Todays Content:

- No: of distinct points ✓
- No: of rectangles ✓
- Intro to TreeMap ✓
- Nearest 1 ✓

Q: Given N 2d points calculate no of distinct points
{x, y}

↳ every point consider only once

Ex1:

	0	1	2	3	4
x[5] = {	2	3	2	5	3 }
y[5] = {	3	6	3	7	6 }

ans = 3

// i^{th} point: x[i], y[i]

Idea:

a) Insert all points in hashmap < x, y > *

key value

Ex1: { 2, 4 } { 2, 6 } insert in hashmap

hm: { ~~<2, 4>~~ } data is overiden
{ <2, 6> }

b) Store all points in hashset<pair<int, int>> hs

is a key

↳ Note: Override hashfunction / Complicated

Ex1: { 2, 4 } { 2, 6 } { 2, 4 } } insert in hashset

hs: { {2, 4} → It is considered as a key
{2, 6} → It is considered as a key

↳ hs.size() we will get 2

c) Store every point as a string by combining x & y

→ (x, y) = "x" + "y"

We need hashset<string> hs

(12, 3) = 123 (1, 23) = 123

Issue: 2 different are getting same substring

→ (x, y) = "x" + "@" + "y"

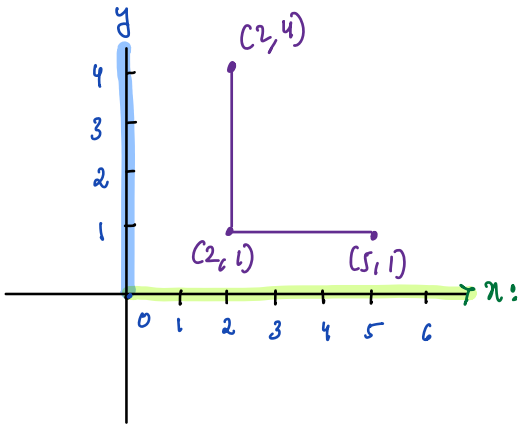
(12, 3) = 12@3 (1, 23) = 123 = 1@23

Above issue is resolved:

```
int 2dpoints(int x[N], int y[N]) { TC: O(N) SC: O(N)
    Hashset<String> hs;
    i=0; i<N; i++) {
        // i^th point = (x[i], y[i])
        String p = to_string(x[i]) + "@" + to_string(y[i])
        hs.insert(p)
    }
    return hs.size()
}
```

// Convert number to string
Please verify its function in your language of choice

// Geometry basics: 2d points & parallel lines along x & y axis



obs:

1) for a line parallel to x-axis y value of point will remain same

2) for a line parallel to y-axis x value of point will remain same

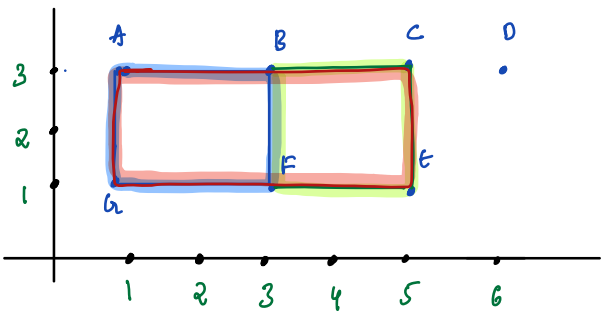
2Q) Given N distinct points, calculate no: of rectangles are formed such that sides are parallel to x-axis & y-axis

Note: We need to have 4 points at 4 corners

Eg: Given 7 points ans = 3

0 1 2 3 4 5 6

{ (1,3) (3,3) (5,3) (6,3) (5,1) (3,1) (1,1) }



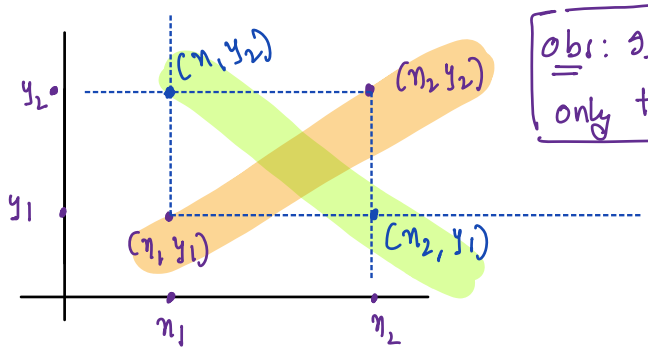
Rectangles:

→ ABGF

→ BCFE

→ ACEG

Idea: To fin a rectangle we need 2 diagnol points



obs: If all 4 points are present only then rectangle formed

↳ For every pair of points check if they form a rectangle

0 1 2 3 4 5 6

{ (1,3) (3,3) (5,3) (6,3) (5,1) (3,1) (1,1) }

$x_1\ y_1$ $x_2\ y_2$ $x_2\ y_1$ $x_1\ y_2$: final count return /2

1 3 3 3 — parallel to x-axis, not diagnol points

1 3 5 3 — parallel to x-axis, not diagnol points

1 3 6 3 — parallel to x-axis, not diagnol points

1 3 5 1 — (5,3) (1,1) rectangle ✓

1 3 3 1 — (3,3) (1,1) rectangle ✓

1 3 1 1 — parallel to y-axis, not diagnol points

3 3 5 3 — parallel to x-axis, not diagnol points

3 3 6 3 — parallel to x-axis, not diagnol points

3 3 5 1 — (5,3) (3,1) rectangle ✓

3 3 3 1 — parallel to y-axis, not diagnol points

3 3 1 1 — (1 3) (3 1) rectangle ✓

2 3 5 3 — parallel to x-axis, not diagnol points

5 3 5 1 — parallel to y-axis, not diagnol points

5 3 3 1 — (3 3) (5,1) rectangle ✓

5 3 1 1 — (1 3) (5,1) rectangle ✓

6 3 5 1 — (5,3) (6,1) → not present rectangle not formed

6 3 3 1 — (3 3) (6,1) → not present rectangle not formed

6 3 1 1 — (1 3) (6,1) → not present rectangle not formed

5 1 3 1 — parallel to x-axis, not diagnol points

5 1 1 1 — parallel to x-axis, not diagnol points

3 1 1 1 — parallel to x-axis, not diagnol points

Pseudocode:

```
int Rectangles(int x[N], int y[N]) {   TC: O(N^2)  SC: O(N)
    HashSet<String> hs; c = 0
    // Store all points in hashset,  { every point is concatenated & stored in hashset }
    for(i = 0; i < N; i++) {
        // i^th point = (x[i], y[i])
        String p = to_string(x[i]) + "@" + to_string(y[i])
        hs.insert(p)
    }
    for(i = 0; i < n; i++) {
        for(j = i+1; j < n; j++) {
            // 2 points (x1, y1) = x[i], y[i]  (x2, y2) = (x[j], y[j])
            if(x1 != x2 && y1 != y2) { // Both are diagnol points
                // We will search for x2 y1 && x1 y2
                String P1 = to_string(x2) + "@" + to_string(y1)
                String P2 = to_string(x1) + "@" + to_string(y2)
                if(hs.search(P1) && hs.search(P2)) {
                    c = c + 1
                }
            }
        }
    }
    return c/2
}
```

Treemap:

Diff between Hashmap/Treemap check this library in your language

: In TreeMap all key value paired are sorted based on inc of keys

Example: Q) Store countries & population

Country	population
India	100
aust	80
ame	120
pak	40
chi	180

Hashmap

key	value
chi	180
pak	40
aust	80
India	100
ame	120

Treemap

key	value
ame	120
aust	80
chi	180
India	100
pak	40

Sorted based on keys

Functions in Treemap

insert(k, v)

Search(k)

access(k)

update(k, v)

delete(k)

Note: A single operation in Treemap / Treeset takes log N time

N: no: of keys we inserted

Functions in Treeset

→ only keys & based on keys data sorted

insert(k)

search(k)

delete(k)

Additional functions: floor/lower_bound ceil/upper_bound

a single operation TC: log N

floor(k): return greatest key <= k

Note: If ceil/floor doesn't exit function return null

ceil(k): return smallest key >= k

Eg: ar[3 2 10 6 8] → Treeset → TS: { 2 3 6 8 10 }

ceil(7): return 8 floor(5) = return 3

ceil(11): return null floor(0) = return null

Iterate on Treeset

rr

ele: 2

last ele: 10

2Q) Given ar[N]=0 & Q queries

2 Type of Queries

Type1 : i : flip data at i^{th} inden : $1 \rightleftharpoons 0$ (ar[i])

Type2 : i : Get nearest inden from i, which has 1
- if ar[i]==1, print i
- if 2 indices are nearest, print min inden
- if no inden exist with val 1, print -1

N=10

	0	1	2	3	4	5	6	7	8	9
ar[10] =	0	0	0	0	0	0	0	0	0	0
			1					1		1

Q = 8

type	inden	
1	2	✓
1	8	✓
1	7	✓
2	4	: return 2
1	8	✓
2	9	: return 7
1	9	: ✓
2	8	✓ return 9

Idea: Given N & Queries . Overall TC: Q*N

↳ Create ar[N]=0 Overall SC: O(N)

Type1 : i : flip data at i^{th} inden TC: O(1)

: ar[i] = 1 - ar[i]

Type2 : i : check if ar[i]==1 TC: O(N)

: Iterate on left get nearest inden in left = l

: Iterate in right get nearest inden in right = r

: return inden which is closest to i

↳ if both are at equal distance return min inden

↳ if l & r doesn't exist return -1

Optimization using Treeset:

Note: Treeset should contain indices with value = 1

N = 10

	0	1	2	3	4	5	6	7	8	9
ar[10] = {	0	0	0	0	0	0	0	0	0	0 }
			1					1		1

Q = 8

type index

1 : 2 ✓

1 : 8 ✓

1 : 7 ✓

2 : 4 → floor : 2, ceil : 7 } print 2

1 : 8 ✓

2 : 9 → floor : 7, ceil : null } print 7

1 : 9 .

2 : 8 → floor : 7, ceil : 9 } Both are at same distance print min index
print 7

Treeset:

{ 2 7 9 }

Note: No need of ar[] just Treeset is Sufficient

Pseudocode:

Idea: Given N & Queries → overall TC: Q * C(logN)

Treeset<int> TS → overall SC: O(N)

Type1: i: search for i in Treeset → TC: O(logN)

if i is present: delete i from Treeset

if i is not present: insert i in Treeset

Type2: i: if i is present in Treeset: print(i) → TC: O(logN)

: Nearest index on left l = TS.floor(i)

: Nearest index on right r = TS.ceil(i)

→ Return index which is nearest

↳ Edge1: If both at equal distance print min index

↳ Edge2: If both l = r = null print(-1)

doesn't exist