3
5→5
2→6

→ O(N) extra space

O(N) Time

Single Traversal

→ $O(N^2)$
Brute force

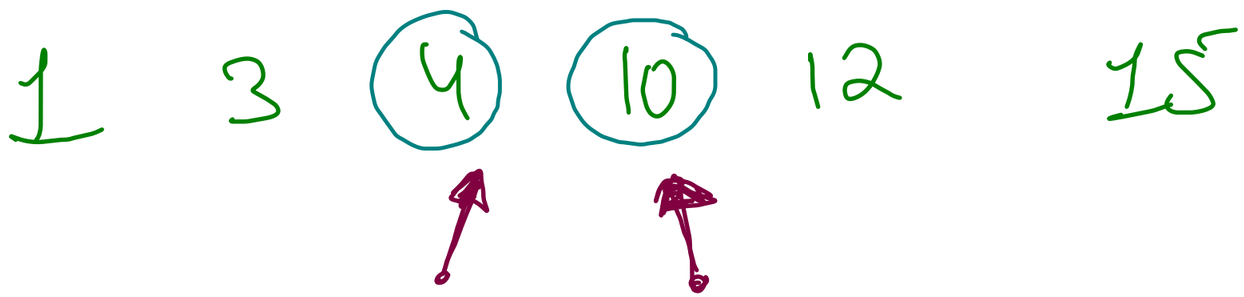
→ O(N)T, O(N)S
Hashmap

→ O(NlogN)T
O(1)S
Two Pointer

```
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;
}
```

O(1/2) Two Sum → Sorted

target = 14



O(n) time

O(1) extra space

|+|5| > 14

↑|+12 < 14

3+12↓ > 14

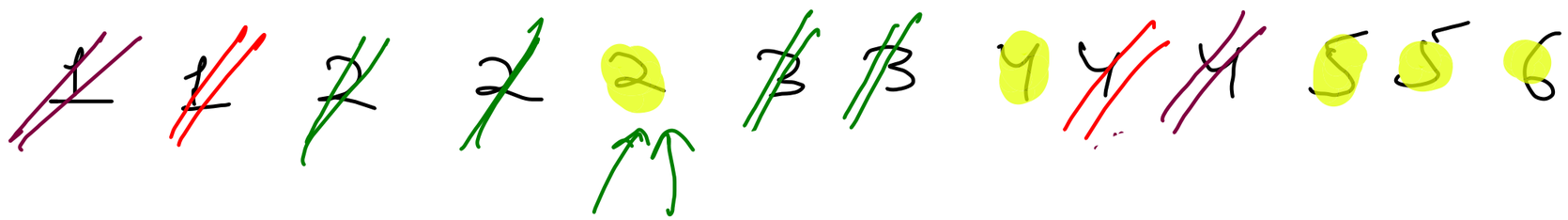
↑3+10 < 14

```
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;
```

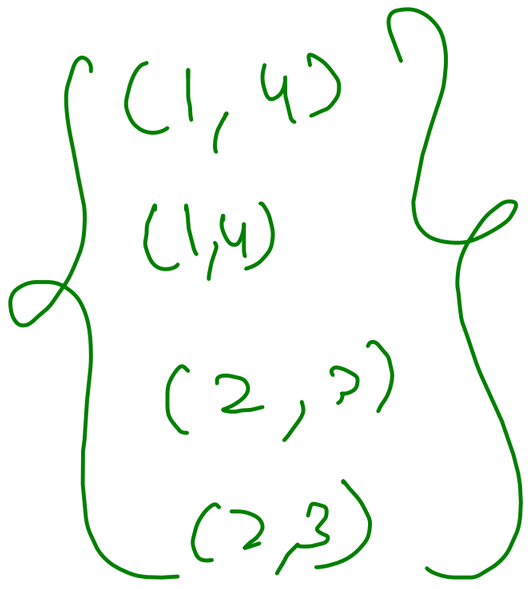
Q(1679)

Target Sum Pairs → Remove

target = 5



Count of pairs = 0 1 2 3 4



O(NlogN) Time

O(1) Extra space

```java
public int maxOperations(int[] nums, int target) {
    Arrays.sort(nums); ← NlogN
    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)

Lint code

(387) Unique Target Sum Pairs

target = 5

1 1 2 2 2 3 3 4 4 4 5

```
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

Two Sum - Closest

lintcode 533


```java
public int twoSumClosest(int[] nums, int target) {
    Arrays.sort(nums);
    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;
}
```

O(NlogN)

O(N)

target = 14

ans = ~~16~~ ~~18~~ 11

1+16 = |17-14| = 3

1+12 = |13-14| = 1

5+12 = |17-14| = 3

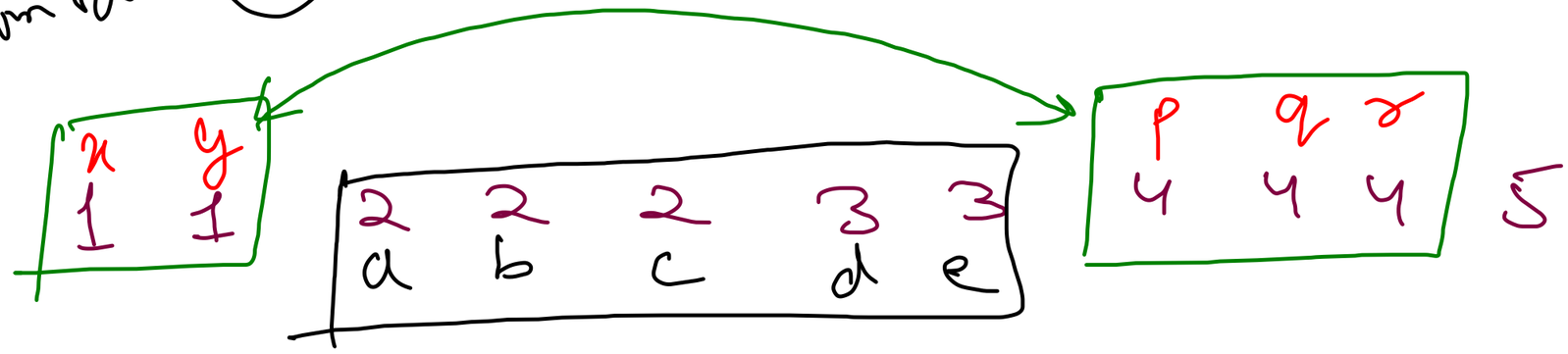
5+10 = |15-14| = 1

5+7 = |12-14| = 2

GFG

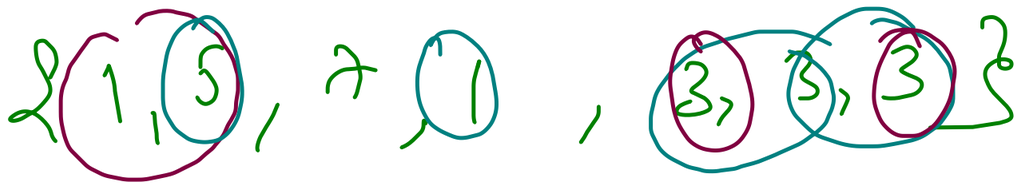
Target Sum Pair - (11)

target = 4



(x,p) (x,q) (x,r) (a,d) (b,d) (c,d) 1f * 4f = 2 * 3

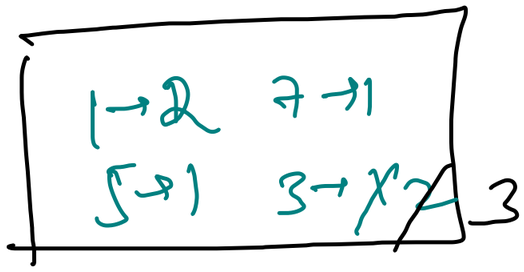
(y,p) (y,q) (x,r) (a,e) (b,e) (c,e) 2f * 3f = 3 * 2



target = 6

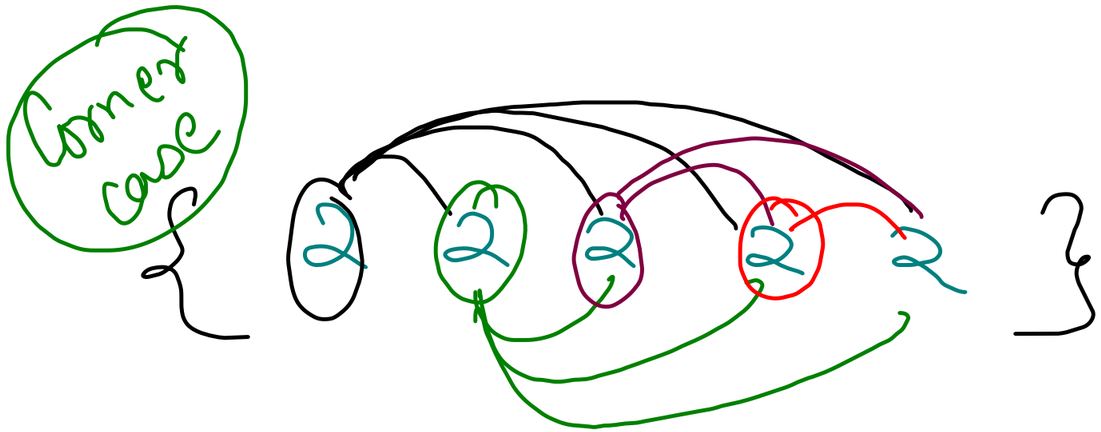
```java
HashMap<Integer, Integer> hm = new HashMap<>();

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;
```

S->O(N) extra space



count = 0 1 2 3 5

O(N) time

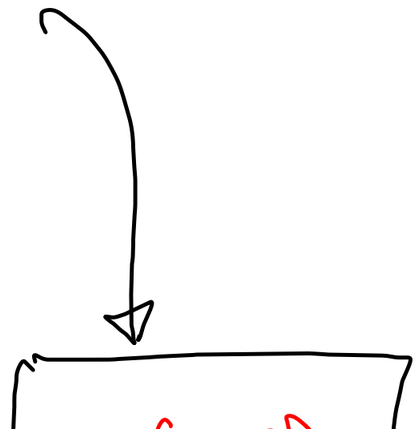


$f[2] = 5$

$(n-1) + (n-2) + (n-3) + \cdots 1 = \boxed{\frac{n*(n-1)}{2}}$

complement == ele

target = 4



Target Sum Pair (IV) [Repeating]

1	5	6
8	10	11
15	16	18

2D to 1D

1 5 6 8 10 11 15 16 18

idx → 4

4 → 1 (4/3) , 1 (4%3)

mxidx = idx/cols

colidx = idx%cols

target = 21

2	4	7
9	10	12
13	16	20

2D to 1D

2 4 7 9 10 12 13 16 20

{1,20} { 5,16} { 8, 13} {11,10}

609 LintCode

2 Sum - smaller or equal

1 3 4 10 12 15

target $\leq$ (14)

$= 14$

< 14

1+15 > 14

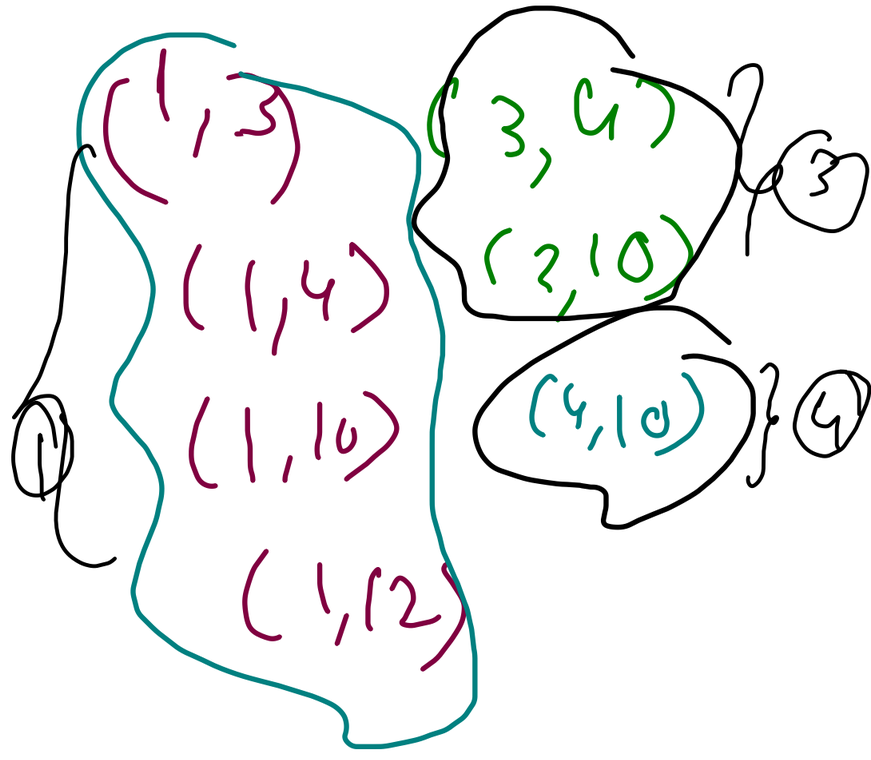
↑ 1+12 < 14 ⇒ (4)

3+12 > 14

↑ 3+10 < 14 ⇒ (2)

4+10 = 14 ⇒ (1)

```java
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 lintcode

2 Sum - Greater

target > 14

1 1 3 4 10 12 15

```
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;
```

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

(12,15)

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

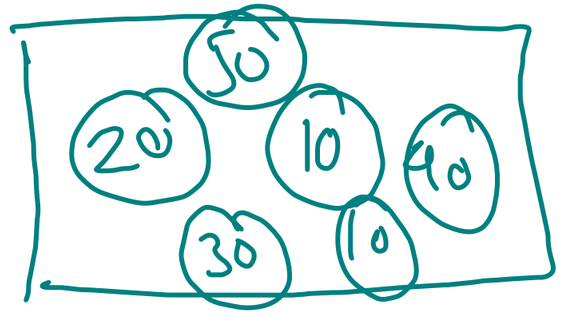
Two Sum - Design / Stream / Online

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)
2. find (40) = False
3. add (10)
4. add (30)
5. add (40)
6. find (50) = True
7. find (40) = True
8. add (10)
9. add (50)
10. find (90) = True

① ArrayList

- add
  - ① Add (cost-) ⇒ O(nlogn)
  - ② Sort
- find
  - Two pointer ⇒ O(N)

② ArrayList

- add → addLast ⇒ O(1)
- find
  - if sorted → two pointer ⇒ O(N)
  - else
    - Sort
    - two pointer

⇒ O(nlogn)

```java
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 → O(log N)

find → Two pointers → O(N)

Java code → Tree Map ⇓ next(), hasNext()

C++ code

```cpp
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;
}
```

④ Hashmap of frequency

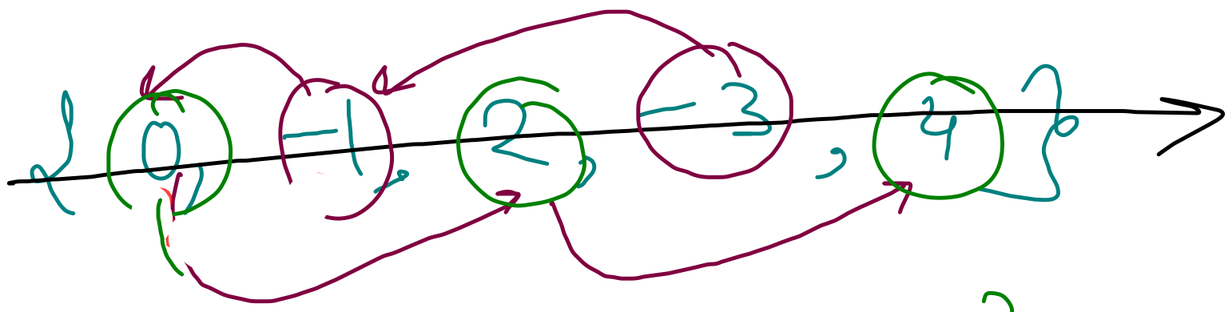
add → frequency update → { O(1) avg case, O(N) worst case (collisions)

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

Corner case → equal complement → freq >= 2

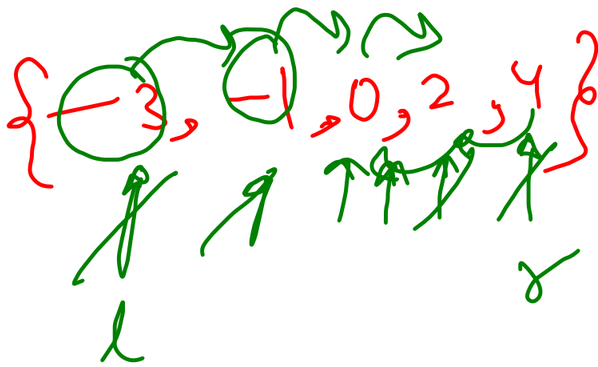
```java
public class TwoSum {
    HashMap<Integer, Integer> freq;

    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;
    }
}
```

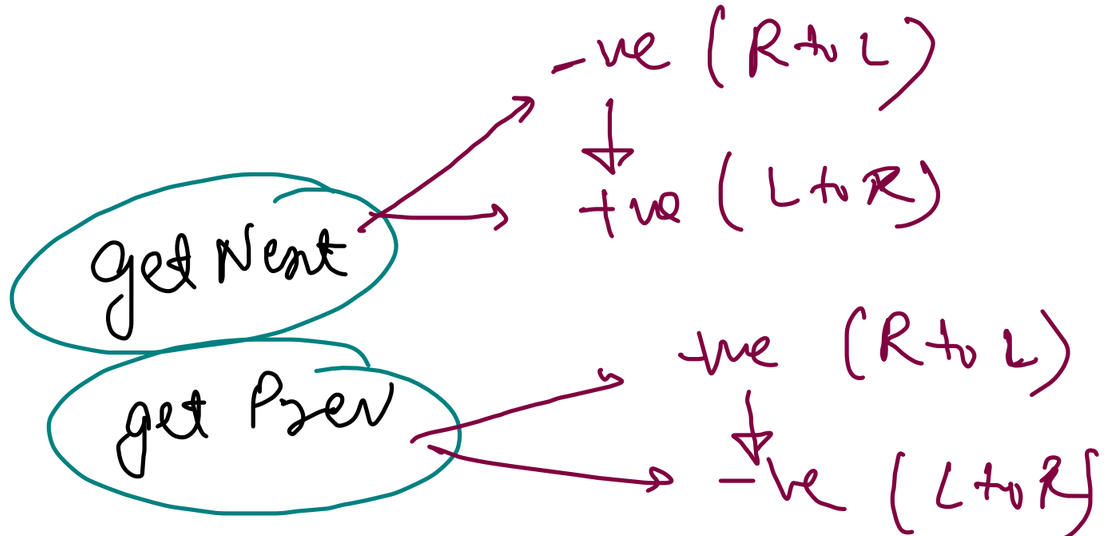
Two Sum - Absolute (lintcode 587) (HW)



int left = ?, right = ?

{-3, 4} {-1, 2}





Lecture (2)

3 Sum

Target Sum Triplets

→ Unique

→ Smaller

→ Closest

Count Valid Triplets

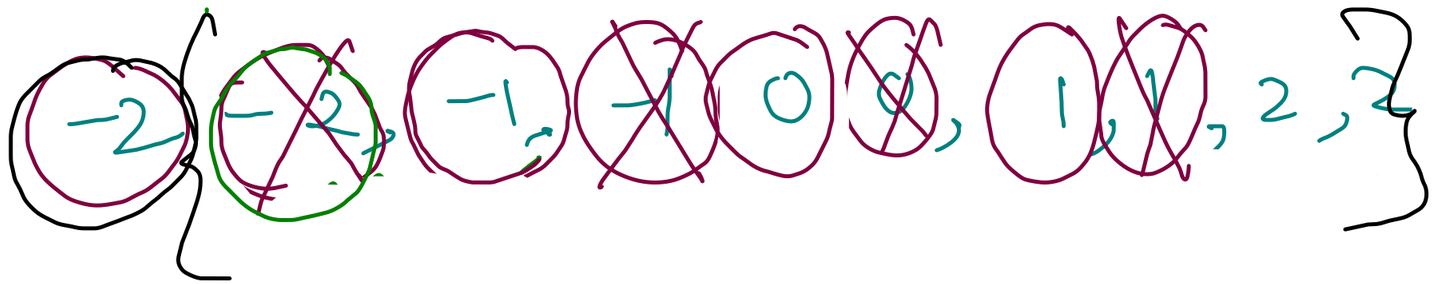
Valid Triangles

Max Sum Triplet

Minimize Differences

K Sum

015 3 Sum- Unique



target = 0

$x+y+z = 0$

-2

① Brute force :→ $O(n^3)$

② 3 pointer :→ Normal loop + Two pointer

$O(N^2)$

{Skip Repeated Elements}

↓ ↓

in normal loop in two pointer {Left}

{first}

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

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

```java
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;
}
```

target = 0

target = 1

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

{-1, -1, 2}

{-1, 0, 1}

```java
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 (Lintcode)

3 Sum Smaller

-2 0, 1, 2, 3

Normal loop

Two ↑ smaller → Smaller

sum < 2

x + y + 3 < 2

-2

y + 3 < 4

(-2) (0, 3)

(-2) (0, 2)

(-2) (0, 1)

(-2) (1, 2)

```java
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;
}
```

$O(N^2)$

```java
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;
}
```

Q16

3 sum closest

↳ Normal loop +
2 Sum - closest

target = 1

-4 -1 1, 2

x + y + z = 1

two pointer

(-4, -1, 1, 2) = |-3 - 1| = 4

(-4, -1, 1, 2) = |-1 - 1| = 2

(-4, -1, 1, 2) = |2 - 1| = 1

abs = ~~+∞~~ 4 2 ans = ~~1~~ 3

sum = -1 + 2 = 1

target = 5

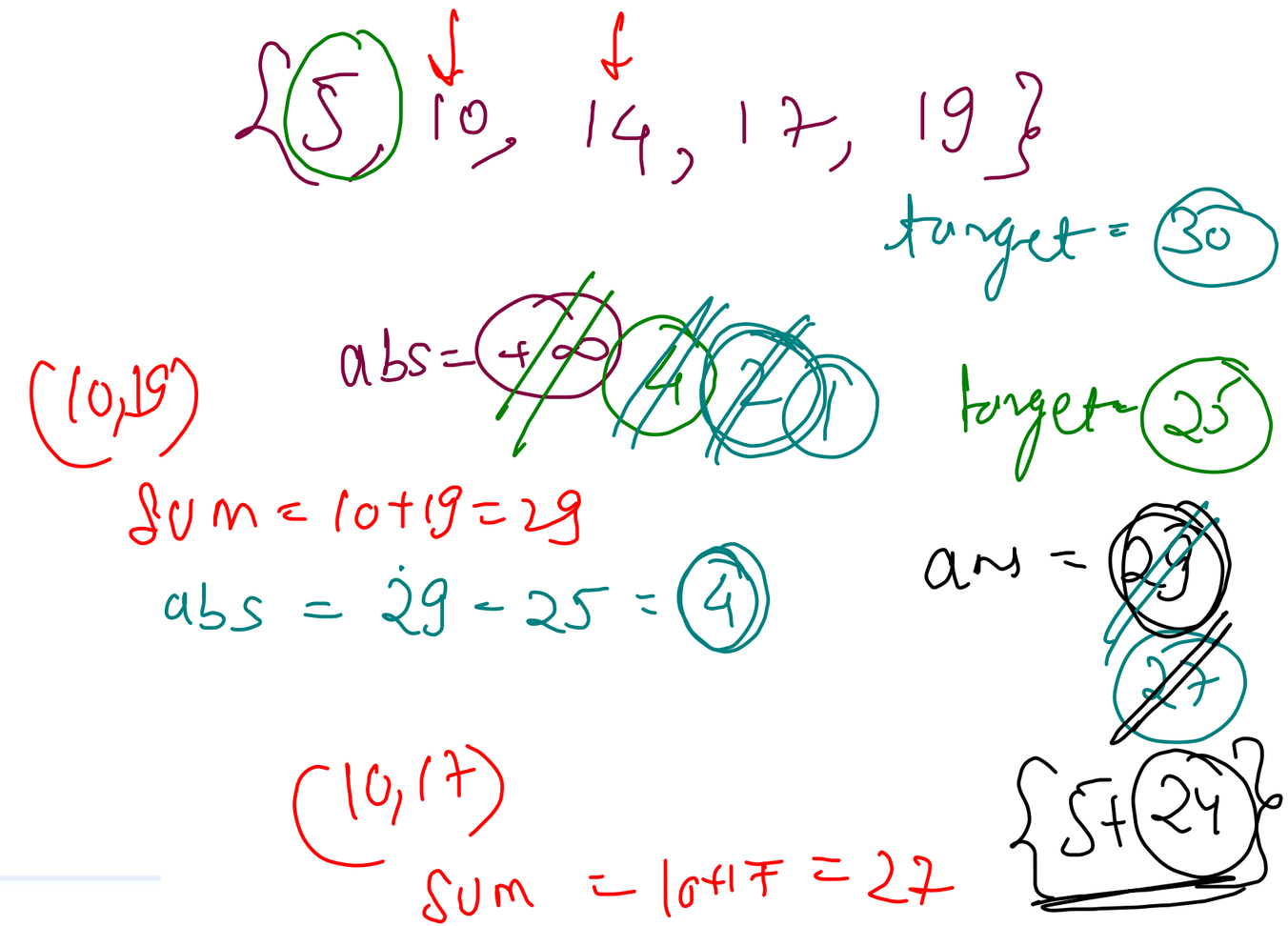
5 - 1 = 4

sum = (-1 + 2) = 3

5 - 3 = 2

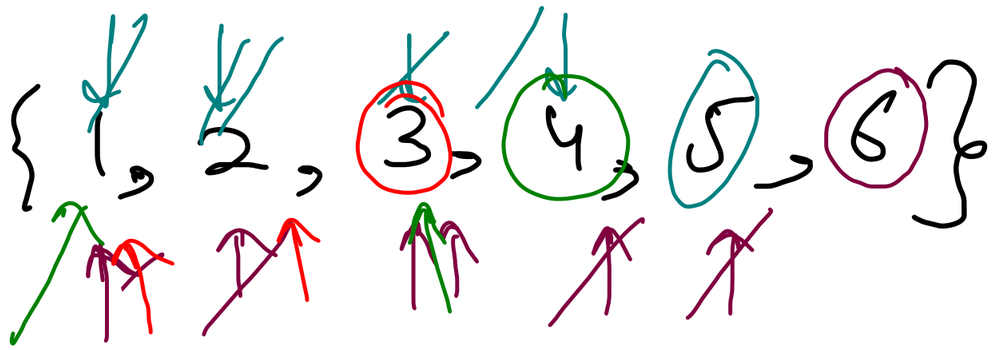
```java
public int twoSumClosest(int[] nums, int left, int target) {
    int right = nums.length - 1;
    int abs = Integer.MAX_VALUE;
    int ans = Integer.MAX_VALUE;

    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;
}
```

```java
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)

(1,4,1)

(1,2,3)

(2,3,1)

(2,4,6)

(1,3,4)

a + b = c

Count triplets

a + b - c = 0

Fix third ele { from right to left }

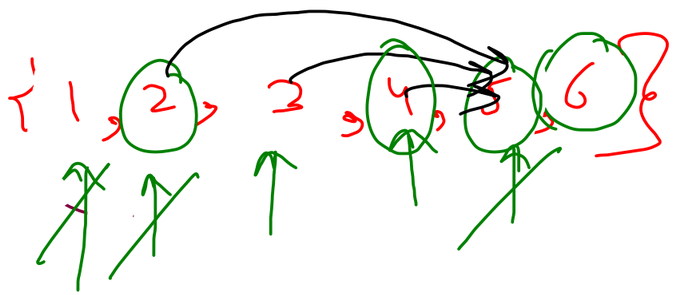
& two pointer on (a+b)

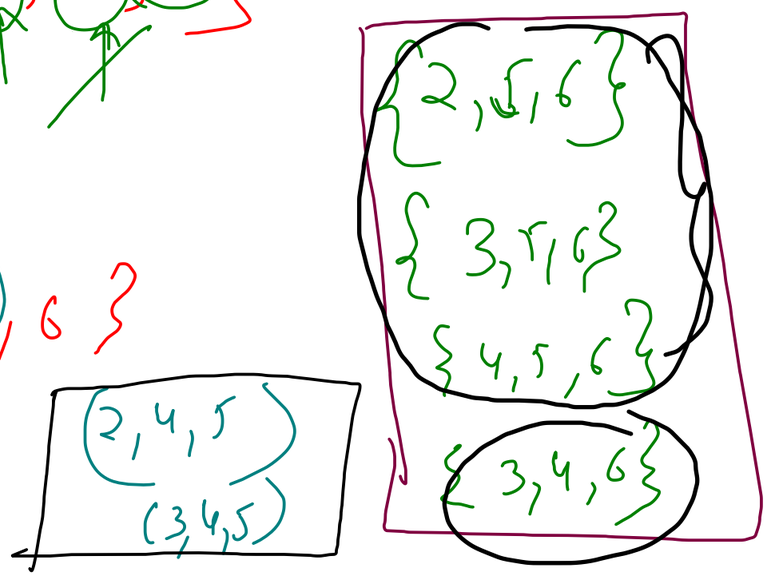
```java
int twoSum(int[] arr, int left, int right, int target){
    int count = 0;
    while(left < right){
        int sum = arr[left] + arr[right];

        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;
}
```

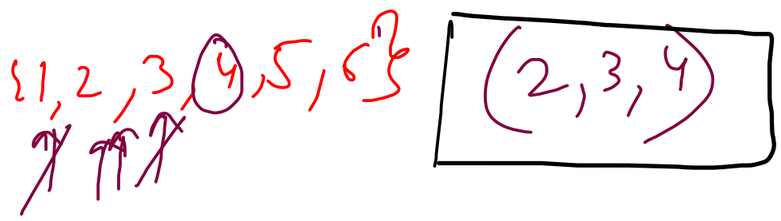
O(n)

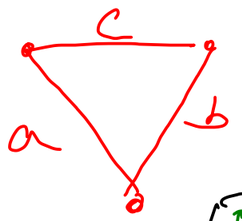
$O(n^2)$

Q611 Count Valid Triangles

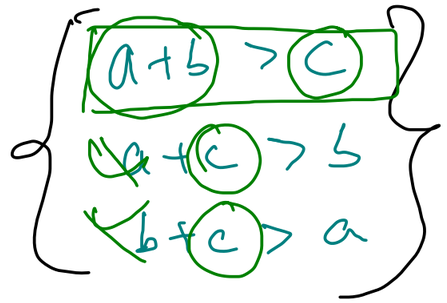






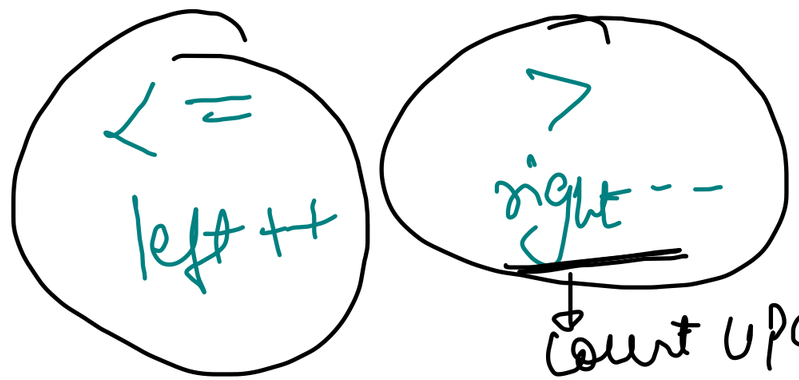


{a, b, c}



a+b > c

Two Sum - Greater



```java
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;
}
```

```java
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;
}
```

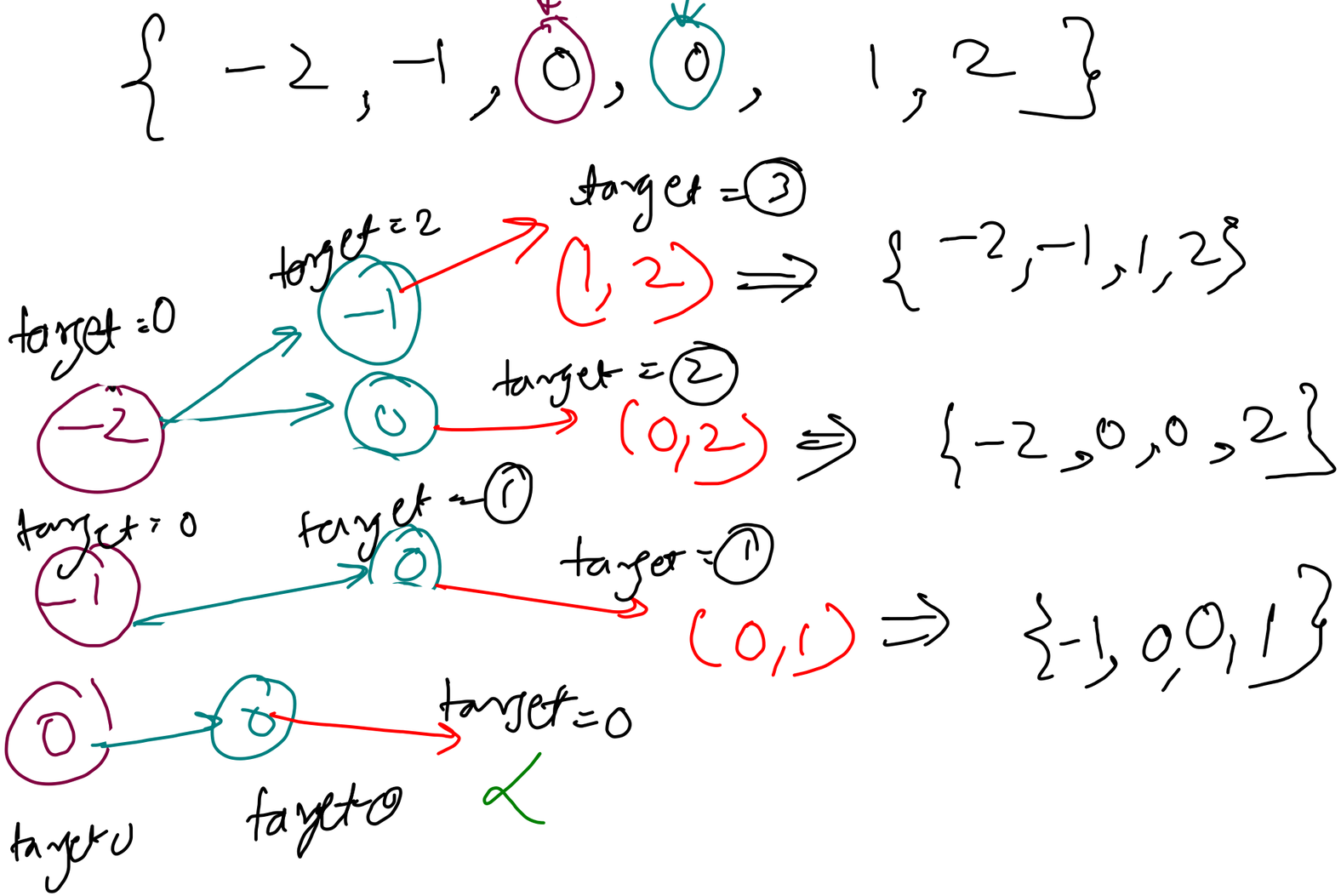
4 Sum = loop + 3Sum = Nested loop + Two pointer

$a + b + c + d = 0$

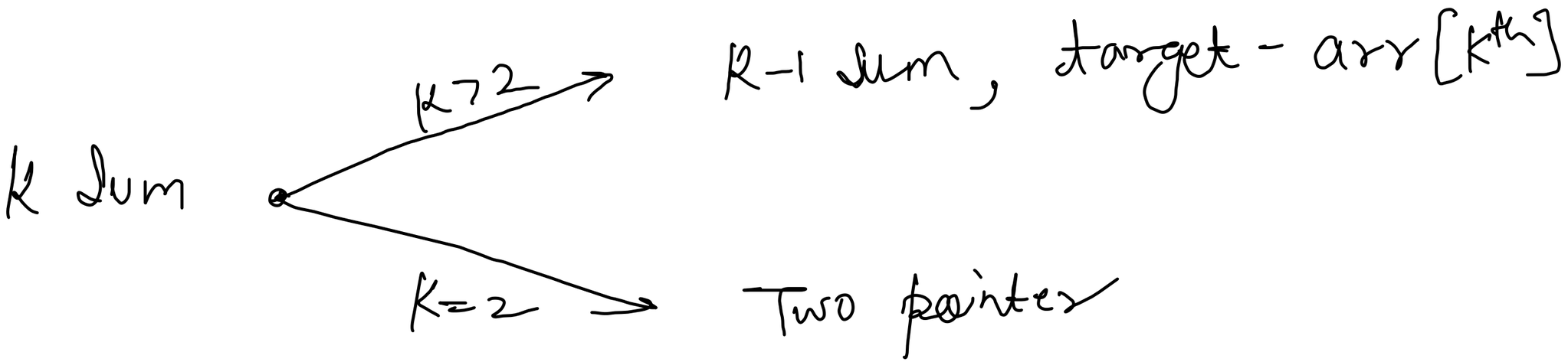
$\{ -2, -1, 0, 0, 1, 2 \}$

① Brute force $\Rightarrow O(N^4)$

② Nested loop + two pointer $\Rightarrow O(N^3)$



K-Sum Recurrence Relation



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

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

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

    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;
}
```

Base case

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

minimum

{1, 2, 10}

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

{1, 8, 9}

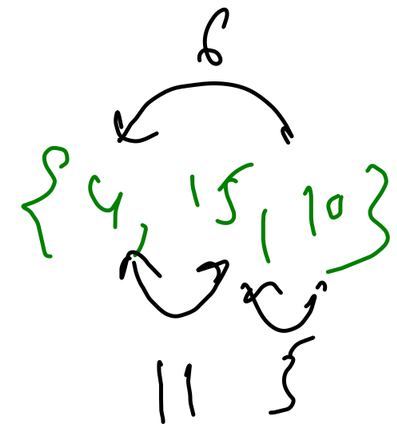
9

{4, 2, 10}

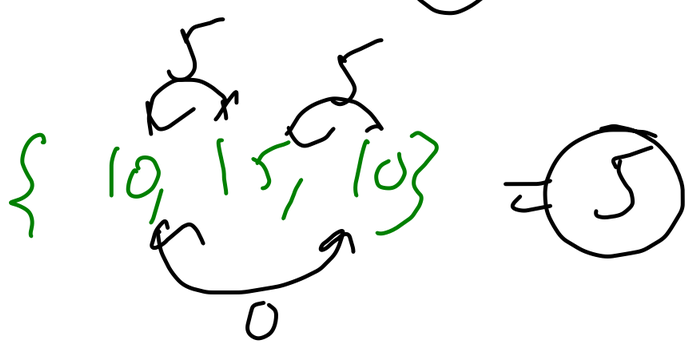
(4,2) (2,10) (4,10)

2, 8, 6

8

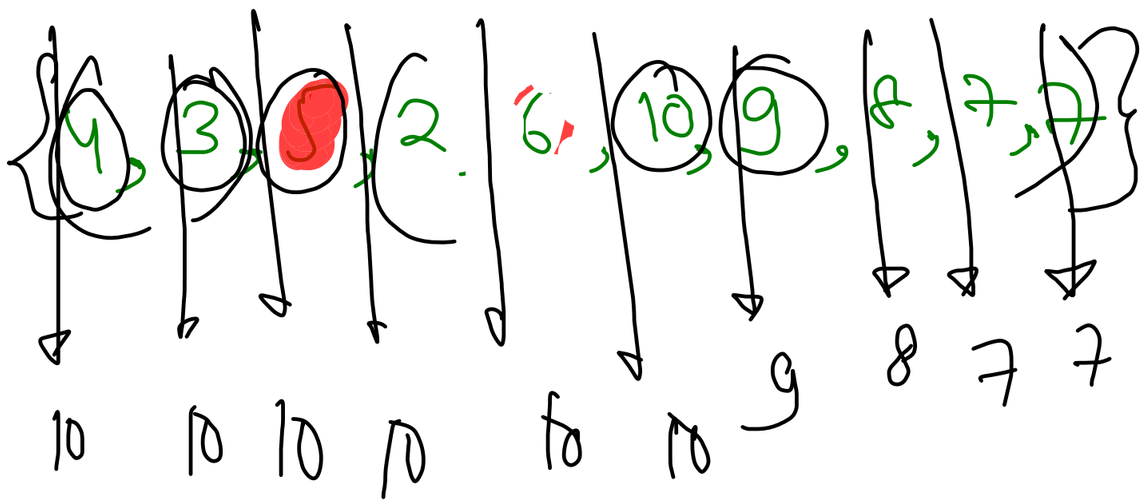


11



```cpp
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;
}
```

$O(n_1 + n_2 + n_3)$



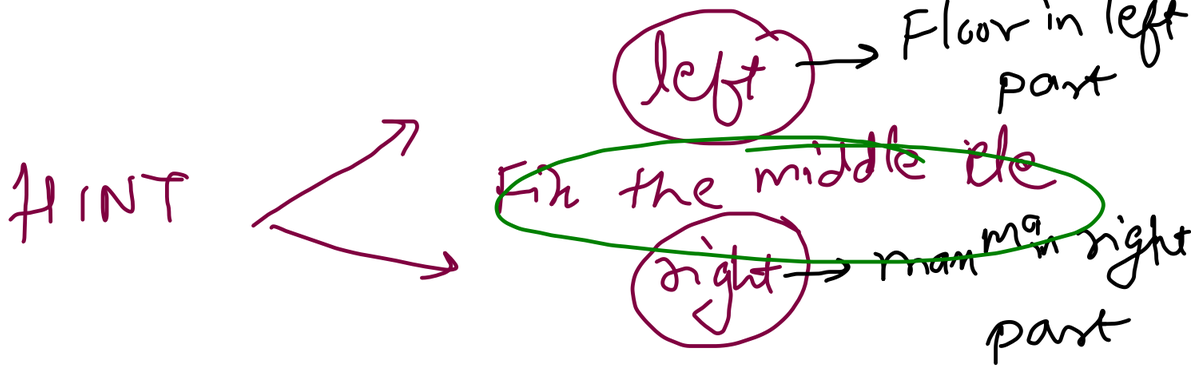
(i, j, k)

(i < j < k)

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

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

HINT



```cpp
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;
}
```

Lecture - (3)

4 Sum

→ 4 Sum (I & II)

→ Count Tuples

→ Sum

→ Product

Difference Pairs

→ Target Diff Pair

→ All

→ Unique

→ longest Diff Pair (I & II)

4Sum -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 → (-2, 1) N^2 pairs

0 → (1, -1) (2, -2)

1 → (2, -1)

C { -1, -1, 2 }

D { 0, 1, 2 } ✓

} Pairs (CD)

-1 → (-1, 0)

0 → (-1, 1) N^2 pairs

1 → (-1, 2)

2 → (2, 0)

3 → (2, 1)

4 → (2, 2)

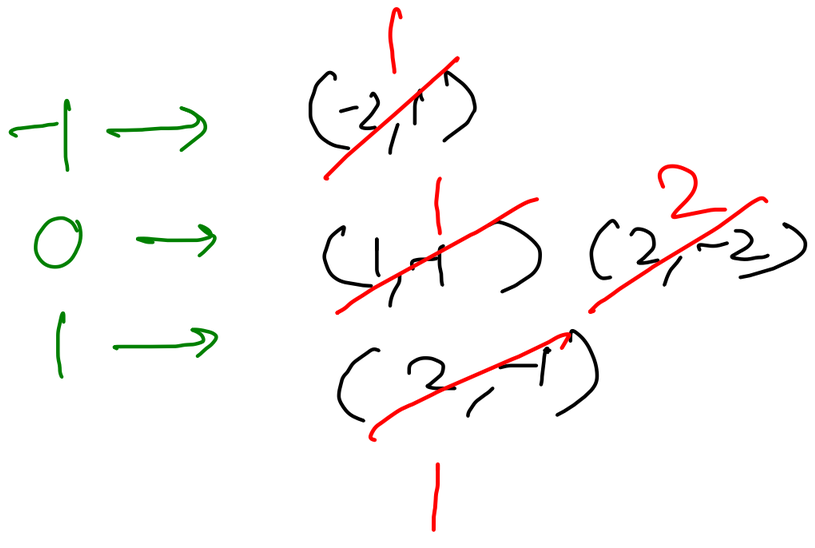
(-2, 1, -1, 2)

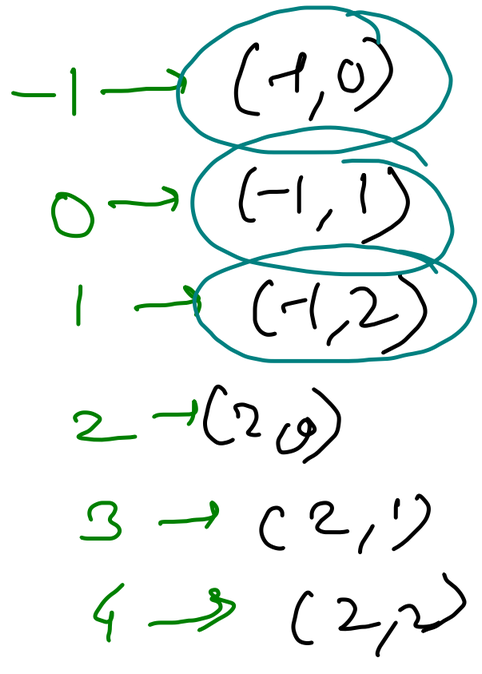
(1, -1, -1, 1)

(2, -2, -1, 1)

(2, -1, -1, 0)

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





Count = 4

```java
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, ~~20~~, 20, ~~60~~, 60, 70}

{10, 20, 60, 70}

{10, 60, 20, 70}

(10,20) 30

(10,20) 0,2

(10,60) 70

(10,60)

(20,20) 40

(20,70) 50 (20,70)

(10,70) 80 (20,60) (20,60) (20,60) (20,60)

(60,60) 120

(60,70) 130 3,5

(60,70) 4,5

{10,70,20,60}

{20,20,60,60}

{20,70,10,60}

{60,60,20,20}

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)

(2,6) (4,3)

(6,2) (3,4)

(6,2) (4,3)

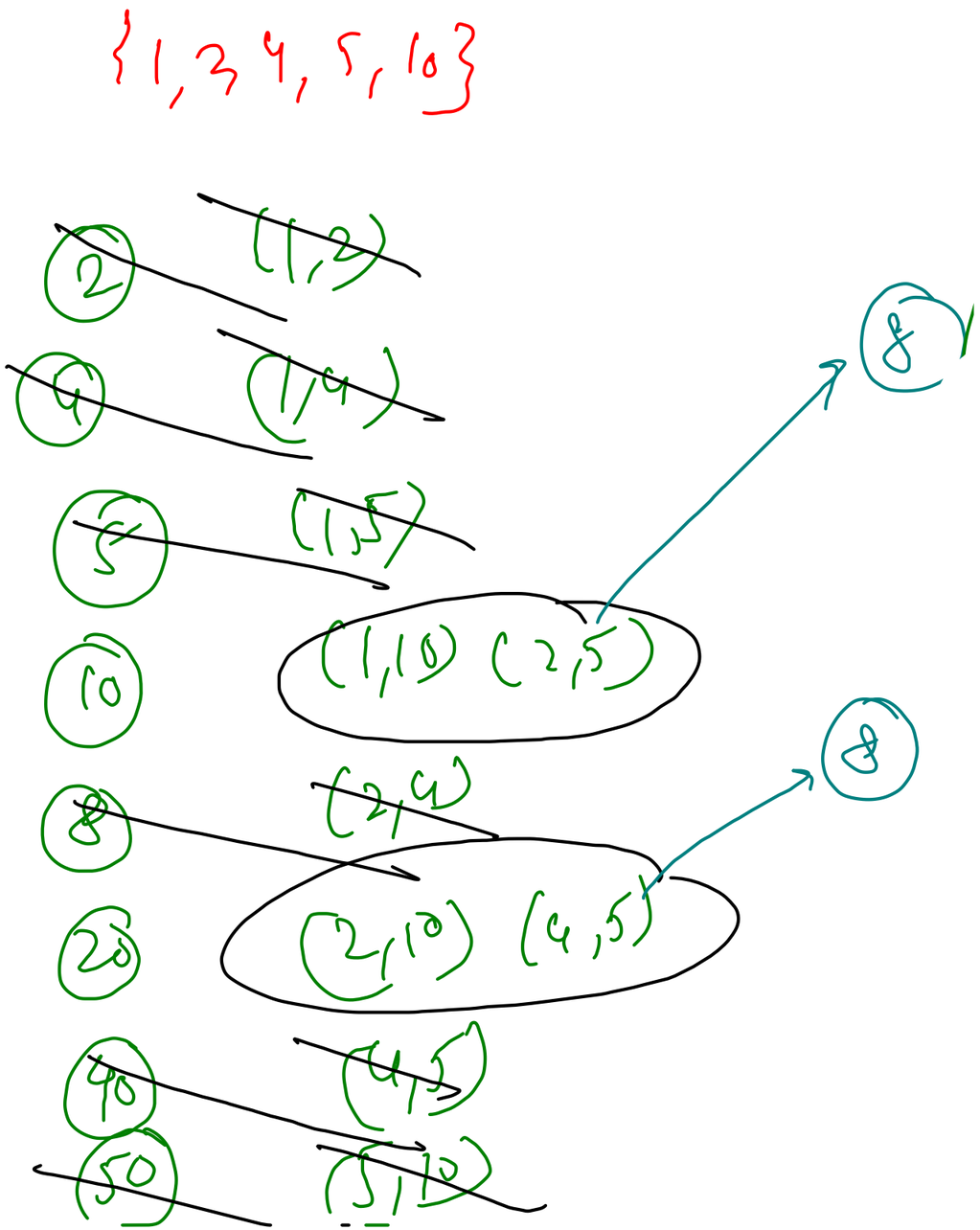
(3,4) (2,6)

(3,4) (6,2)

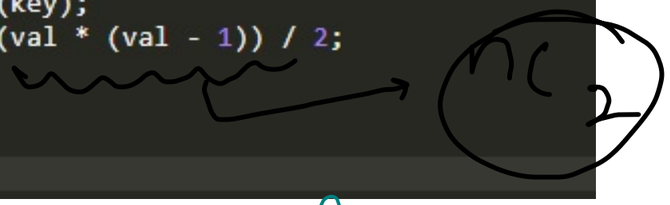
(4,3) (2,6)

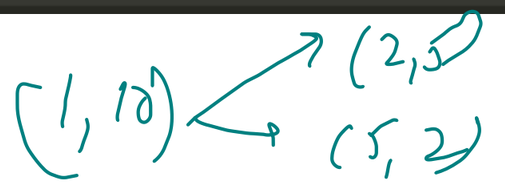
(4,3) (6,2)

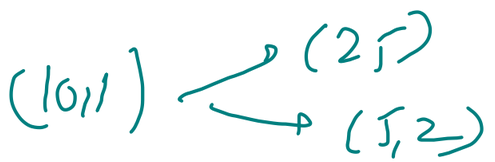
```java
public int tupleSameProduct(int[] nums) {
    HashMap<Integer, Integer> AB = new HashMap<>();

    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;
}
```




```java
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;
}
```







Difference Pair

target = 7

Direct^n ⇒ same

{ 1, 2, 3, 10, 11 }

arr[p2] − arr[p1] = target

11 − 1 = 10

10 − 1 = 9

11 − 2 = 9

8

n[p2] − n[p1]

= 2 − 1 = 1 ↑

p2++

⇒ 3 − 1 = 2 ↑

p2++

⇒ 10 − 1 = 9 ↓

p1++

⇒ 10 − 2 = 8 ↓

p1++

⇒ 10 − 3 = 7

```java
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}

target = 5

M, T, W

p2 ↓ p1 ↓

{ 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 { ptr1 == ptr2 }

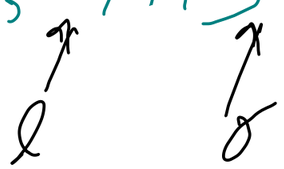
ptr2++

ptr1 != ptr2

Pair found

Q532 Count difference Pairs } Unique }

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



{1,3}

{5,3}

```
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);
}
```

2006 {AI}

K Difference Pair

{1, 2, 2, 1, 3, 4, 3, 4, 3}

0 1 2 3

target = 1

1 → 2

2 → 2 (2, 2)

3 → 3 (3, 3, 3)

4 → 2

target = 2

2 * 2 = 4

2 * 3 = 6

3 * 2 = 6

(6 = 6)

1 2
0 1

1 2
0 2

2 1
1 3

2 1
2 3

Corner case

target = 0

x - y = 0

x = y

$nC_2 = \frac{n*(n-1)}{2}$

```java
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^m width ramp

January & February Schedule

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

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

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

Sunday → 9 AM to 5 PM → DSA {level 2}

→ 9 PM to 12AM → 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
- Atmost K unique
- Exact K unique

→ Subarrays sum in Range

classic

(TWO POINTER)

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

{Max Sum window}

window = (4)

Fixed window size

Static

N = 8

k = 4

{ 5 1 3 4 | 9 8 7 1 }

0 1 2 3 4 5 6 7

sum = 5+1+3+4+9+8+7+1

max sum = 28

```
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 4 2 3 4 5 3 | 4 4 6 2 }

{ 3, 3, 4, 3, 3, 3, 3 }
0 1 2 3 4 5 6

2 → 1

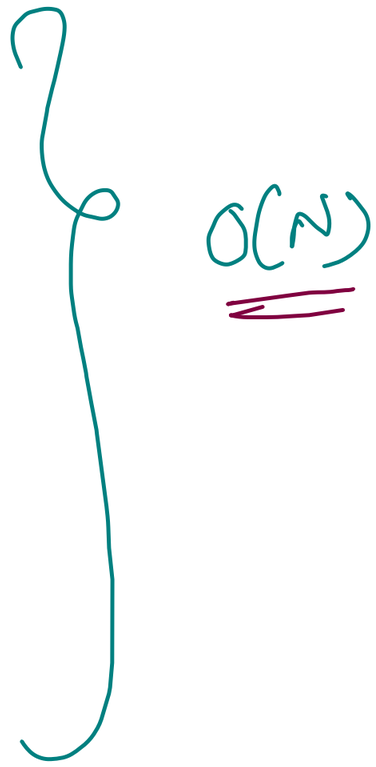
4 → ~~1~~ ~~2~~ ~~1~~
2

6 → 1

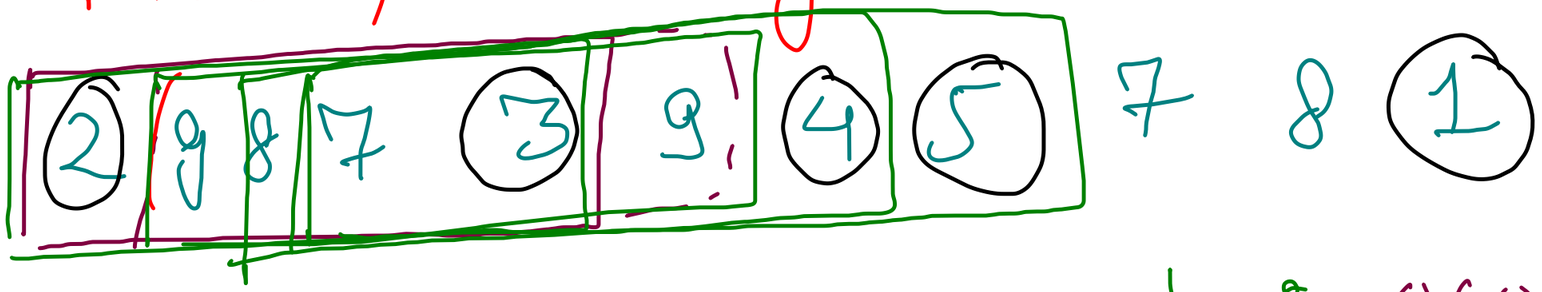
Include
- present → update (freq ++) O(1)
- not present → insert (key, 1) O(1)

Exclude
- freq > 1 { freq-- } O(1)
- freq = 1 { remove } O(1)

```java
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;
}
```



Min swaps to k together



k = 6

① → Calculate window's size O(N)

② → sliding window
↳ total − ele ≤ K } O(N)

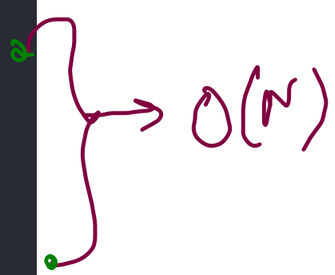
j − 2 = 3

7 9 425 87 ✓

7 254 987 ✓

7987 524 ✓

```java
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;
}
```

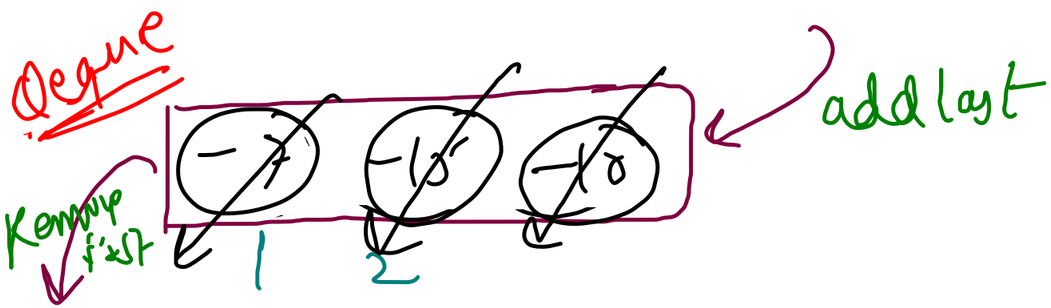


First -ve In Every window of Size (K)

window = (4)

{12, -1, -7, 8}, -15, -10, {30, 16, 28, 402}

0 1 2 3 4 5 6 7 8 9



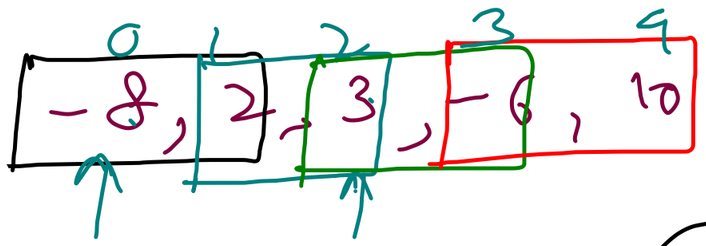
-1, -1, -7, -15, -15, -10, 0

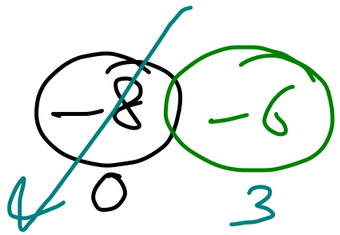
Exclude → -ve → remove first
+ve → do nothing

Include → -ve → add last
+ve → do nothing

answer → que empty → 0
else q.getFirst()

```java
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;
}
```





$\leq 4-2$

O(N)

continue at 10:20

{Anagram Permutation} in String

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

"abcc"
a→1
b→1
c→2

"bcca"
a→1/0/1
b→1
c→2

"ccaa"
a→2
c→2

"caab"
a→2
c→1
b→1

"aabd"
a→2
d→1
b→1

"abdd"
a→1
b→1
d→2

"bdda"
b→1 d→2 a→1

"dacb"
a→1 c→1
b→1 d→1

"abcd
a→1 | b→1 | c→1
// a→1

4!
bacd
bcda
cabd
!
!

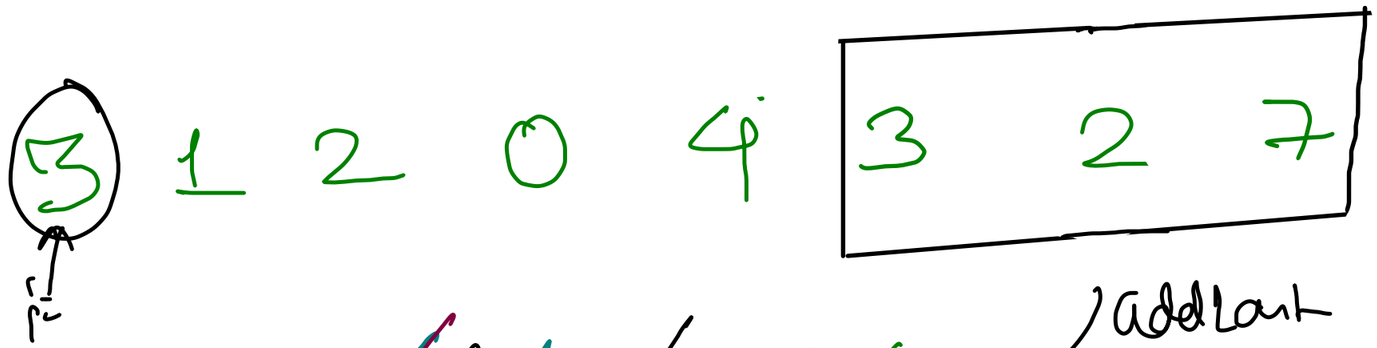
```java
public boolean isEqual(int[] a, int[] b){
    for(int i=0; i<26; i++){
        if(a[i] != b[i]) return false;
    }
    return true;
}
public boolean checkInclusion(String s1, String s2) {
    int[] reqFreq = new int[26];
    int window = s1.length();

    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;
}
```

$O(26 * N)$

Sliding Window Maximum

Next Greater / Stack → O(N)

Deque (Sliding window)



window = 3

→ Deque

removeFirst → out of window

removeLast → val < addval

addLast → always

deque → chronological order

decreasing order

max → queFront

```java
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
→ 1 pop for each ele

(DYNAMIC WINDOW)

Lecture (5)

{ Sunday Morning }

→ Max Consecutive ones (I, II & III)

→ Minimum Window Substring (I & II)

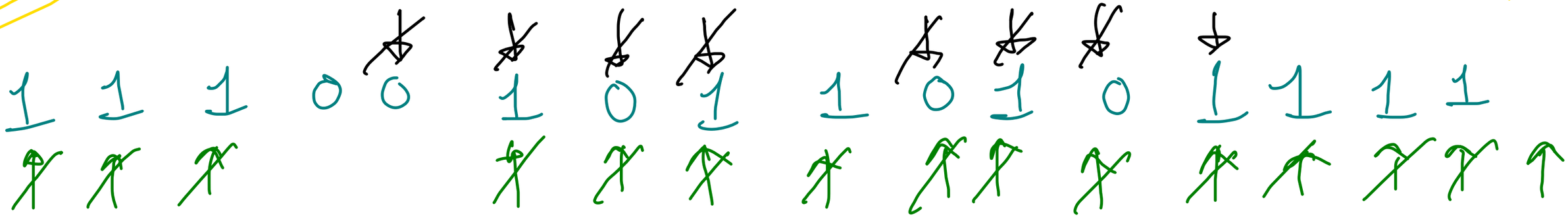
→ Substring with W/o Repeating

LC

485 Max - Consecutive Ones - (1)

count = 3/4

k = 0




```java
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)

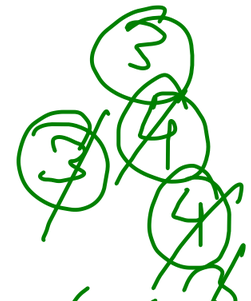
LC Q1004

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].
```

k = 3



```java
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;
}
```

currZeros = 0 1 2 3

maxLen = 0 1 2 3 4 5 6 7 8

(Q76 leetcode)

Minimum window substring

{ IMPORTANT }

O(2*n)

pattern

a b b c d c

a → 1
b → 2
c → 2
d → 1

d b a e c b b a b d c a a f b d d c a b g b a

d=1 b=3 a=2 c=1 2

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

mC = 1

3 4

```
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++;
    }
}
```

right++ ⇒ making invalid substring valid

left++ ⇒ making valid substring of min length

min = 6

(03) (leetcode)

"Longest Substring without repeating characters"

k = 1

longest
valid → include
Invalid → Exclude

a b b a c b c d a e b

Hashmap

a → ✗ ✗ 1
b → ✗ ✗ 1
c → 1
d →
e →

main
count = ✗ ✗ ✗ ✗ ✗ 3 {right - left + 1}

a, ab, b¹, b², ba, a, bac, ac, c

Max length

O(N)

Count valid

```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;
}
```

```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;
}
```

lecture (6)

3pm to 6pm

{Sunday Evening}

1 → Substring with
- Almost K unique
- Exact K unique

2 → K odds → K Repeating

→ Repeating Replacement

3
- Smallest Subarray with Sum ≥ X
- Maximum Sum Subarray < X

4
- Count Subarrays with Sum < X
- Count Subarrays with Product < X

5
- Count Subarrays with Maximum > X
- Count Subarrays with Max in Range

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;
```

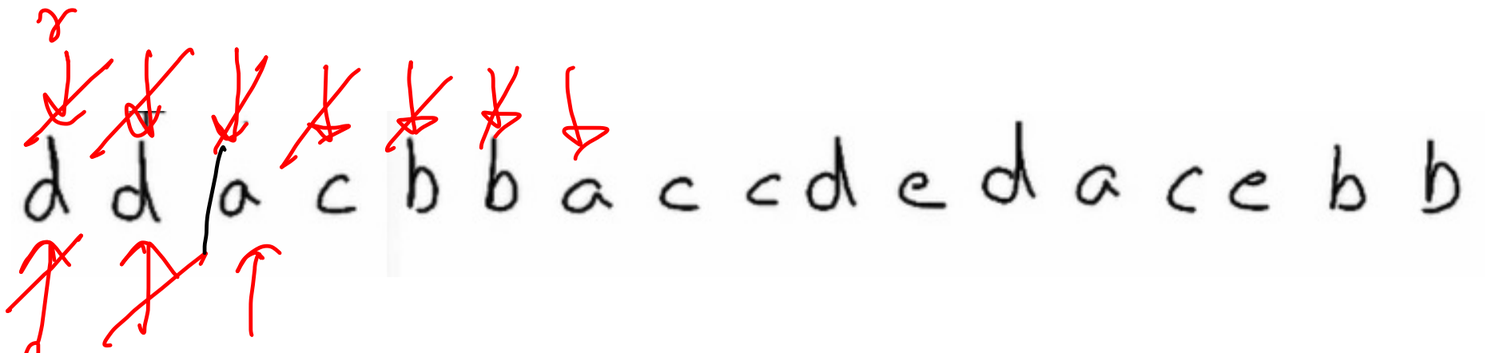
Exact k → longest

```java
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;
```

Exact k → count

same logic will not work for count

longest → Substring with atmost k unique characters
Count



k = 3

d →
a →
c →
b →

maxLen = 1 2 3 4 5

count = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 22

d, d, dd, dda, da, a
ddac, dac, ac, c, acb, cb, b, acbb, cbb, bb, 1

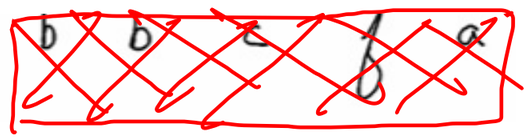
Invalid → left ++ (exclude)

valid → right ++ (include)

→ Atmost K unique − Atmost (k-1) unique

Count → Exact k unique characters

a b c a b d a



k = 3 − 2

abca

(10)

a, b, c, a

ab, bc, ca

abc, bca

abca

(3)

abca

bca

abc

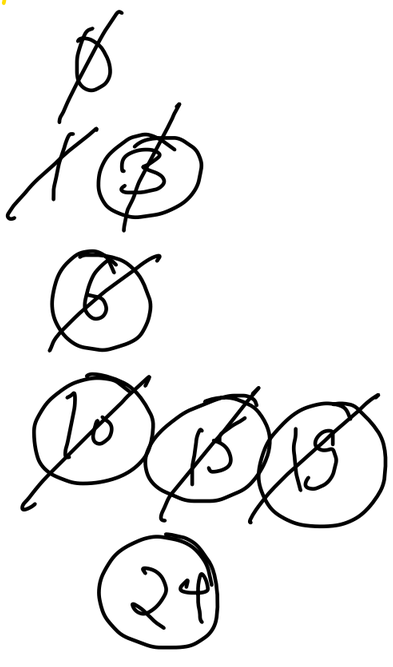
a, b, c,

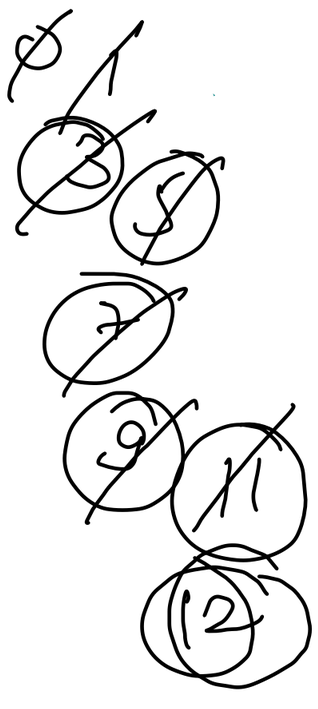
ab, bc, ca

Exactly 3 = Atmost 3 − Atmost 2

24 − 12

⇐ (12)





① → "a"

② → "ab"
"b"

③ → "abc"
"bc"
"c"

④ → "abca"
"bca"
"ca"

```
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

Exactly k = Atmost k odd no's − Atmost(k−1) odd no's

Even No's : Do not Insert

```cpp
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

k = 3

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

26 times
two pointer
O(26N) = O(N)

A → longest Substring with atmost k (!= A) char

B → longest substring with atmost k (!= B) char

'a', 'b', 'c', 'd' ----->

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

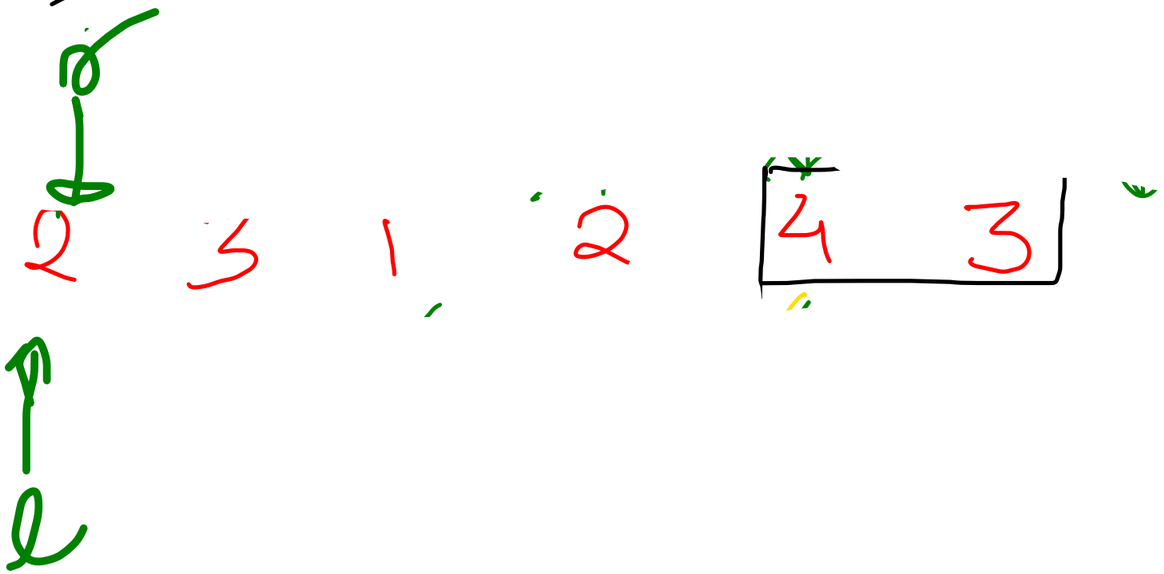
Max Consecutive Ones

```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++;

        // 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 leetcode — Smallest subarray with sum ≥ target

target = 7 → Restriction



sum = 0 + 2 + 3 + 1 + 2 − 2 + 4 − 3 − 1 + 3 − 2 − 4

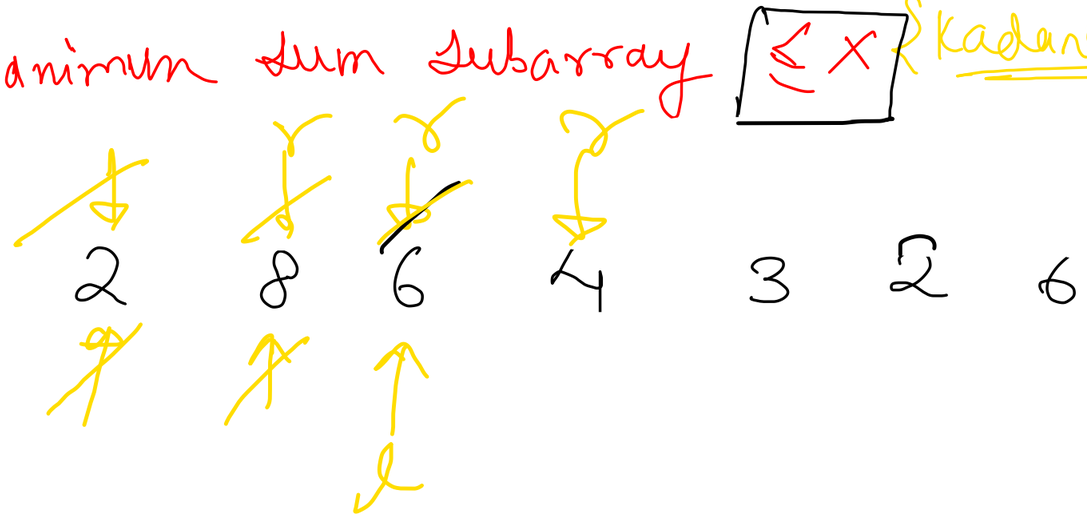
minlen = ∞ 4 3 2

right++ → Invalid subarray (sum < target)

left++ → Smallest valid subarray

```java
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 ≤ X (Kadane's)



target = 7

Sat λ Sun { DSA 5:30 + SD 2:30 = 8hrs }

sum = 2 10 8 6

maxSum = 0 ≠ 6

valid (Include)
right ++ ⇒
Sum <= target

invalid (Exclude)
left ++
Sum > target

```
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;
```

Two Pointer - lecture (7)

713 LC Subarray Less than K

Product = 100

curr = 10 : 10*5 = 5*2, = 100/10 = 10*6 = 60

count = 0 1 2 3

right

10, 5, 2, 6

left

```
if(k <= 1) return 0;

int left = 0;
int product = 1, count = 0;

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

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

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

return count;
```

795. Number of Subarrays with Bounded Maximum

Medium 1476 85 Add to List Share

3 9 4 12 2 1 6 9

right

left

Valid → Inclusion

Invalid → Exclusion

max = 3 9

count = 1 2 6 10 15

{3} {3,9} {9}

{3,9,4} {9,4} {4}

{3,9,4,12} {9,4,12} {12}

left = 3 → right = 8

[3, 8] ← maximum of subarray should lie in this range

count of subarrays

Max (2) − Max (8)

{0,1,2,3,4,5,6,7,8} − {0,1,2}

= {3,4,5,6}