HashMap 2

Sorting → Custom comparator
→ Merge Sort & Quick sort

Searching → Binary Search on Answer

Two pointers

Content for Today

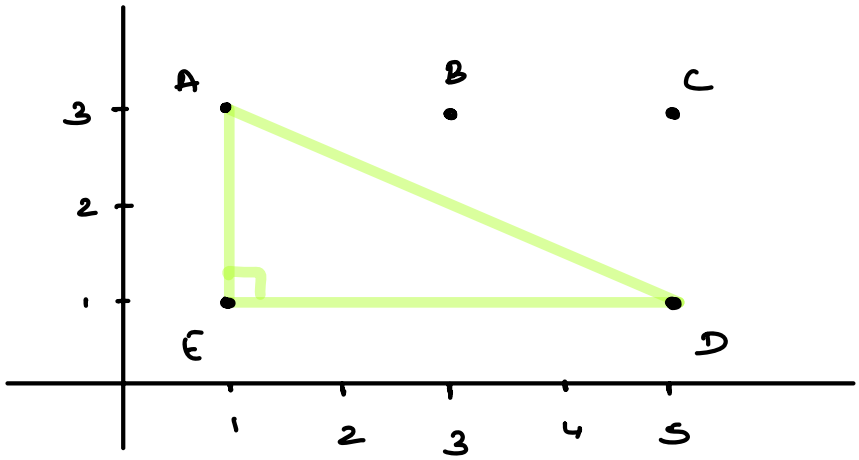
01. Count no. of triangles
    TreeMap / TreeSet
02. Nearest one
03. Smallest window substring

01. Given coordinates of N distinct points on a 2D plane Count the no. of right triangle using the given set of points such that two small sides of a △ should be parallel to x-axis & y-axis

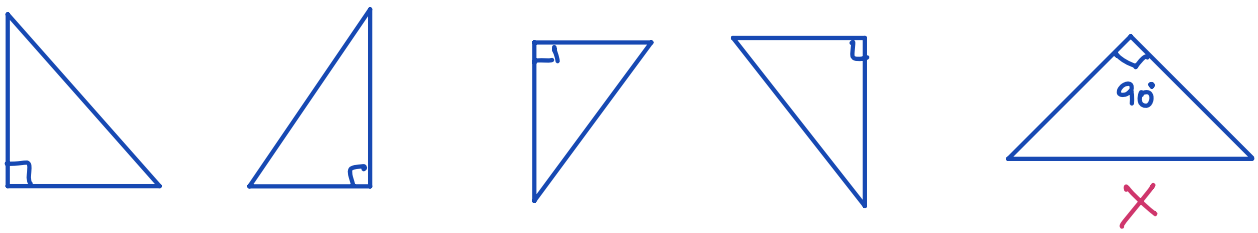
$A = [1 \quad 3 \quad 5 \quad 5 \quad 1]$

$B = [3 \quad 3 \quad 3 \quad 1 \quad 1]$

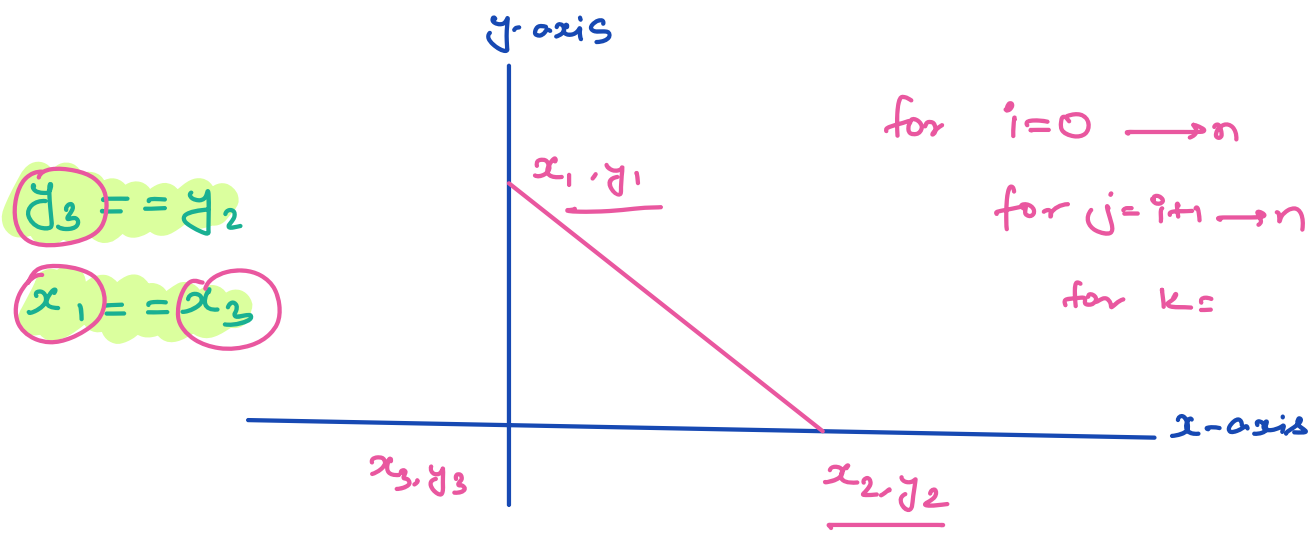
Ans = 6



ABE
ACE
BCD
ACD
CDE
AED



X



$y_3 == y_2$

$x_1 == x_3$

for $i=0 \rightarrow n$

for $j=i+1 \rightarrow n$

for $k=$

Brute force → Take all triplets & check if they are forming a Δ where smaller sides are // to x-axis & // to y-axis

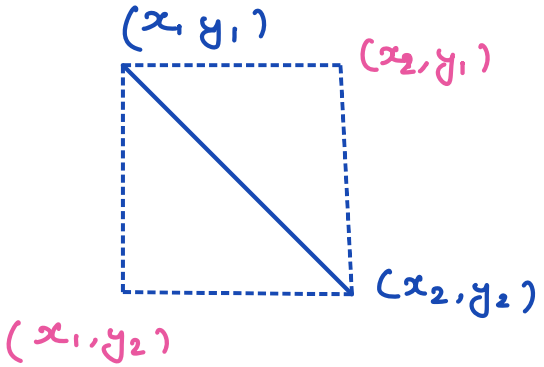
```
for (i=0 ⟶ N-1)
    for (j=i+1 ⟶ N-1)
        for (k=0 ⟶ n-1)
            if (k!=i && k!=j){
                if (A[i]==A[k] && B[k]==B[j]) ||
                 A[j]==A[k] && B[k]==B[i]){
                    count = count + 1;
                }
            }
```

TC: $O(n^3)$

SC: $O(1)$

Optimise TC: $O(n^2)$

Idea 2



i j

↓ ↓

A = { 1 2 3 4 }

B = { 4 3 2 1 }

$x_1, y_1 = A[i], B[i]$

$x_2, y_2 = A[j], B[j]$

Find Idea → Consider all the pairs

→ Check for (x_1, y_2) & Check for (x_2, y_1) } Look for these coordinates in hashset

```
HashSet<String> hs = new HashSet<>();
// Insert all the points in the hs as string

for(i=0; i<n; i++){
    String xy = A[i] + @ + B[i]
    hs.add(xy);
}
```

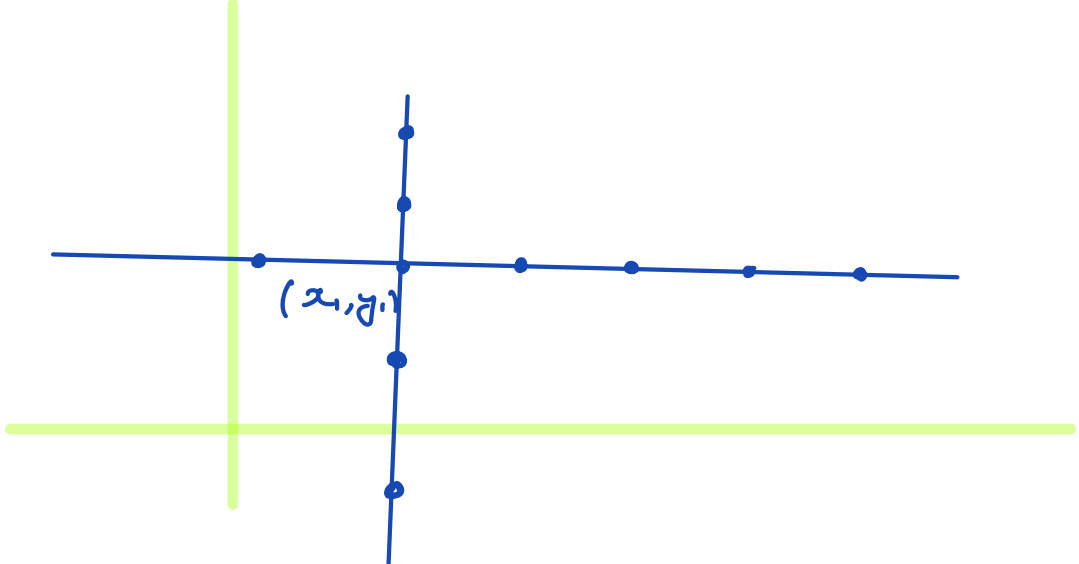
```
for ( i=0; i<n; i++) {
    for ( j=i+1; j<n; j++) {
        if (x1==x2 || y1==y2) continue; // lying on the straight line
        String P1 = x1 + @ + y2
        String P2 = x2 + @ + y1
        if (hs.contains(P1)) count++;
        if (hs.contains(P2)) count++;
    }
}
```

TC : $O(n^2)$

SC : $O(n)$

Optimise again

TC : $O(n)$



Keeping this x_1, y_1 as Right angle, how many triangles can we form

5 point same as x_1

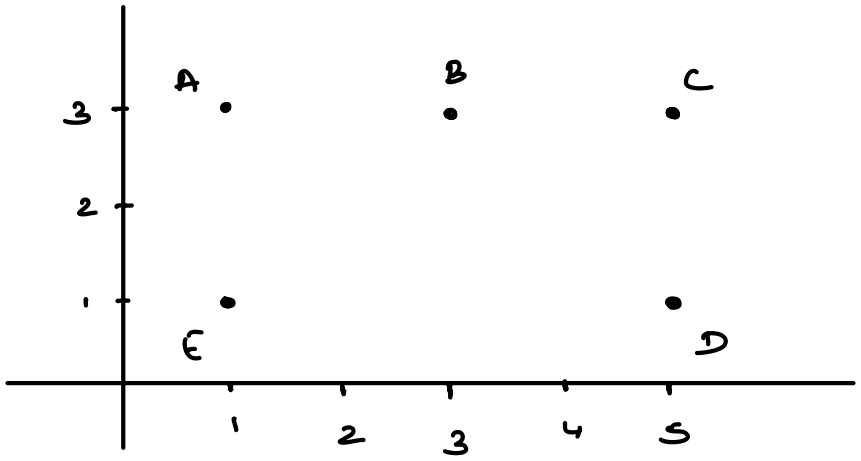
6 point same as y_1

$4 * 5 = \underline{20}$ right angled triangle

* Final Idea

→ For a particular point (x_1, y_1)

{ Count the no. of points same as x_1 − 1 } * { Count the no. of points same as y_1 − 1 }



A (1,3) = (2-1) * (3-1) = 2

B (3,3) = (1-1) * (3-1) = 0

C (5,3) = (2-1) * (3-1) = 2

D (5,1) = (2-1) * (2-1) = 1

E (1,1) = (2-1) * (2-1) = 1

Ans = 6

Find Idea → Create two hashmap

HM1 & HM2

HM1 → frequency of all distinct x

HM2 → frequency of all distinct y

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

```
int c1 = hm1.get(A[i]) - 1;
int c2 = hm2.get(B[i]) - 1;

ans += c1 * c2
```

}

TC: O(n)
SC: O(n)

{1, 3, 7, 10}

TreeMap

floor(4) → 3
ceil(4) → 7
} O(log n)

01. In treemap, data is sorted in increasing order of keys
02. Each insertion / deletion takes O(log N) time, where N is no. of keys
03. It is implemented using Balanced Binary Search Tree

Treeset

01. In treeset, data is sorted in ascending order of keys.
02. Each insertion / deletion takes O(log N) time, where N is no. of keys
03. It is implemented using Balanced BST.

Q You are given an array having all ele = 0 & Q queries

2 Type of queries

Type 1 :- Flip data at i^{th} index

Type 2 :- Get nearest idx from i which has value 1

→ if (ar[i] == 1) print i

→ if 2 indices are nearest, print min idx

→ if no index exist with val 1, print -1

Eg: ar[10] = { 0 0 0 0 0 0 0 0 0 0 }

idx: 0 1 2 3 4 5 6 7 8 9

1 1 1 0

Queries

Type	Index	
1	2	
1	8	
1	7	
2	4	→ ans = 2
1	8	
2	9	→ ans = 7
1	6	
2	6	→ ans = 6
2	4	→ ans = 2

Idea 1

Type 1 → Flip the data at i idx

ar[i] = 1 − ar[i]

Type 2 →

if (ar[i] == 1) print i

else {

l ← i (go left)

ld = i − l

i → r

rd = r − i

if (ld ≤ rd) print l

else print r;

TC : O(Q*n)

SC : O(1)

3

* Idea 2 → Optimise using Treeset

Eg: ar[10] = { 0 0 ~~0~~ 0 0 0 0 ~~0~~ ~~0~~ 0 }

(0 1 2 3 4 5 6 7 8 9)

(1) (1 + 0)

Queries

Treeset = { 2, 7, ~~8~~ }

	Type	Index			
✓	(1)	2			
✓	1	8			
✓	1	7			
→	2	4	→ floor(4) = 2	ld = (2)	return floor(4)
→	1	8	ceil(4) = 7	rd = 3	
→	2	(9)	→ floor(9) = 7	ld = 9 - 7 = (2)	return floor(9)
			ceil(9) = ∞	rd = ∞ - 9 = ∞	

Code → {TODO}

TC : O(Q * log N)

SC : O(n)

10:30 pm → 10:40 pm

```
for ( i=0 ; i<Q ; i++ )
    int type = Q[i][0]
    int idx = Q[i][1]

    if (type == 1) {
        ar[ind] = 1 - ar[ind];
        if (ts.contains(ind)) { ts.remove(ind) }
        else { ts.add(ind) };
    }
    else {
        if (ts.contains(ind)) print(ind);
        else {
            int ld = ∞ , int rd = ∞
            if (ts.floor(ind) != null) {
                ld = ind - ts.floor(ind);
            }
            if (ts.ceil(ind) != null) {
                rd = ts.ceil(ind) - ind;
            }
            if(ld == ∞ && rd == ∞) print(-1);
            if (ld ≤ rd) print ts.floor(ind);
            else print ts.ceil(ind)
        }
    }
}
```

TC : Q * log n

SC : O(n)

Q Given a string A of length N & a string B of length M Find the length of smallest substring in A which contains all characters of B in any order.

```
      0 1 2 3 4 5 6 7 8
A = " a b c d b c x y a
B = " c a c "
```

```
      0 1 2 3 4 5 6 7 8
A = " a b a c b x a b y "
B = " c x "
```

BF - For all the substrings, check if a substring contains all the characters of B or not.

$\frac{n(n+1)}{2}$

TC : $O(n^2 * n) \simeq O(n^3)$

SC = $O(128) \simeq O(1)$ freq array = 256 size

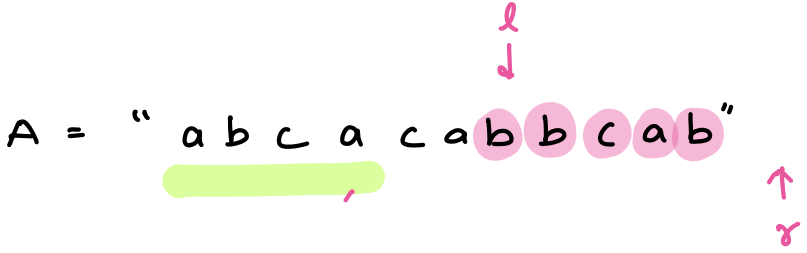
$= O(26) \simeq O(1)$

$= O(256) \simeq O(1)$

* Substring = l r

A = " a b c a c a b b c a b "

B = " a a b " → { a → 2, b → 1 }



a → ~~+2+2+2~~ 1 ans = 4 ✓ ✓

b → ~~x0~~ x2 3 5

c → ~~x2+2~~ 1 5

```
for ( int i=0 ⟶ m-1){
|    Bf[B[i] - 'a']++;
3
```

```
for (int i=0 ⟶ m-1) {
|    AF[A[i] - 'a']++;
3
```

```
if (check(AF, BF)) return m;
```

ans = ∞

l = 0

r = m-1

```
while ( r < n ) {
    if ( check ( AF, BF ) == true ) {
        ans = min ( ans, r - l + 1 );
        AF [ A[l] - 'a' ] --;
        l = l + 1
    } else {
        r = r + 1
        if (r == n) break;
        AF [ A[r] - 'a' ] ++;
    }
}
```

TC : O(n)

SC : O(1)

boolean check ⟶ {TODO}