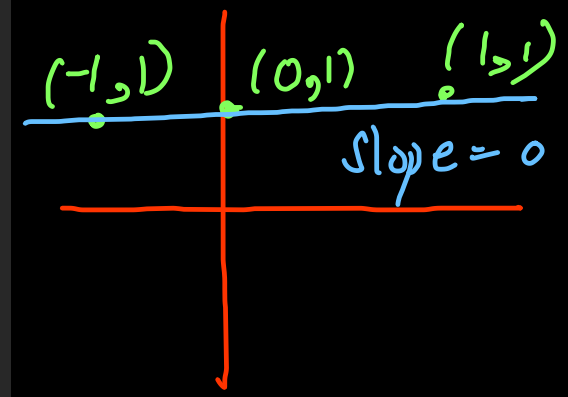
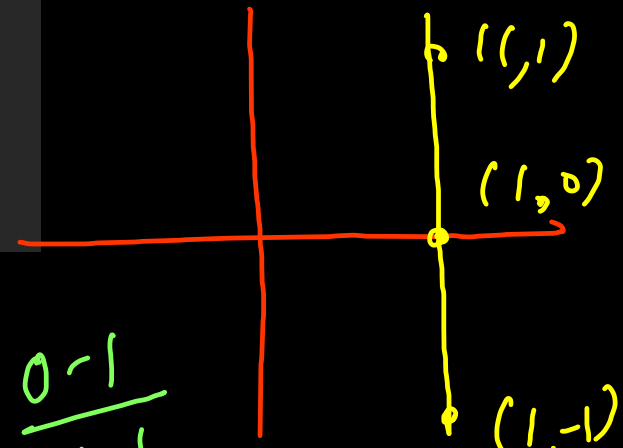
```java
public double getSlope(int[] p1, int[] p2){
    if(p1[0] == p2[0]) return Double.POSITIVE_INFINITY; // parallel to y axis
    return  ((double)p2[1] - p1[1]) / (p2[0] - p1[0]);
}

public boolean checkStraightLine(int[][] pts) {
    double slope = getSlope(pts[0], pts[1]);

    for(int i = 2; i < pts.length; i++){
        double curr = getSlope(pts[i], pts[0]);
        if(curr != slope) return false;
    }

    return true;
}
```

0(1)

$(y_2 - y_1)/(x_2 - x_1)$

O(n)



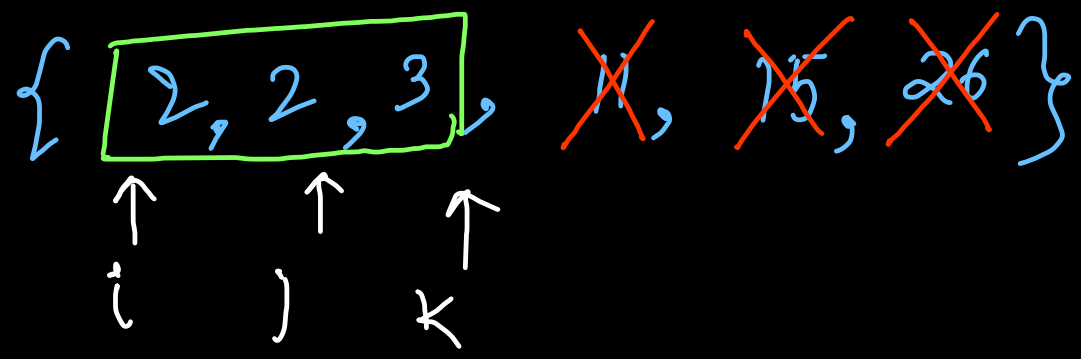


$\text{Slope} = \frac{0-1}{1-1} = \frac{-1}{0} = +\infty$

LC 1037 Valid Boomerang

```java
public double getSlope(int[] p1, int[] p2){
    if(p1[0] == p2[0] && p1[1] == p2[1]) return Double.NEGATIVE_INFINITY;
    if(p1[0] == p2[0]) return Double.POSITIVE_INFINITY; // parallel to y axis
    return  ((double)p2[1] - p1[1]) / (p2[0] - p1[0]);
}

public boolean isBoomerang(int[][] points) {
    double s1 = getSlope(points[0], points[1]);
    double s2 = getSlope(points[1], points[2]);

    if(s1 == Double.NEGATIVE_INFINITY || s2 == Double.NEGATIVE_INFINITY)
        return false;

    if(s1 == s2) return false;
    return true; // distinct and different slope
}
```

Largest Perimeter → Triangle



$11+15 \not> 26$

$3+11 \not> 15$

$2+3 \not> 11$

$2+2 > 3$

Triangle can be used using (a, b, c) sides

- $a+b > c$
- $a+c > b$
- $b+c > a$

→ $a+b > c$ if c is max^m size

```java
public int largestPerimeter(int[] nums) {
    Arrays.sort(nums);
    int n = nums.length;
    int i = n - 3, j = n - 2, k = n - 1;

    while(i >= 0){
        if(nums[i] + nums[j] > nums[k])
            return nums[i] + nums[j] + nums[k];
        i--; j--; k--;
    }

    return 0;
}
```

nlogn

valid triangle

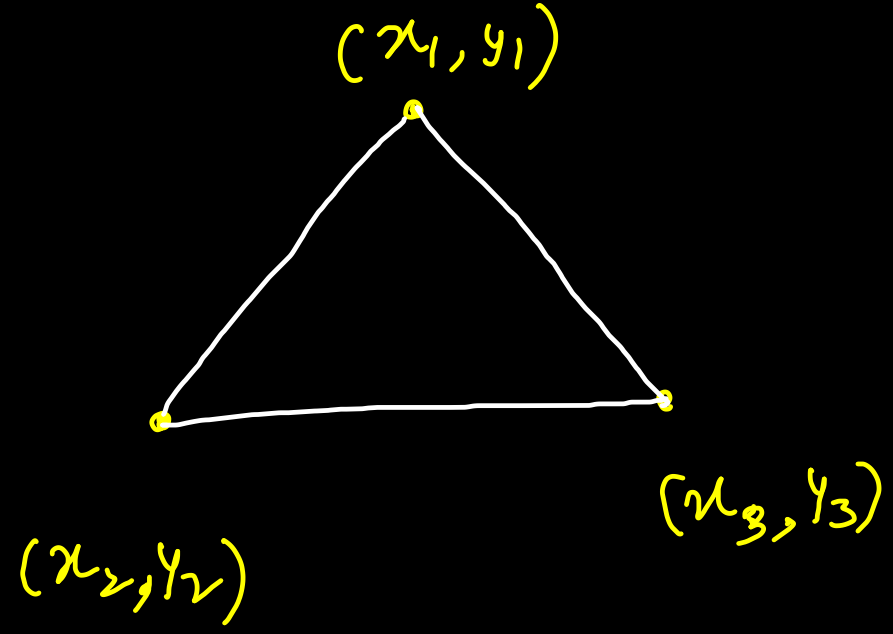
perimeter

LC 976) Largest Perimeter Triangle.

Three Pointer Technique

O(n)

Area of Triangle



$$= \frac{1}{2}\left(x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) \right)$$

```java
public double getArea(int[] p1, int[] p2, int[] p3){
    return Math.abs(0.5 * (p1[0] * (p2[1] - p3[1])
        + p2[0] * (p3[1] - p1[1]) + p3[0] * (p1[1] - p2[1])));
}
public double largestTriangleArea(int[][] points) {
    double max = 0;

    for(int[] p1: points)
    for(int[] p2: points)
    for(int[] p3: points)
        max = Math.max(max, getArea(p1, p2, p3));

    return max;
}
```

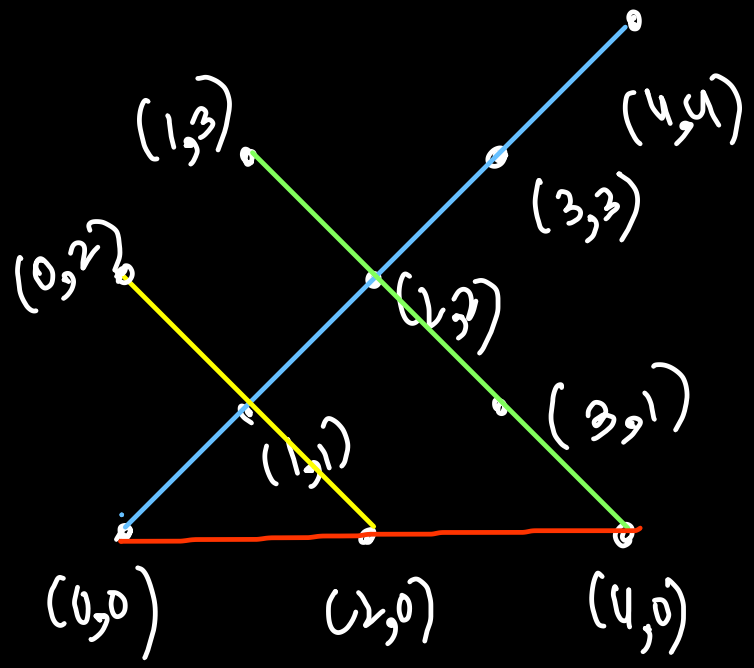
$n = 50$ max

$O(n^3)$

Maxm Points on a line

{Google Interview}

Maximum number of collinear pts



Brute force

Hashmap<slope, count>

slope → double

count → integer

+∞ : 2

-1 : 2+1

0 : 2

basept as (0,2)

$O(n^2)$

max = ~~3~~ 5

+∞ : 2

+1 : 2+1+1+1

0 : 2+1

basept : (0,0)

```java
public double getSlope(int[] p1, int[] p2) {
    if (p1[0] == p2[0] && p1[1] == p2[1])
        return Double.NEGATIVE_INFINITY;
    if (p1[0] == p2[0])
        return Double.POSITIVE_INFINITY; // parallel to y axis
    return ((double) p2[1] - p1[1]) / (p2[0] - p1[0]);
}

public int maxPoints(int[][] points) {
    if(points.length <= 2) return points.length;
    int collinear = 2;

    for(int[] p1: points){
        HashMap<Double, Integer> lines = new HashMap<>();
        // Double -> Slope, Integer -> Count of Points

        for(int[] p2: points){
            double slope = getSlope(p1, p2);
            lines.put(slope, lines.getOrDefault(slope, 1) + 1);
            collinear = Math.max(collinear, lines.get(slope));
        }
    }

    return collinear;
}
```

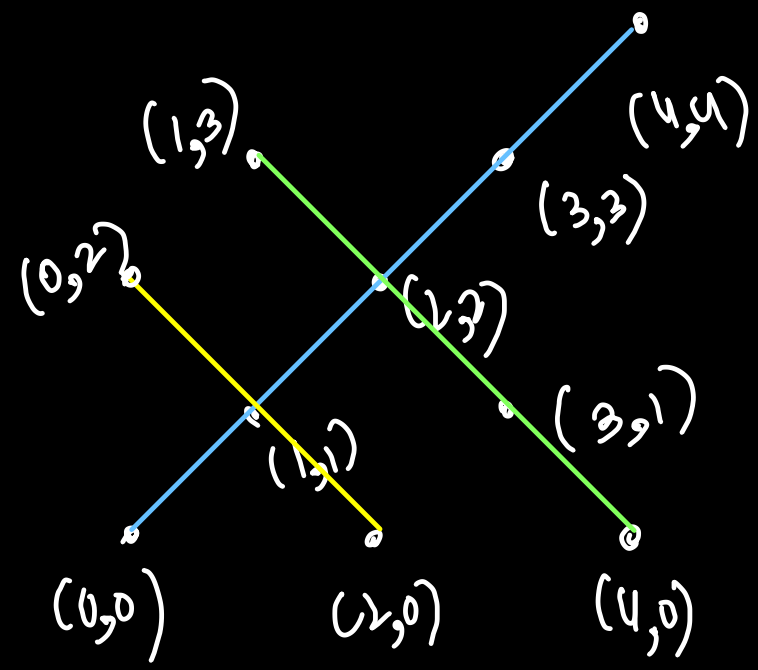
→ new for every basept

O(n²)

basept existed already

-1

collinear = ~~2~~ ~~3~~ ~~4~~ 5



basept (0,2)

+1 : 2

0 : 2

-1 : 2+1
yellow

basept (1,3)

+1 : 2

-1 : 2+1+1
green

basept (0,0)

+1 : 2
+1
+1
+1
blue

LC 492

```
public int[] constructRectangle(int area) {
    int breadth = (int)Math.sqrt(area);
    while(area % breadth != 0) breadth--;
    int length = area / breadth;
    return new int[]{length, breadth};
}
```

$O(\log N)$

$O(\sqrt{N})$ worst (prime no)

$n = 37$

breadth $= \sqrt{37} = 6$

6 5 4 3 2 1

$n = 50$

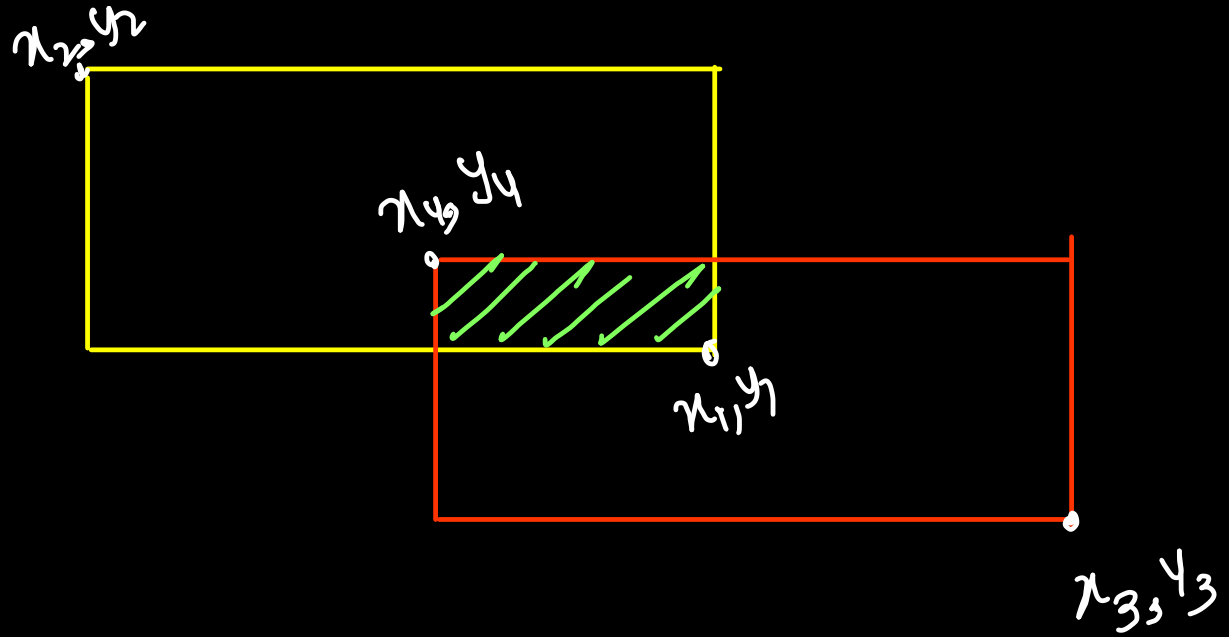
breadth $= \sqrt{50} = 7 \Rightarrow 50 \% 7 \neq 0$

$6 \Rightarrow 50 \% 6 \neq 0$

$5 \Rightarrow 50 \% 5 == 0 \Rightarrow$ length $= 50/5 = 10$

{10, 5} L >= W

Are rectangles overlap



$x_1 > x_4$

&& $y_4 > y_1$

&& $y_2 > y_3$

&& $x_3 > x_2$

```java
public boolean isRectangleOverlap(int[] rec1, int[] rec2) {
    return (rec1[0] < rec2[2]
        && rec1[1] < rec2[3]
        && rec1[2] > rec2[0]
        && rec1[3] > rec2[1]);
}
```

→ O(1) time