Todays Content :

- length of longest sequence
- # No: of unique pairs
- # No: of right triangles
- # No: of rectangles : TODO Extension to No: of right Triangle
  - // sides have to parallel to x-axis & y-axis

Note: Insert int in hashset/hashmap → O(1)

- Insert a String as key in hashset/hashmap → O(L)
  - // L is length of String

Q1: Given ar[N] ele, find length of longest seq (subsequence) which can be re-arranged in a strictly increasing by 1.

{Note: Inden elements don't have to be continous}

Eg1: ar[ ] = { -1 8 5 3 10 2 4 9 } ans = 4

(0 1 2 3 4 5 6 7)

Seq1 = { 8 5 10 } = { 5 8 10 } *

Seq2 = { 3 2 4 } = { 2 3 4 } len = 3

Seq3 = { 5 3 2 4 } = { 2 3 4 5 } len = 4

Seq4 = { 8 10 9 } = { 8 9 10 } len = 3

Eg2: ar[] = { 3 8 2 1 9 6 5 6 7 2 } ans = 5

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

Seq1 = { 8 9 6 5 6 7 } ⇒ { 5 6 6 7 8 9 } *

Seq2 = { 8 9 6 5 7 } ⇒ { 5 6 7 8 9 } len = 5

Ideal: Sort & compare adj elements TC: O(N log N + N) // Iterating & comp adj ele

(N log N → sorting)

ar[ ] = { -1 8 5 3 10 2 4 9 }

sort

ar[ ] = { -1 2 3 4 5 8 9 10 } ans = 4

(-1 | 2 3 4 5 | 8 9 10 → 1, 4, 3)

ar[] = { 3 8 2 1 9 6 5 6 7 2 }

sort

ar[] = { 1 2 2 3 5 6 6 7 8 9 } ans = 5

(+1 +1 +0 +1 | +1 +1 +0 +1 +1 +1 → 3, 5)

Edge Case: If data is repeating, we don't reset count

Idea2: Take every element as start of seq & get length of seq

TC: O(N²) O(N)

arr[] → hashset

Ex1: arr[ ] = { -1 8 2 3 7 1 4 9 }

Start:

-1 → 0* : 1

8 → 9 → 10* : 2

2 → 3 → 4 → 5* : 3

3 → 4 → 5* : 2

7 → 8 → 9 → 10* : 3

1 → 2 → 3 → 4 → 5* : 4

4 → 5* : 1

9 → 10* : 1

Ex2: arr[] = { 9 7 6 8 10 }

{8} 9 → 10 → 11* : 2

{6} 7 → 8 → 9 → 10 → 11* : 4

{5}* 6 → 7 → 8 → 9 → 10 → 11* : 5

{7} 8 → 9 → 10 → 11* : 3

{9} 10 → 11* : 1

Opt: Before we start seq at ar[i], check if ar[i]-1 is present in ar[] or not, start seq if and only if ar[i]-1 is not present : TC: O(N) → every element is at max iter 2/3 times

Ex1: ar[ ] = { -1 8 2 3 7 1 4 9 }

befr	start
-2 *	-1 → 0* : 1
7 ✓	8 * skip
1 ✓	2 * skip
2 ✓	3 * skip
6 *	7 → 8 → 9 → 10* : 3
0 *	1 → 2 → 3 → 4 → 5* : 4
3 ✓	4 *
8 ✓	9 *

Edge Case : ar[ ] = { 6 6 6 6 8 9 7 10 }

befr	start
5 *	6 → 7 → 8 → 9 → 10 → 11* : 5
5 *	6 → 7 → 8 → 9 → 10 → 11* : 5
5 *	6 → 7 → 8 → 9 → 10 → 11* : 5
5 *	6 → 7 → 8 → 9 → 10 → 11* : 5

Note: Iterate on hashset to avoid duplicates

```
int lonseq(int ar[N]) {        TC:O(N) SC:O(N)
    Hashset<int> hs // declare hashset in your language of choice
    insert ar[] in hs: → O(N)
    ans = 0
    for(x in hs){ // x is iterating all keys of hashset
        // start seq from x, if x-1 is not present in hs
        if( hs.search(x-1) == false) {
            cnt = 1, y = x+1
            while(hs.search(y) == True){
                y++, cnt++
            }
            ans = max(ans, cnt)
        }
    }
}
```

Idea Binay Search: ✓ → TC: $\log_2^N * [N]$ (// check function)

(// Binay search iterations)

a) Target: Max length Sequence

b) Search Spaa: low = 1, high = N

c) Discard: Eg: ar[10]: low = 1, high = 10

Subseq with len = 5 ✓ — ... 3 4 5 — T T T

Subseq with len = 8 * — 8 9 10 — F F F

Note: In your check func, check if subseq of given len possible or not
↳ ≈ TC:O(N)

2Q) Given N 2d points, calculate no: of distinct points?

{x, y}

Eg1: x[5] = { 2, 1, 3, 2, 2 } ans = 4
y[5] = { 3, 1, 2, 4, 3 }
0 1 2 3 4 *

Idea: Insert all points in hashset

→ a) hashset<int> hs *

→ b) hashset<pair<int, int>> hs ✓ { over-ride hash function Syntax TODO }

→ c) hashset<string> hs:

: Convert each point (x, y) → string

{ x, y → x y *
: (12, 3) → 123 (1, 23) → 123

x, y → x + " " + y // Keep separator between 2 points
: (12, 3) → "12_3" // Then 2 strings are different
: (1, 23) → "1_23" }

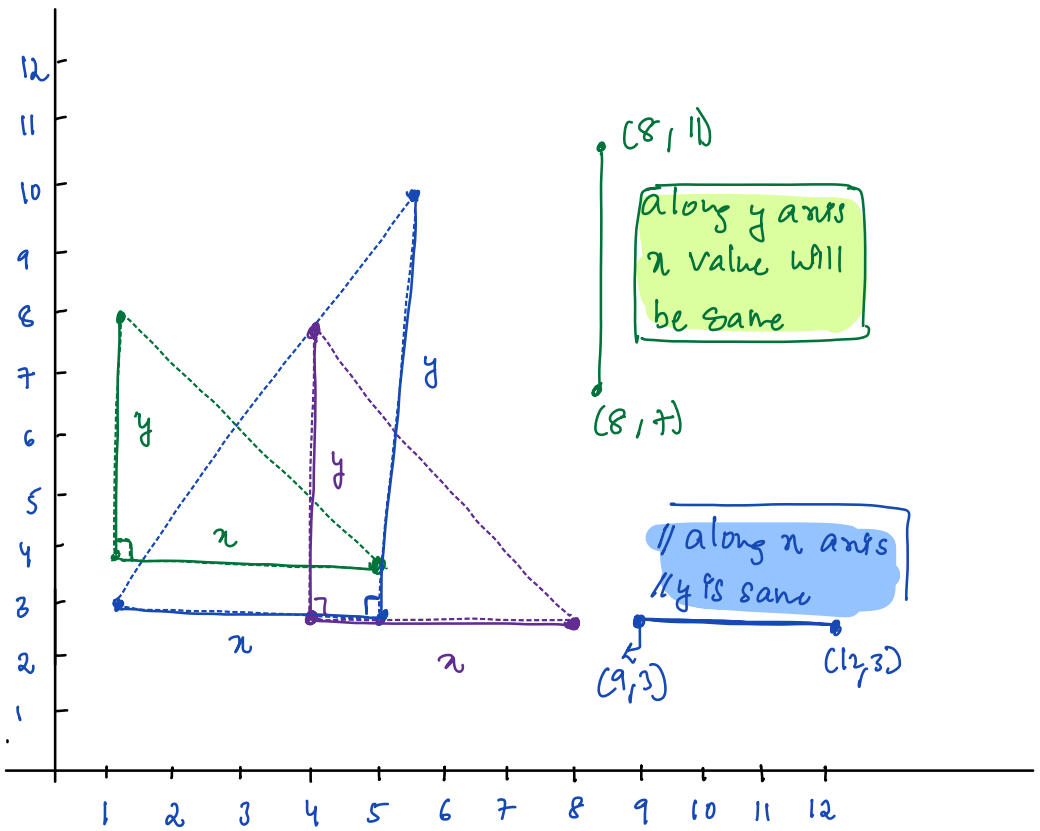
Pseudo code:

```
int points(int x[N], y[N]) {                    TC → O(N)  SC: O(N)
    hashset<string> hs
    x[i] ≤ 10^9 : 6 5 4 4 3
    for(i=0; i<n; i++) {
        string point = to_string(x[i]) + " " + to_string(y[i])
        hs.insert(point)
    }
    return hs.size()
}
```

to_string → Convert number to string

2Q) Given 3 distinct points in a 2D place check if they form triangle, such that shorter sides are parallel to x-axis & y-axis

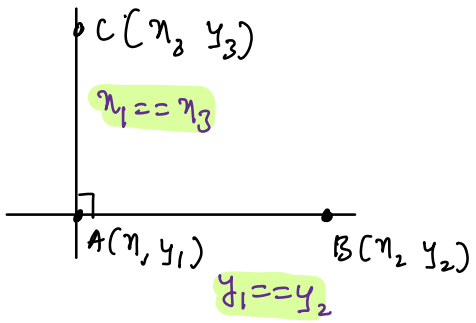
	A	B	C
Ex1:	(1, 8)	(1, 4)	(5, 4)
Ex2:	(5, 10)	(1, 3)	(5, 3)
Ex3:	(4, 3)	(8, 3)	(4, 8)

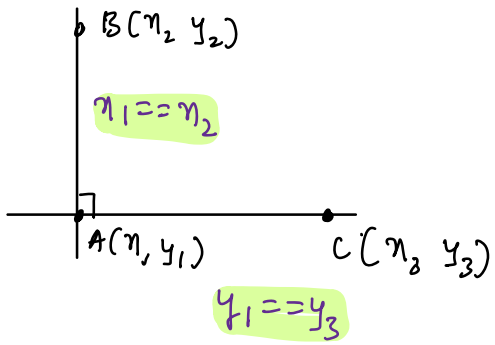


Cases: $A(x_1, y_1)$ $B(x_2, y_2)$ $C(x_3, y_3)$

// Assume A is 90°

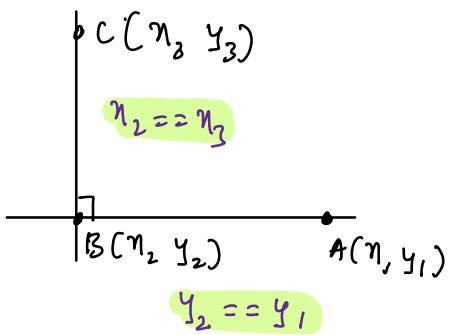
$x_1 == x_3$ && $y_1 == y_2$ or $x_1 == x_2$ && $y_1 == y_3$

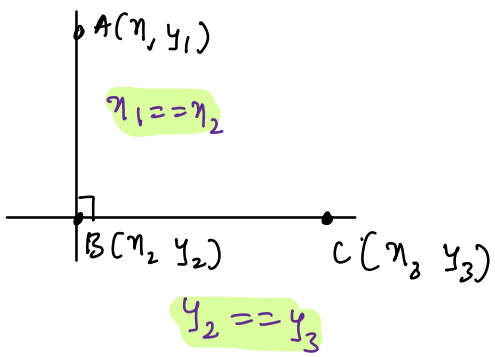




// Assume B is 90°

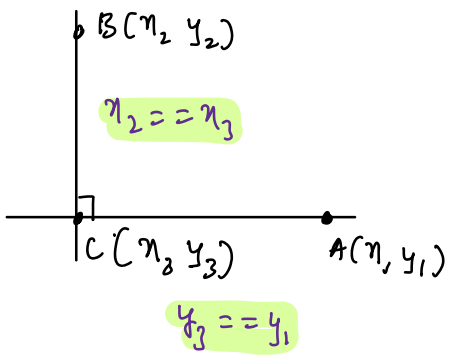
$x_2 == x_3$ && $y_2 == y_1$ or $x_1 == x_2$ && $y_2 == y_3$

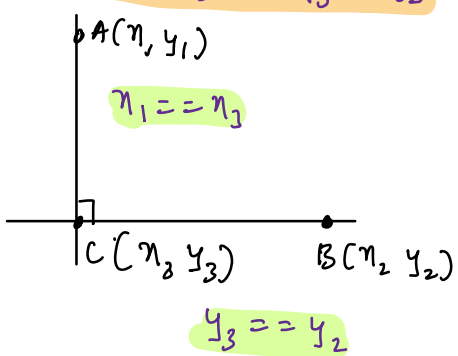




// Assume C is 90°

$x_2 == x_3$ && $y_3 == y_1$ or $x_1 == x_3$ && $y_3 == y_2$



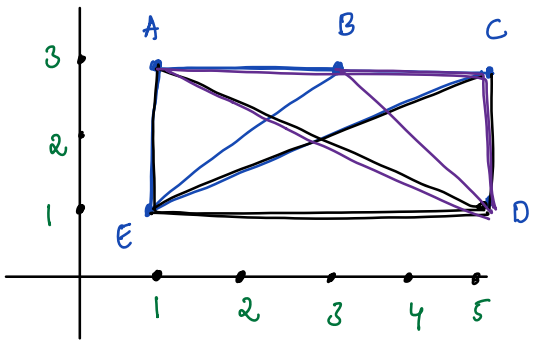


TODO: write all 6 conditions TC: O(1) SC: O(1)

// Given N distinct points, check how many triangle are formed such that, shorter sides are parallel to x-axis & y-axis

Note: 2 arrays x[N], y[N], such that i^{th} = (x[i], y[i])

Ex1:

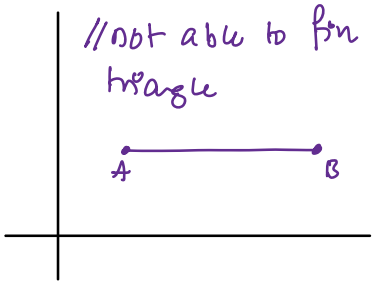


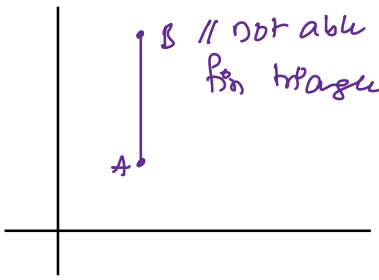
1) B A E ✓
2) C A E ✓
3) A E D ✓
4) C D E ✓
5) B C D ✓
6) A C D ✓

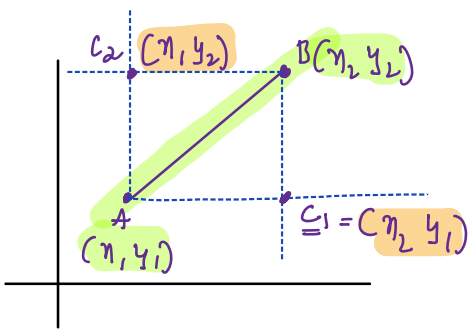
Ideal: For every triplet check, if they from a triangle such that one of them is parallel to x-axis & y-axis

```
i=0; i<N; i++){                    TC: O(N3)  SC: O(1)
    j=i+1; j<n; j++){
        k=j+1; k<n; k++){
            // given 3 points
               check they fom
               required
        }
    }
}
```

Idea2: Need Only 2 points



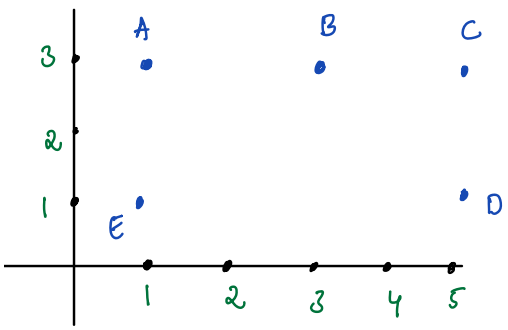




fix 2 opposite points of a hypotenuse & search for C [90°]

pairs:

P1	P2	P3	
A	B *		
A	C *		
A	D	(x1, y2)	(x2, y1)
(1, 3)	(5, 1)	(1, 1) ✓	(5, 3) ✓
x1 y1	x2 y2		
A	E *		
B	C *		
B	D	(x1, y2)	(x2, y1)
(3, 3)	(5, 1)	(3, 1)	(5, 3) ✓
x1 y1	x2 y2		
B	E	(x1, y2)	(x2, y1)
(3, 3)	(1, 1)	(3, 1)	(1, 3) ✓
x1 y1	x2 y2		
C	E	(x1, y2)	(x2, y1)
(5, 3)	(1, 1)	(5, 1) ✓	(1, 3) ✓
x1 y1	x2 y2		



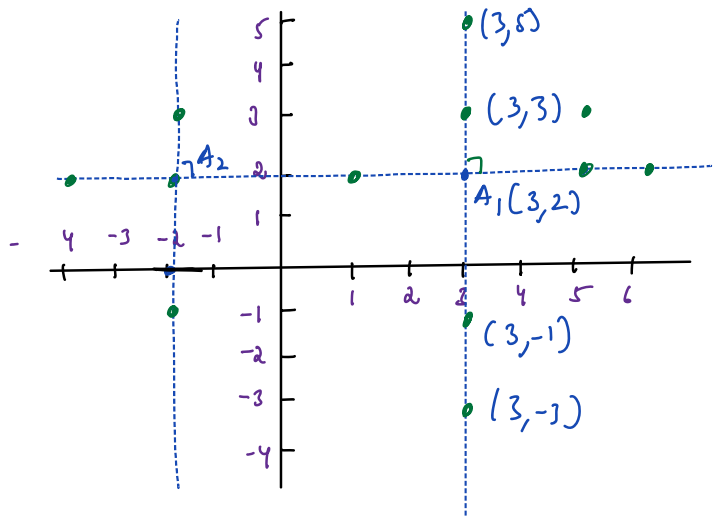
```
int Triangles( int x[], int y[]){   TC: O(N^2)   SC: O(N)

    Hashset<String> hs;
    a) Insert all points in hs : O(N)
    c=0;
    i=0; i<N; i++){                    } ⇒ O(N^2)
        j=i+1; j<N; j++){              }
            // (x1, y1) = {x[i], y[i]}  (x2, y2) = {x[j], y[j]}
            if(y1 != y2 && x1 != x2) // not parallel to x-axis
                                        & y-axis
                // P1 = (x1, y2)  P2 = (x2, y1)
                String P1 = to_string(x1) + " " + to_string(y2)
                String P2 = to_string(x2) + " " + to_string(y1)
                if(hs.search(P1)==True){c=c+1;}
                if(hs.search(P2)==True){c=c+1;}
            }
    }
}
```

Opt:3 Idea: Say we fix 90° point

No: of 90° triangle with A as 90° (x_1, y_i) = [no: of points with $x = x_1$ − 1] * [no: of points with $y = y_i$ − 1]

Idea: Take every point as centre & calculate no: of triangle for that,

→ // To optimise this store as freq of x value & y value




```
int Triangles(int x[], int y[]){   TC: O(N)  SC: O(N)
    HashMap<int, int> fx, fy
    a) // Insert x[] → in hashmap fx : O(N)
    b) // Insert y[] → in hashmap fy : O(N)
    c = 0;
    i = 0; i < N; i++){
        // x1 = x[i]  y1 = y[i]
        // no: of points with x = x1  cx = fx[x1]
        // no: of points with y = y1  cy = fy[y1]
        c = c + [cx - 1] * [cy - 1]
    }
    return c;
}
```