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

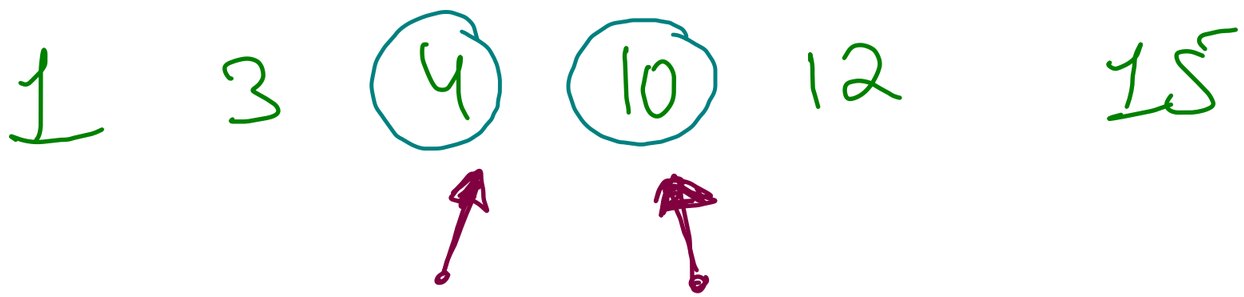
→ O(N)T, O(N)S
Hashmap

→ 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

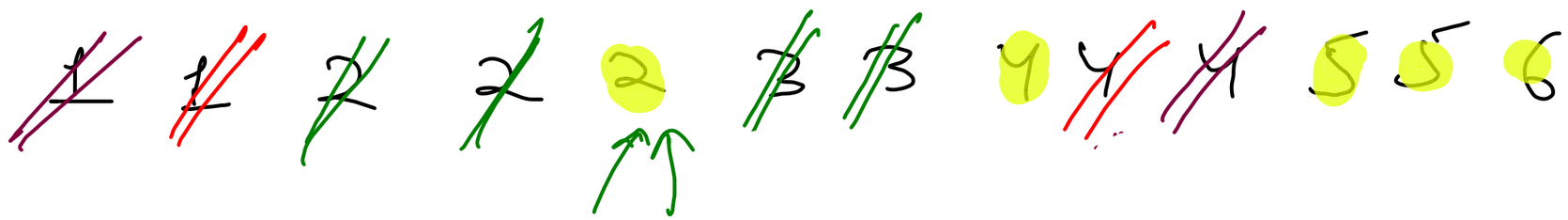
↑3+10 < 14

```
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;
```

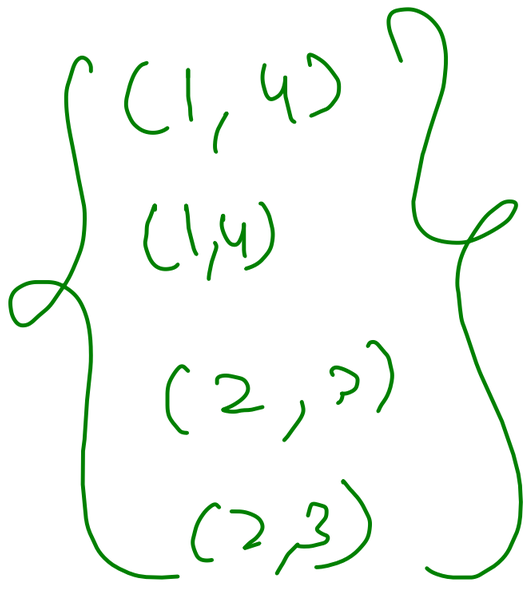
Q(1679)

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)

Lintcode (387)

Unique Target Sum Pairs

target = 5

1 1 2 2 2 3 3 4 4 4 5


```java
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

bintade
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

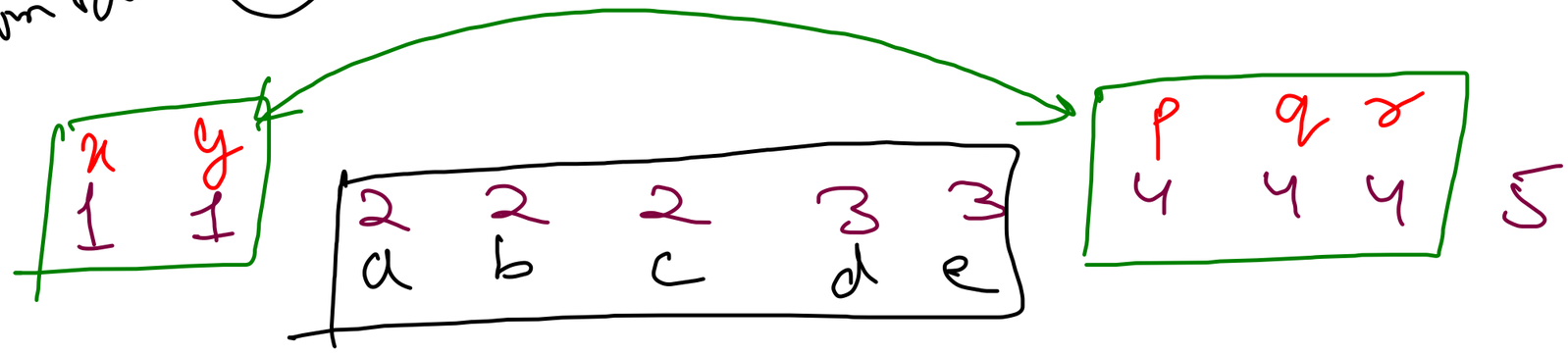
5+12 = |17-14| = 3

5+10 = |15-14| = 1

5+7 = |12-14| = 2

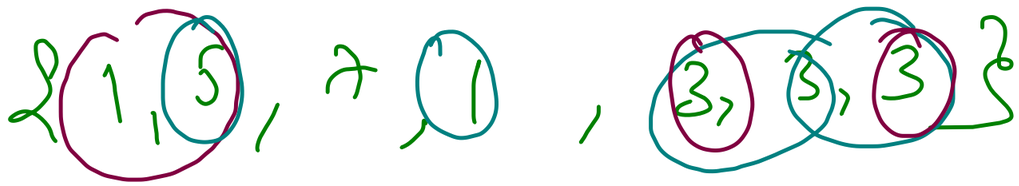
GFG
Target Sum Pair (11)

target = 5



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

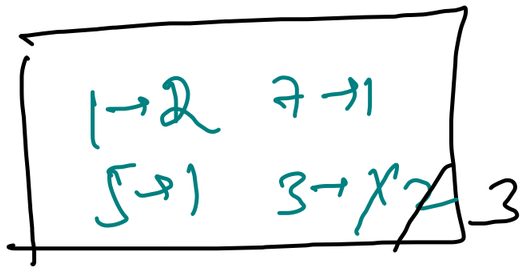
(y,p) (y,q) (x,r) (a,e) (b,e) (c,e) 2f * 3f = 3 * 2



target = 6

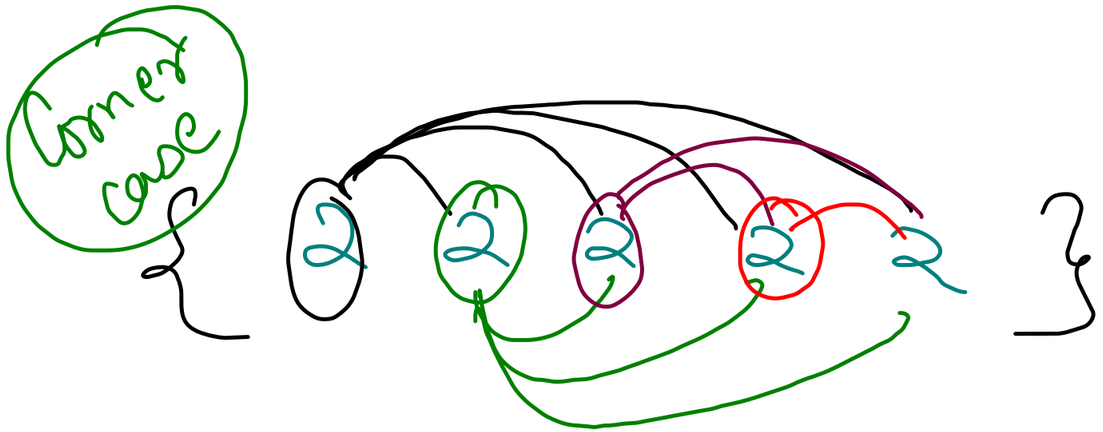
```java
HashMap<Integer, Integer> hm = new HashMap<>();

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

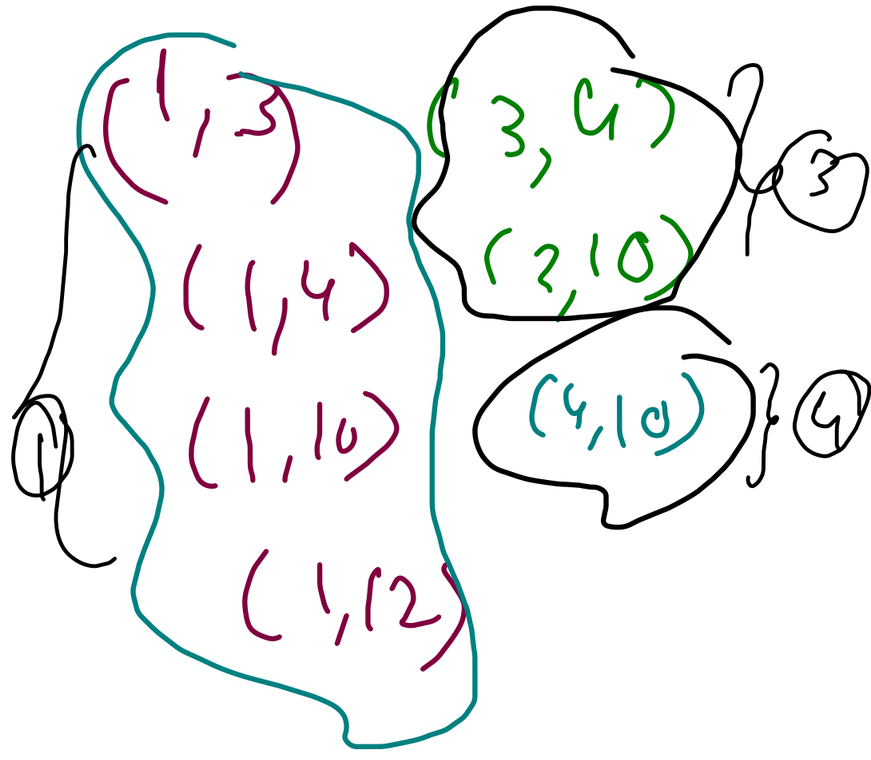
↑ 1+12 < 14 ⇒ (4)

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

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;
```

target > 14

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

(12,15)

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

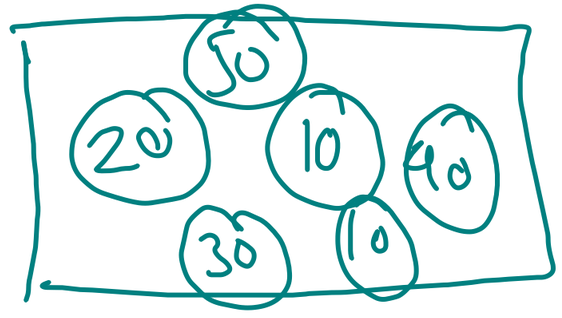
Two Sum - Design / Stream / Online

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

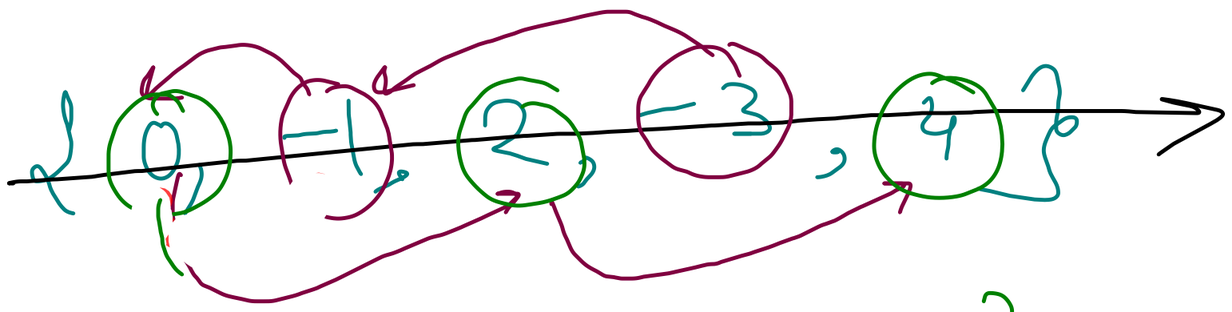
add → frequency update → { O(1) avg case, O(N) worst case (collisions)

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

Corner case → equal complement → freq >= 2

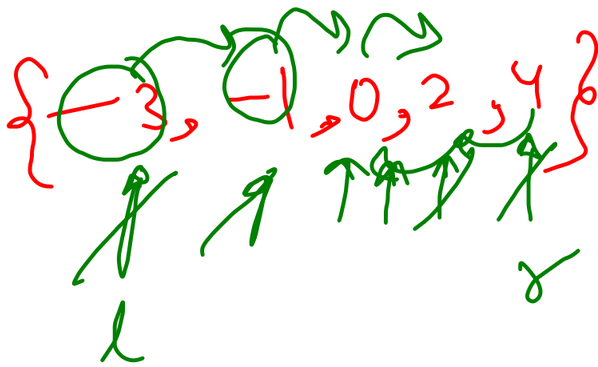
```java
public class TwoSum {
    HashMap<Integer, Integer> freq;

    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;
    }
}
```

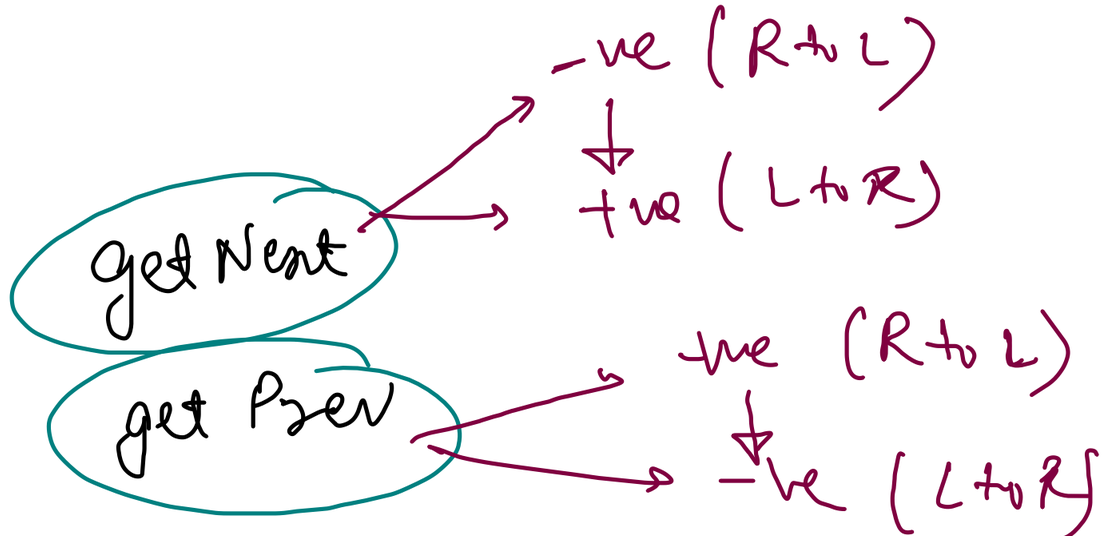
Two Sum - Absolute (lintcode 587) (HW)



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

4 Sum

→ 4 Sum (I & II)

→ Count Tuples
- → Sum
- → Product

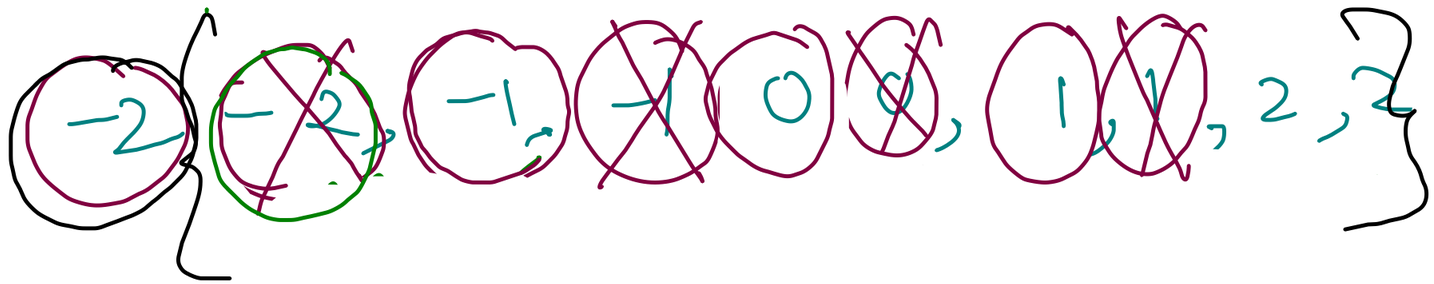
K Sum

Difference Pairs

→ Target Diff Pair
- → All
- → Unique

→ longest Diff Pair (I & II)

015 / 3 Sum - Unique



target = 0

$x + y + z = 0$

$x = -2$

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

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

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

abs = ~~+∞~~ 4 2 ans = ~~1~~ 3

target = 5

sum = 1 + 2 = 1

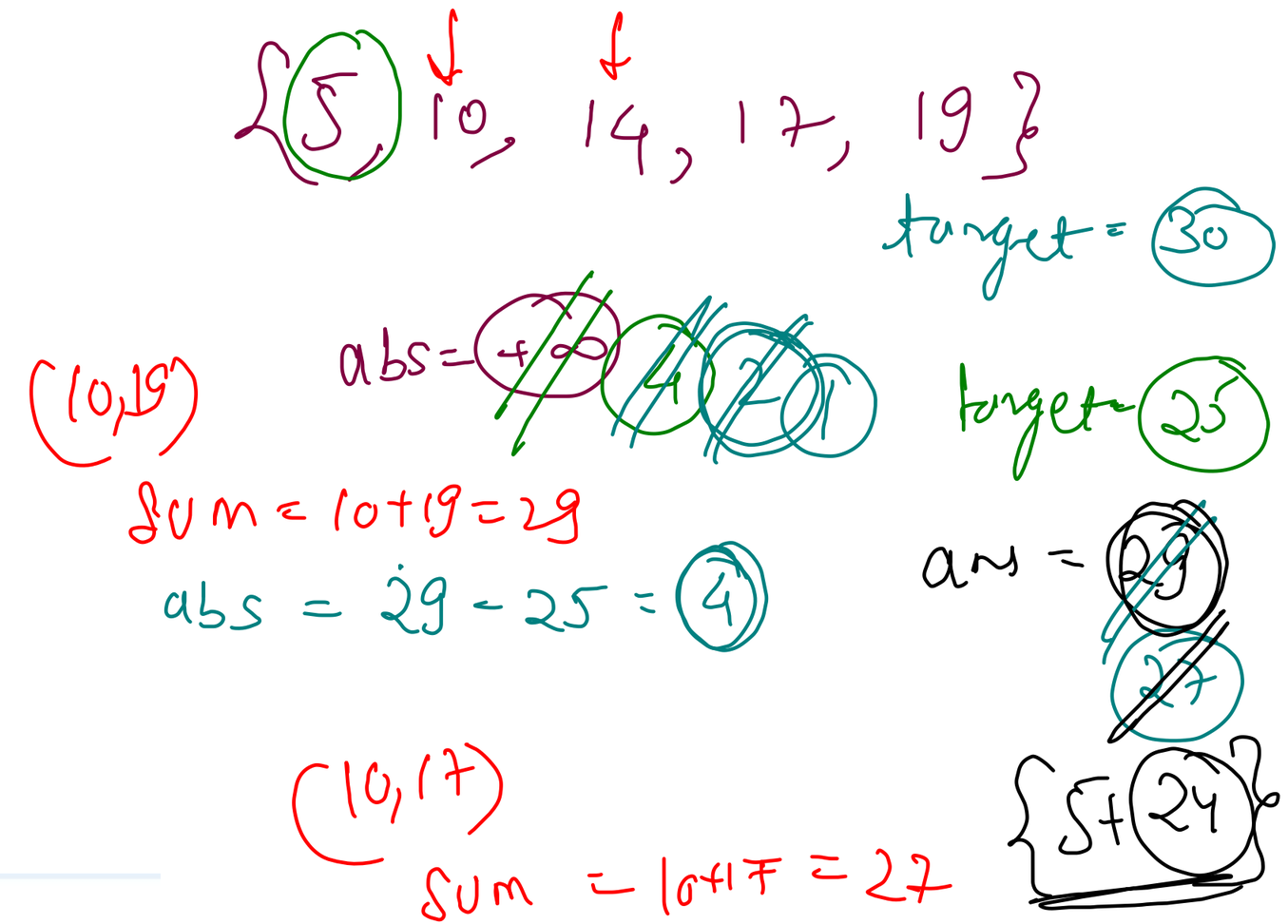
5 - 1 = 4

sum = (1 + 2) = 3

5 - 3 = 2

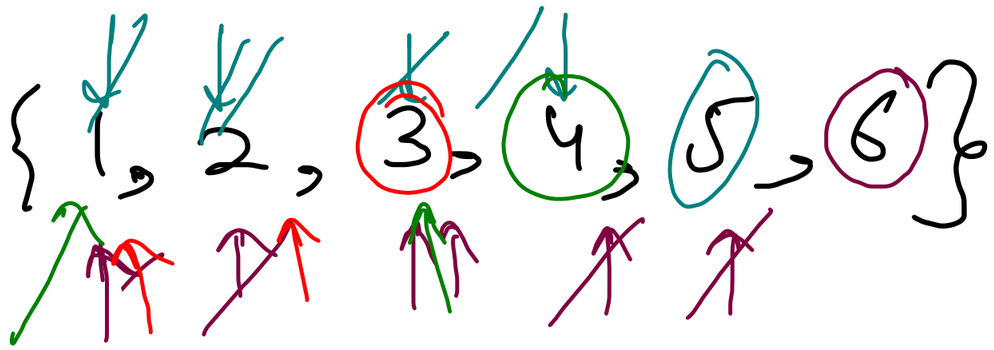
```java
public int twoSumClosest(int[] nums, int left, int target) {
    int right = nums.length - 1;
    int abs = Integer.MAX_VALUE;
    int ans = Integer.MAX_VALUE;

    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 }

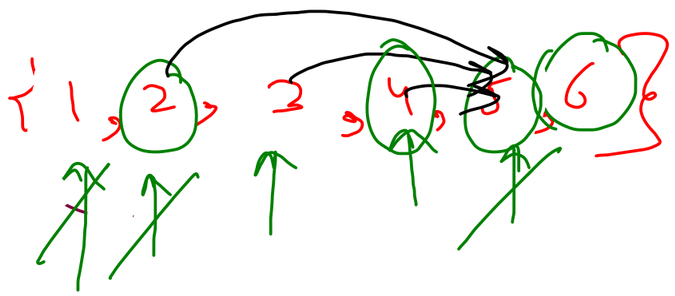
& two pointer on (a+b)

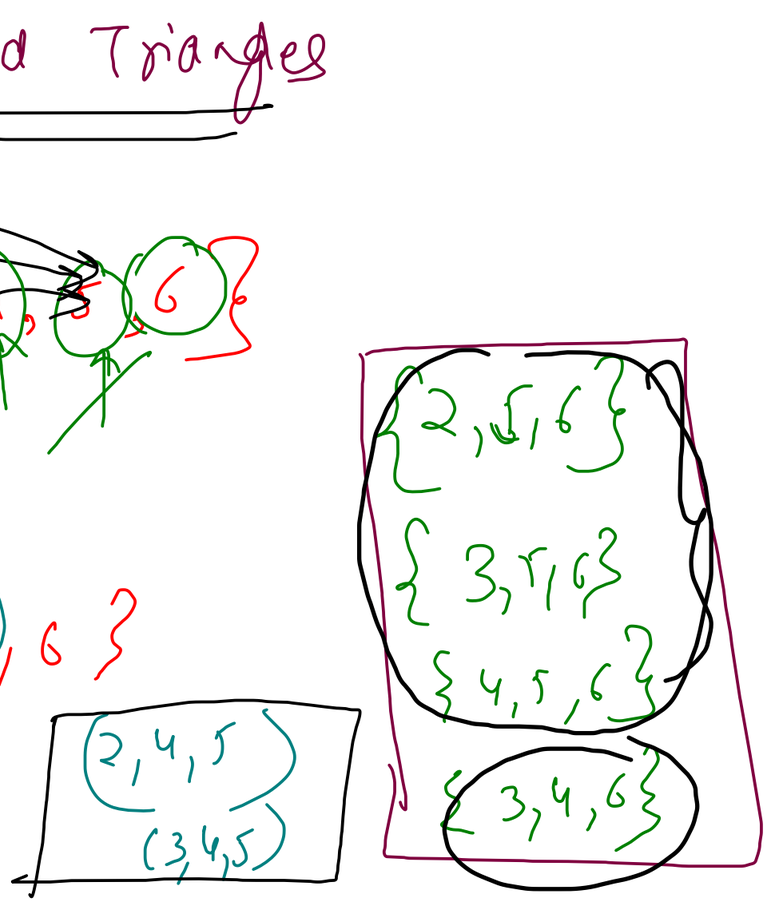
```java
int twoSum(int[] arr, int left, int right, int target){
    int count = 0;
    while(left < right){
        int sum = arr[left] + arr[right];

        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;
}
```

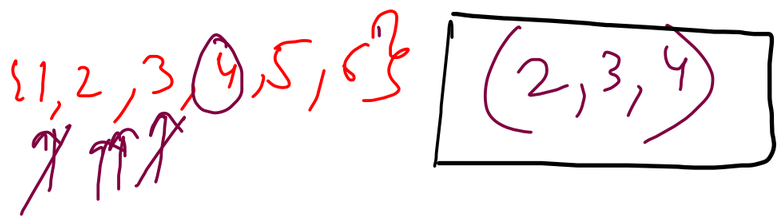
O(n)

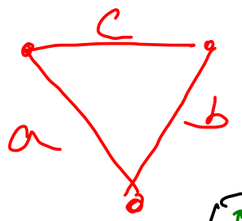
$O(n^2)$

Q611 Count Valid Triangles

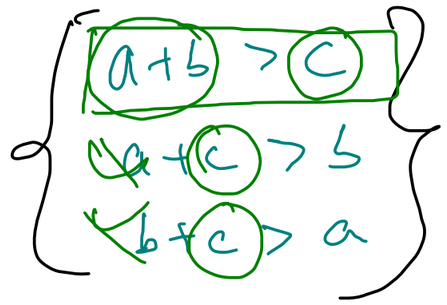






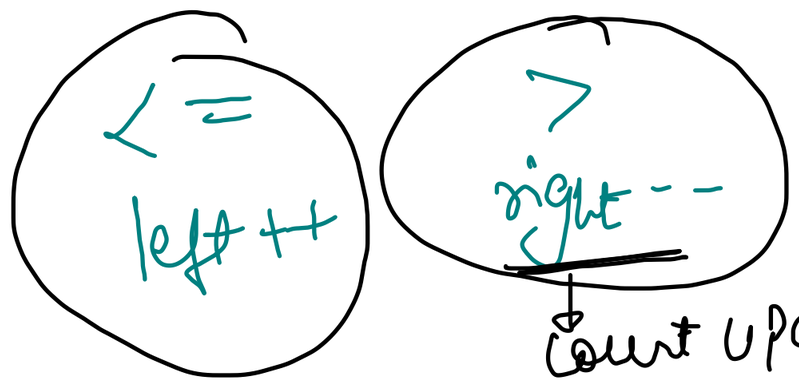


{a, b, c}



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;
}
```

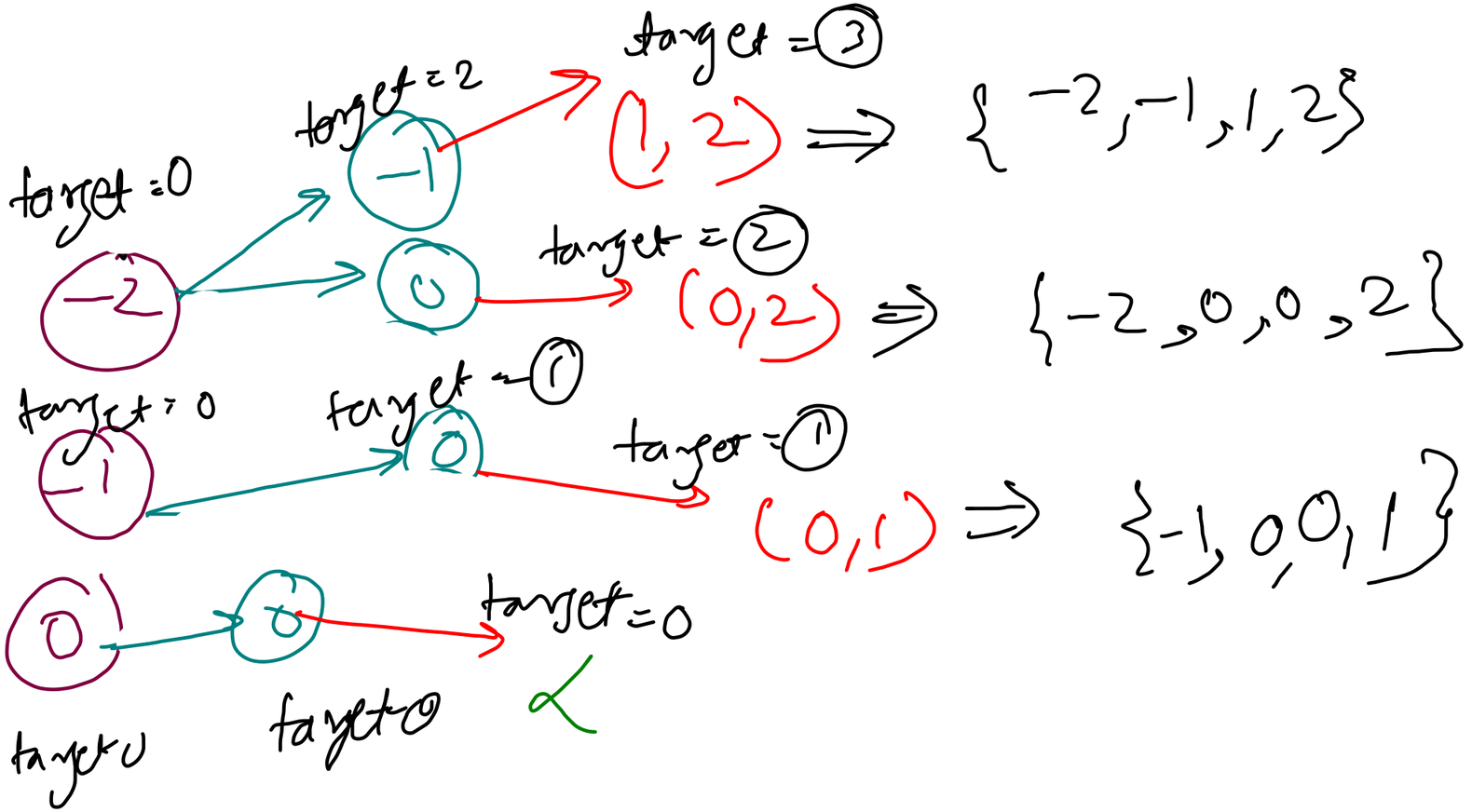
4 Sum = loop + 3Sum = Nested loop + Two pointer

$a + b + c + d = 0$

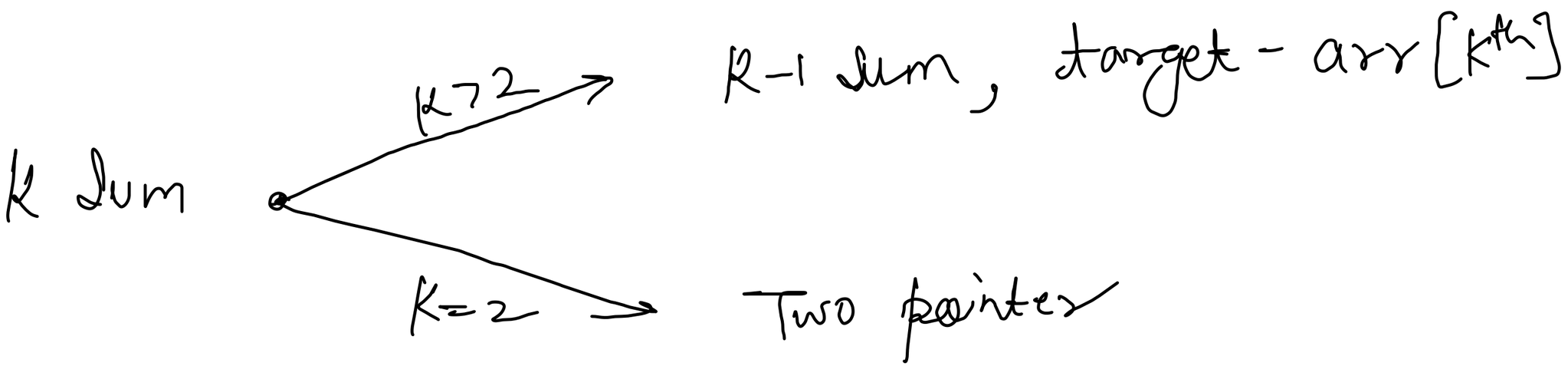
$\{ -2, -1, 0, 0, 1, 2 \}$

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}

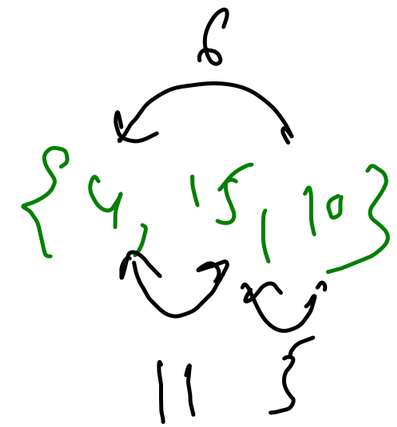
9

{4, 2, 10}

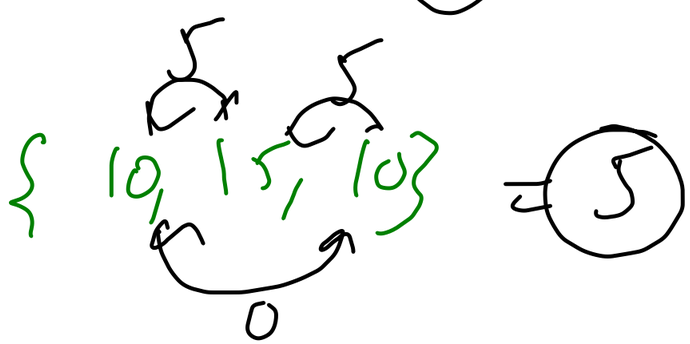
(4,2) (2,10) (4,10)

2, 8, 6

8

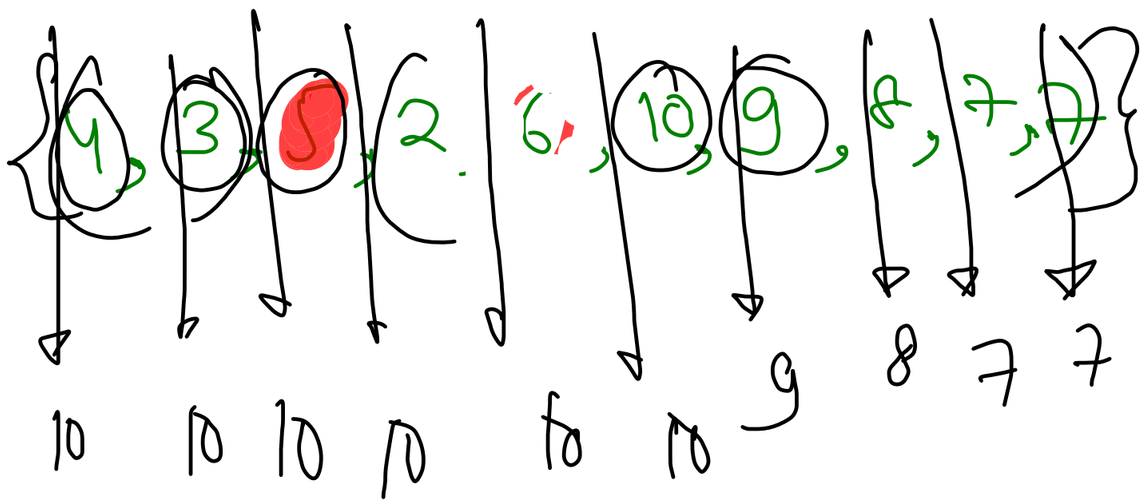


11



```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;
}
```

$O(n_1 + n_2 + n_3)$



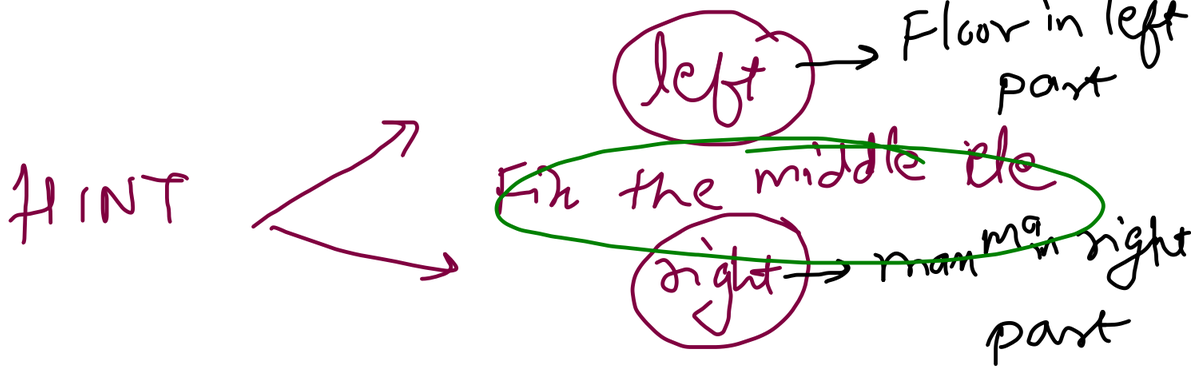
(i, j, k)

$(i < j < k)$

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

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

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;
}
```