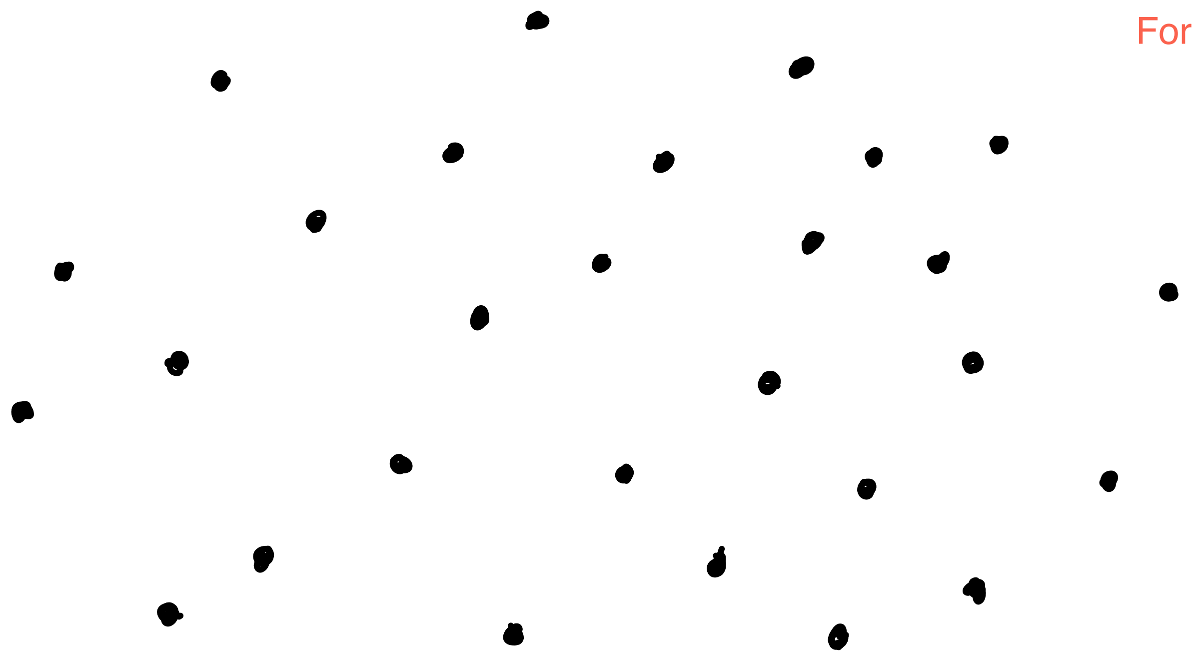
# "ULTIMATE PLANAR C.H. ALGORITHM ?"

KIRKPATRICK-SEIDEL

It's a divide & conquer algorithm

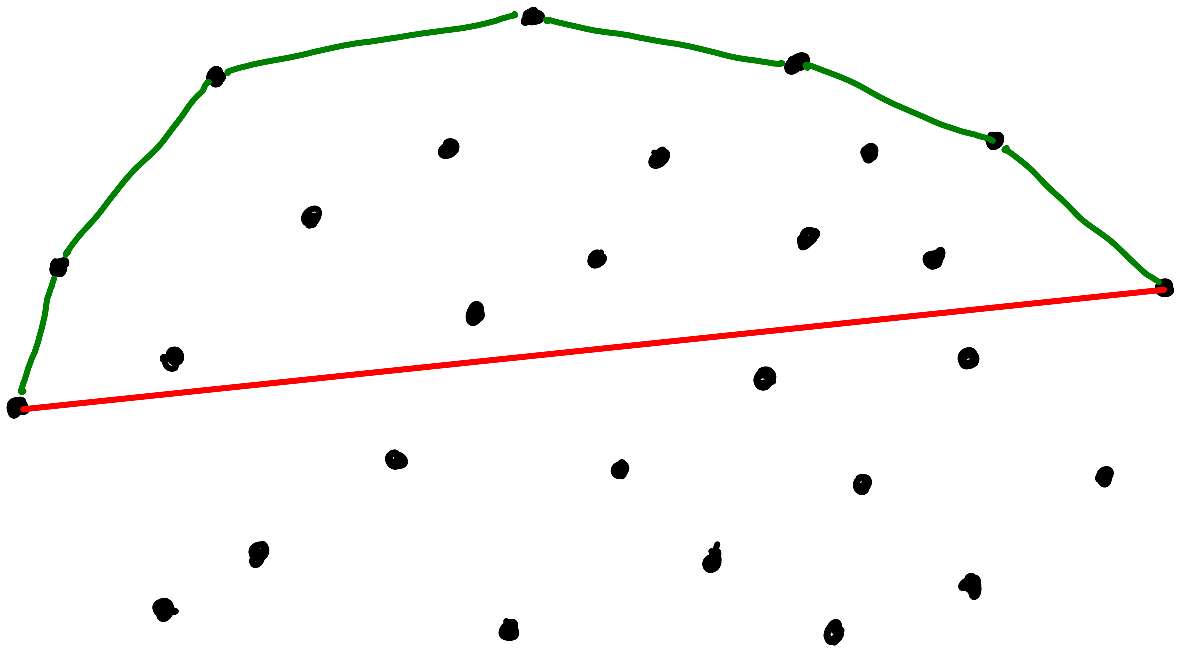
For comp260, this is just for context



# "ULTIMATE PLANAR C.H. ALGORITHM ?"

KIRKPATRICK-SEIDEL

It's a divide & conquer algorithm



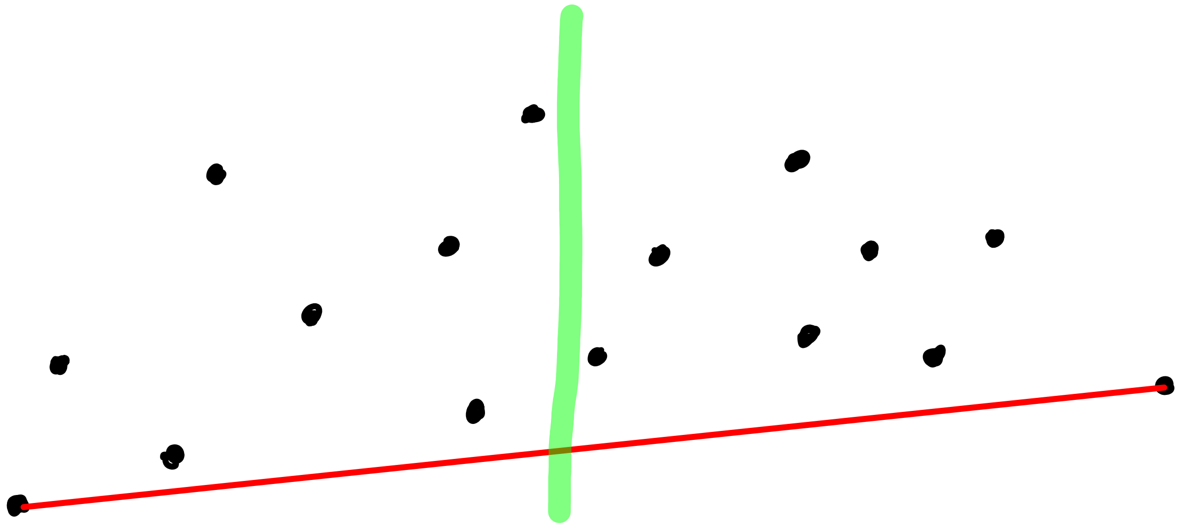
Upper hull only

# "ULTIMATE PLANAR C.H. ALGORITHM ?"

## KIRKPATRICK-SEIDEL

It's a divide & conquer algorithm

divide-conquer-merge



Upper hull only

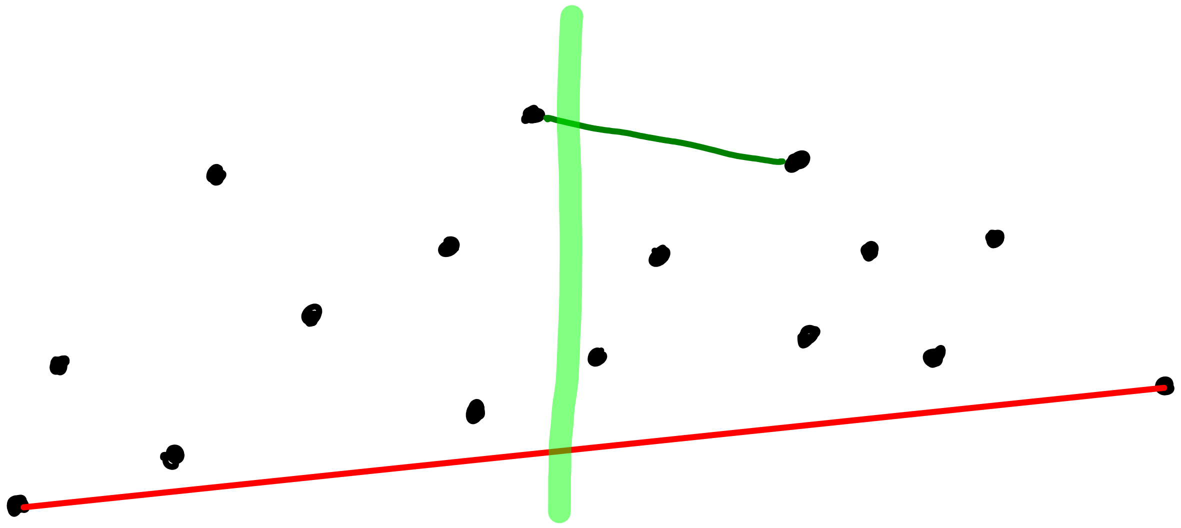
# "ULTIMATE PLANAR C.H. ALGORITHM ?"

## KIRKPATRICK-SEIDEL

It's a divide & conquer algorithm

~~divide-conquer-merge~~

divide-merge-conquer !



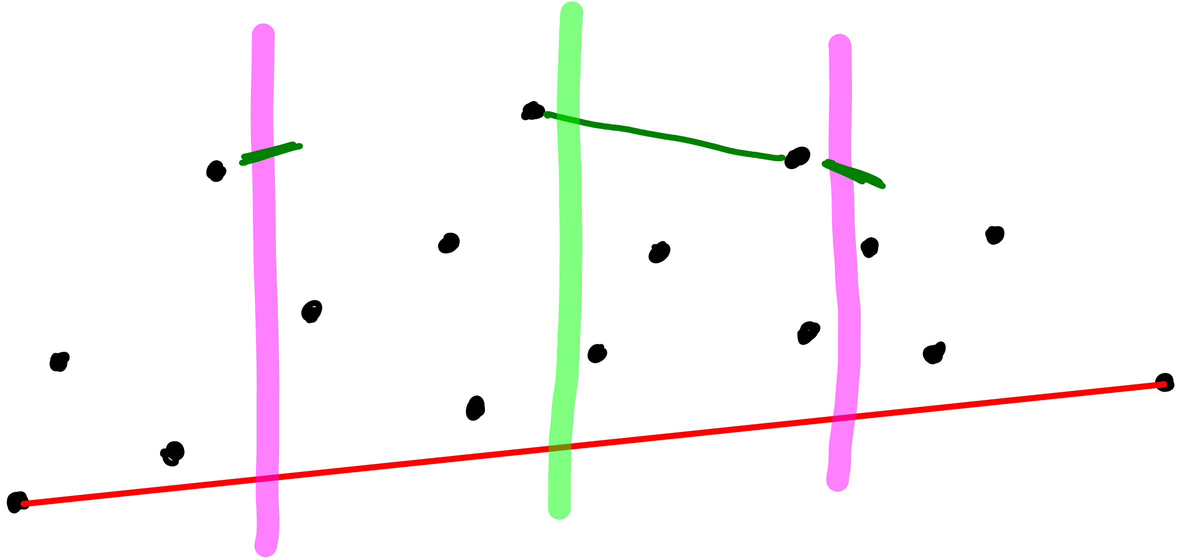
Upper hull only

# "ULTIMATE PLANAR C.H. ALGORITHM?"

KIRKPATRICK-SEIDEL

It's a divide & conquer algorithm

divide-merge-conquer

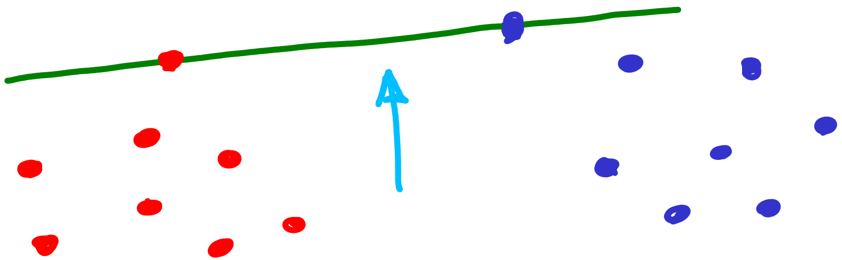


Upper hull only

comp260:
start here.

# FINDING A BRIDGE in LINEAR TIME

Let the bridge have slope $K^*$



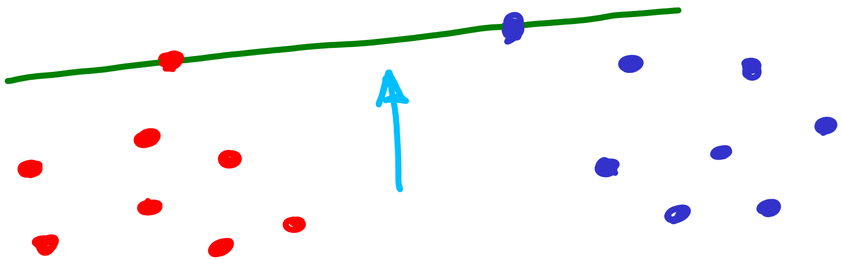
comp260: these points are colored depending on what side of the ray they are on.
We just care that there is some ray.
In the context of convex hulls, the ray is the median of the point set, in a particular direction that depends on what has happened before in the convex hull algorithm that uses this bridge-finding procedure.

# FINDING A BRIDGE in LINEAR TIME

Let the bridge have slope $K^*$

Suppose we <u>guess</u> slope K.

Sweep K



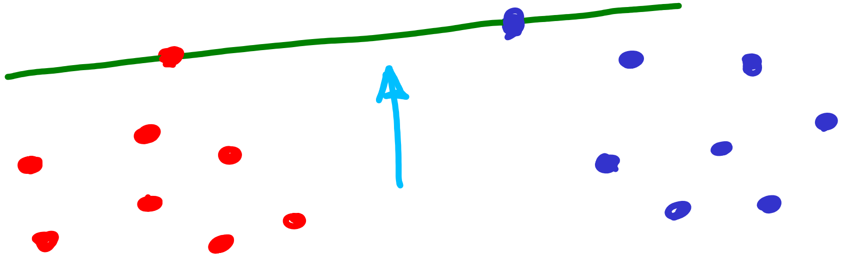
Guess $K = K^*$
↳ confirm bridge

# FINDING A BRIDGE in LINEAR TIME

Let the bridge have slope $K^*$

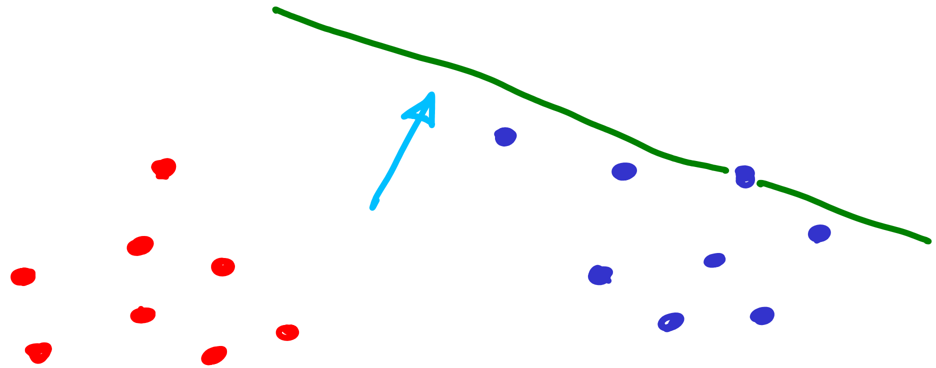
Suppose we <u>guess</u> slope $K$.

Sweep $K$



Guess $K = K^*$
↳ confirm bridge

Guess $K < K^*$
↳ sweep stops on blue

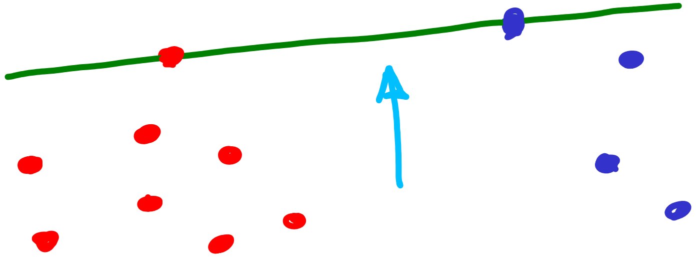


# FINDING A BRIDGE in LINEAR TIME

Let the bridge have slope $K^*$

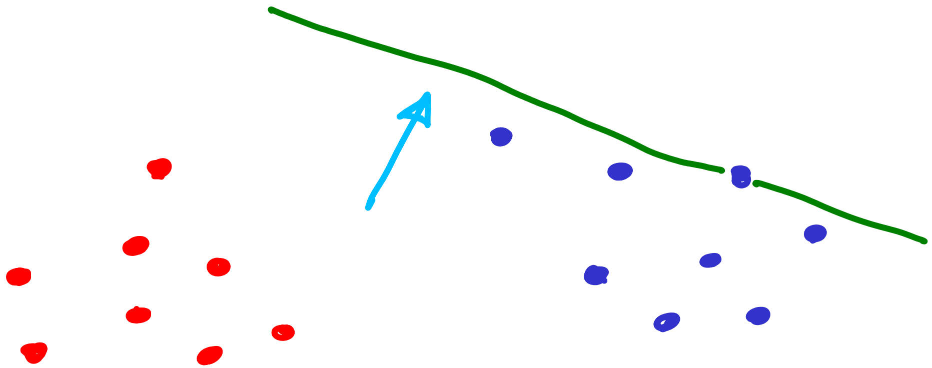
Suppose we <u>guess</u> slope $K$.

Sweep $K$

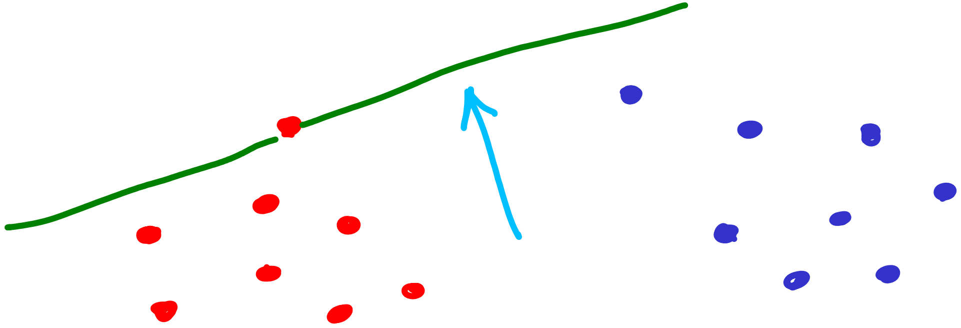


Guess $K = K^*$
↳ confirm bridge

Guess $K < K^*$
↳ sweep stops on blue



Guess $K > K^*$
↳ sweep stops on red

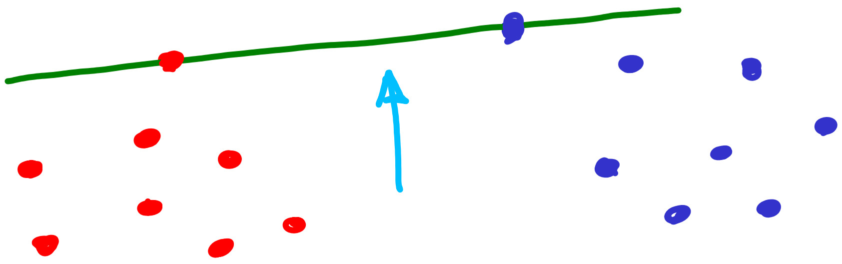


# FINDING A BRIDGE in LINEAR TIME

Let the bridge have slope $K^*$

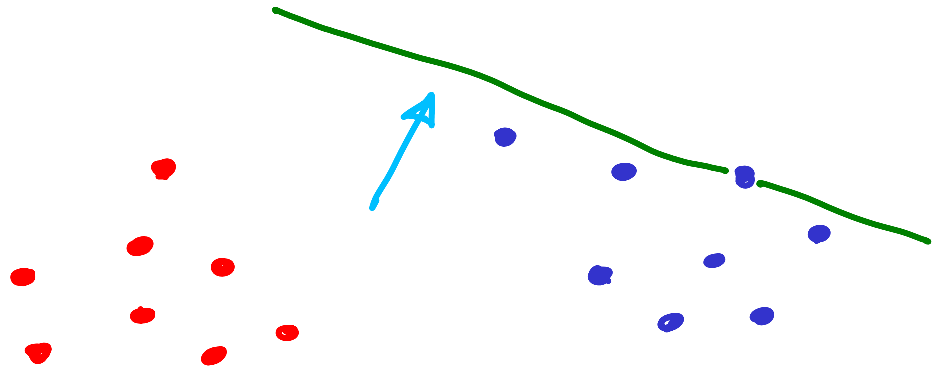
Suppose we **guess** slope $K$.

Sweep $K$

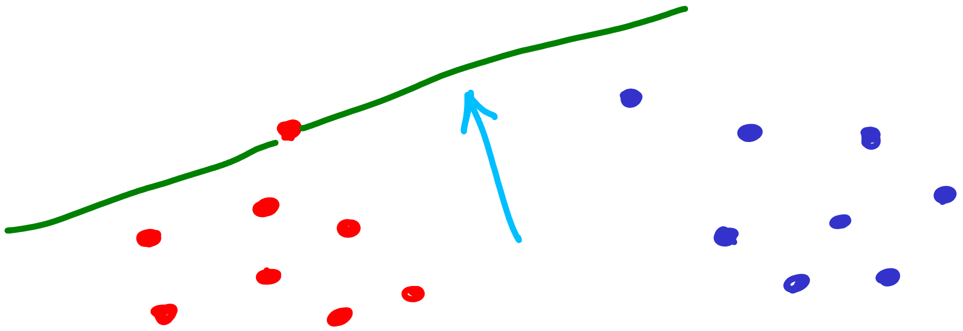


Guess $K = K^*$
↳ confirm bridge

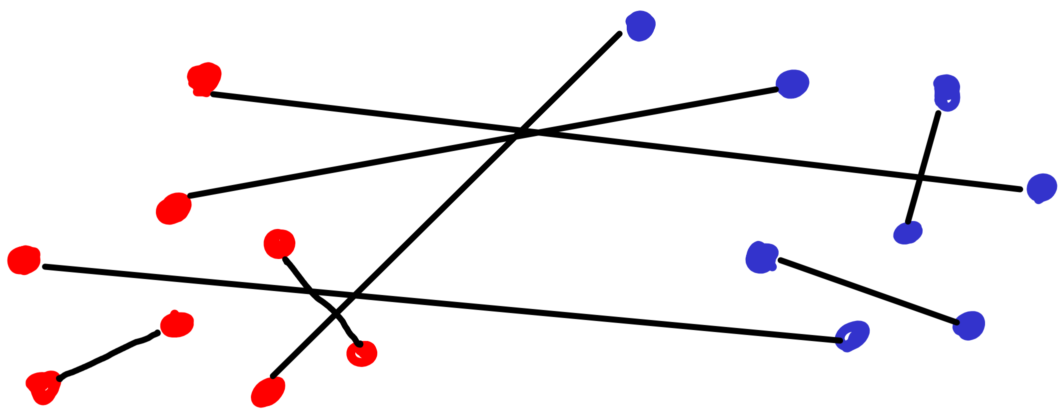
Guess $K < K^*$
↳ sweep stops on blue



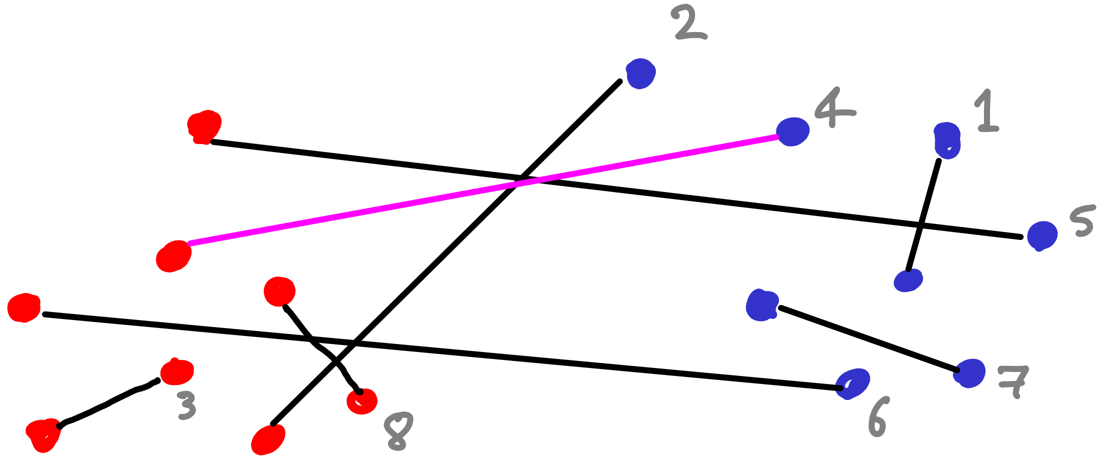
Guess $K > K^*$
↳ sweep stops on red



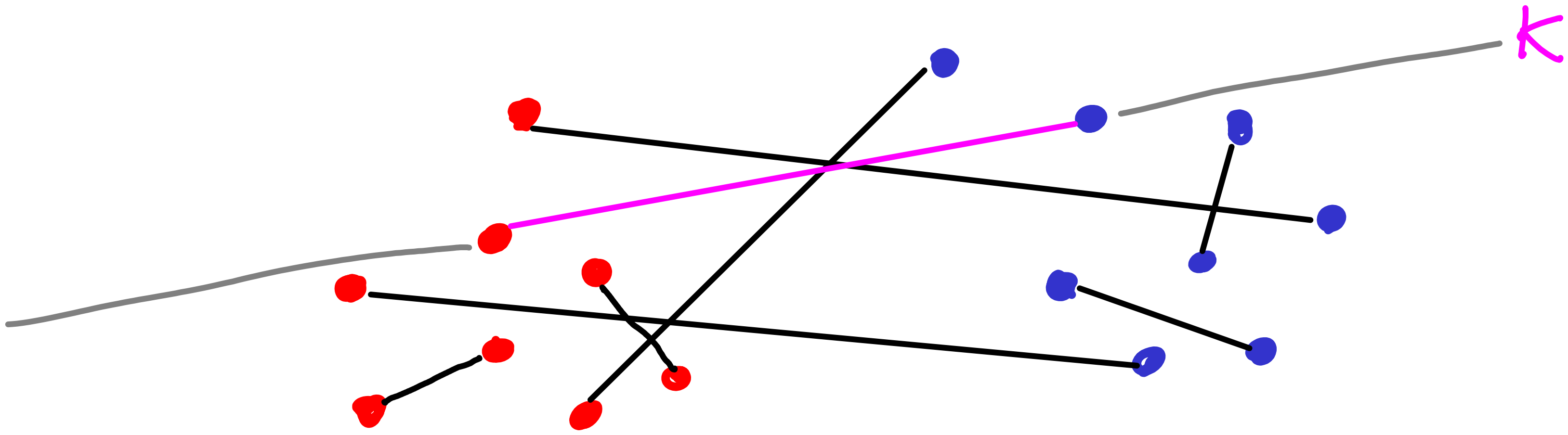
$O(n)$ time
to guess & verify



- Arbitrarily pair up points

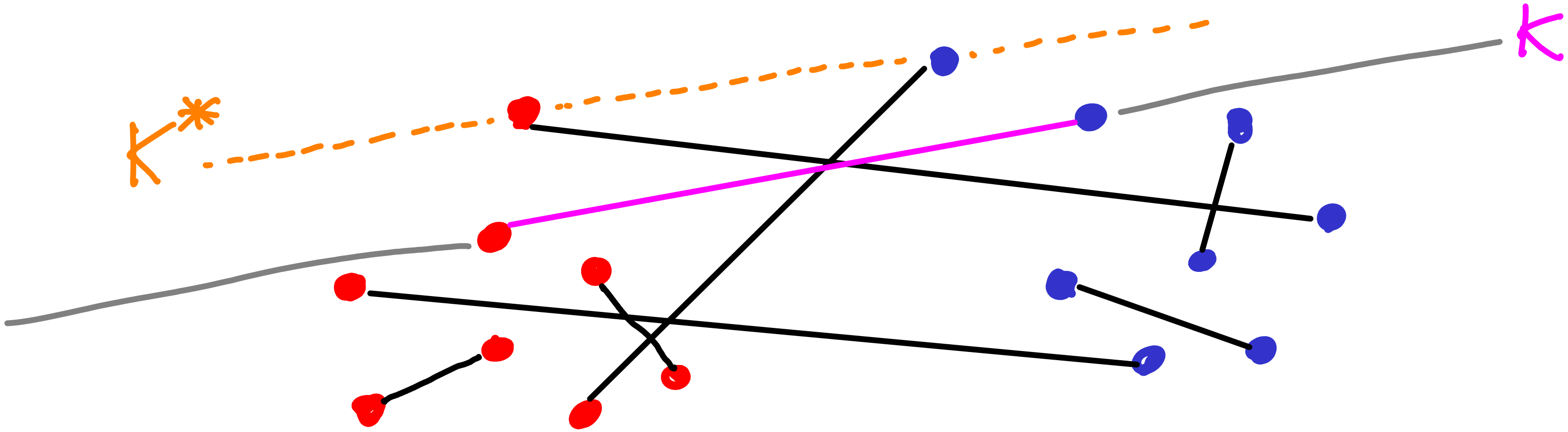



- Arbitrarily pair up points
- Find median slope



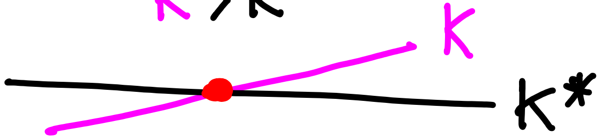

- Arbitrarily pair up points
- Find median slope

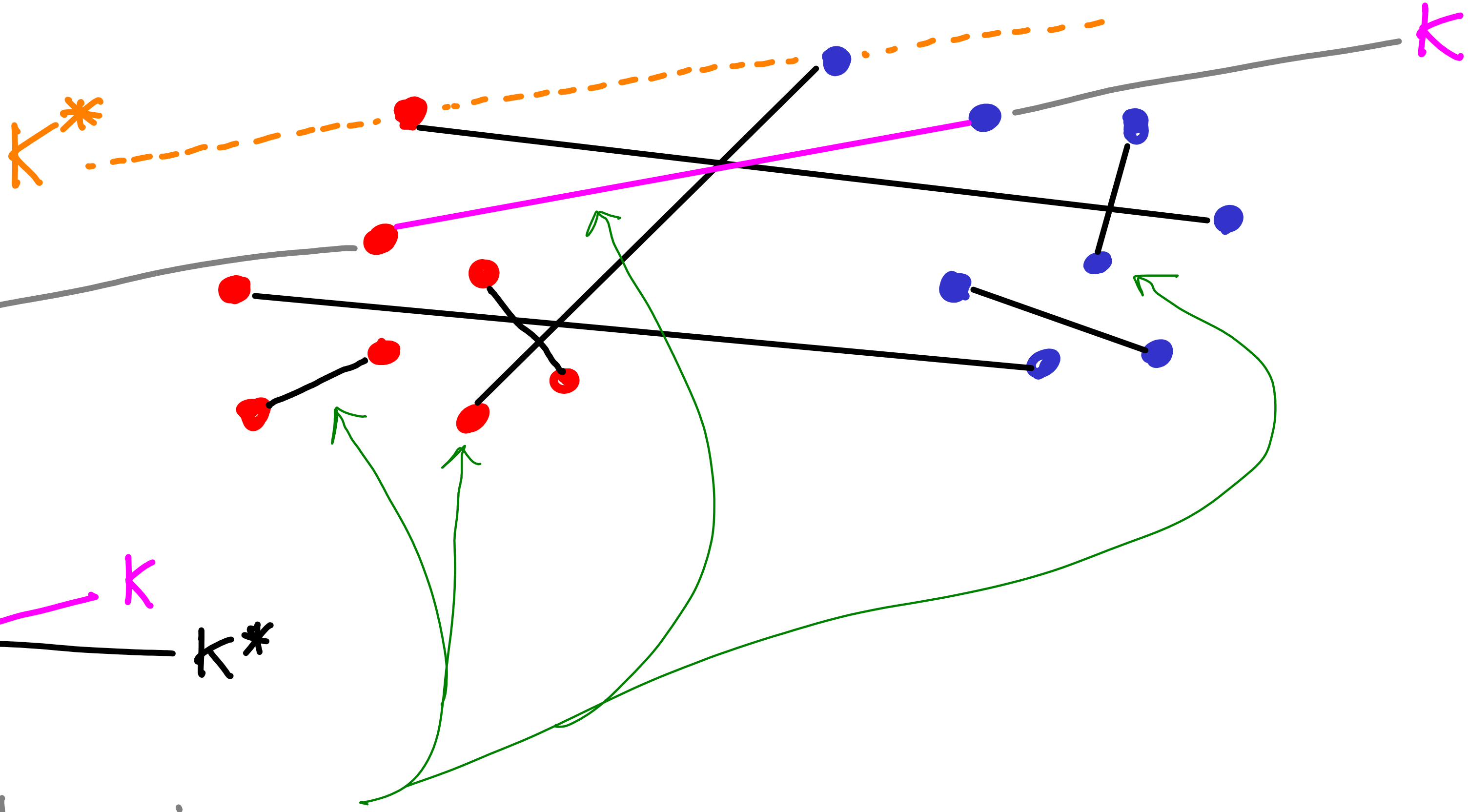

- Guess K = median



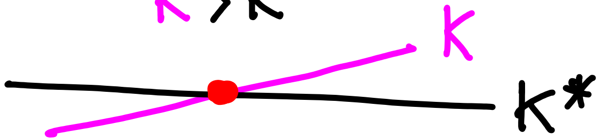

Case 1

$K > K^*$



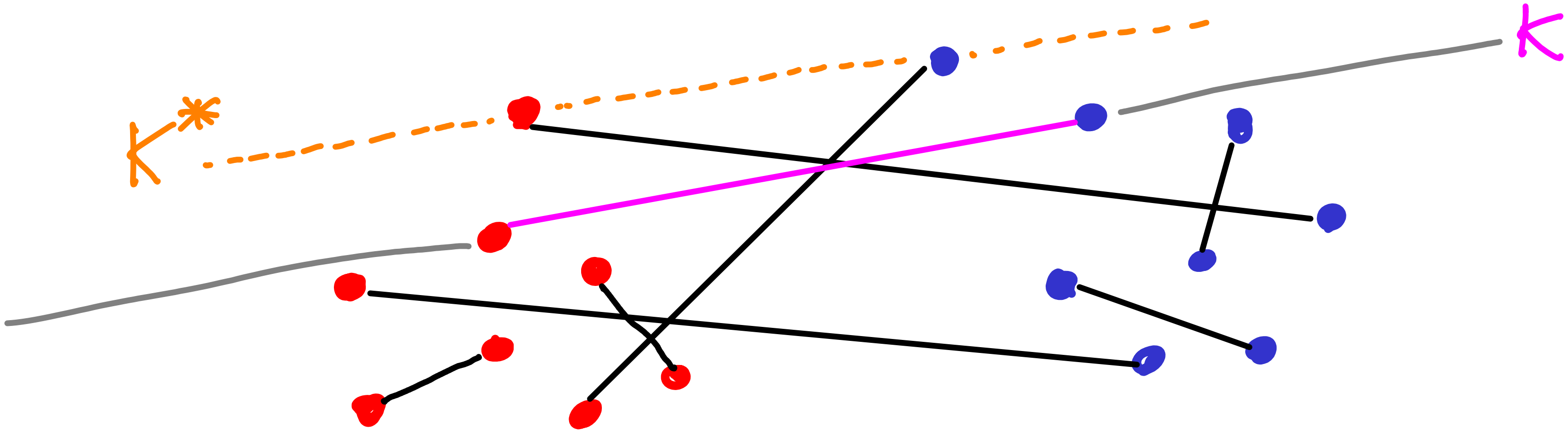



Case 1 $K > K^*$

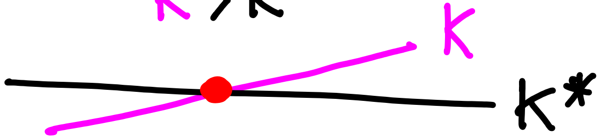



Half of the pairs have slope $K' \geqslant K$, so

$K' > K^*$

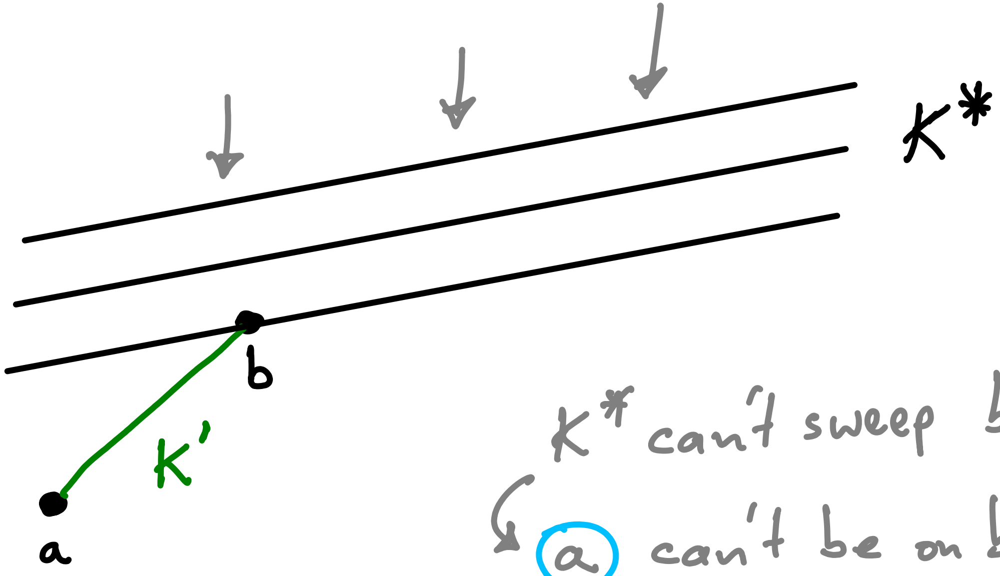



Case 1 $K > K^*$




Half of the pairs have slope $K' > K$, so

$K' > K^*$

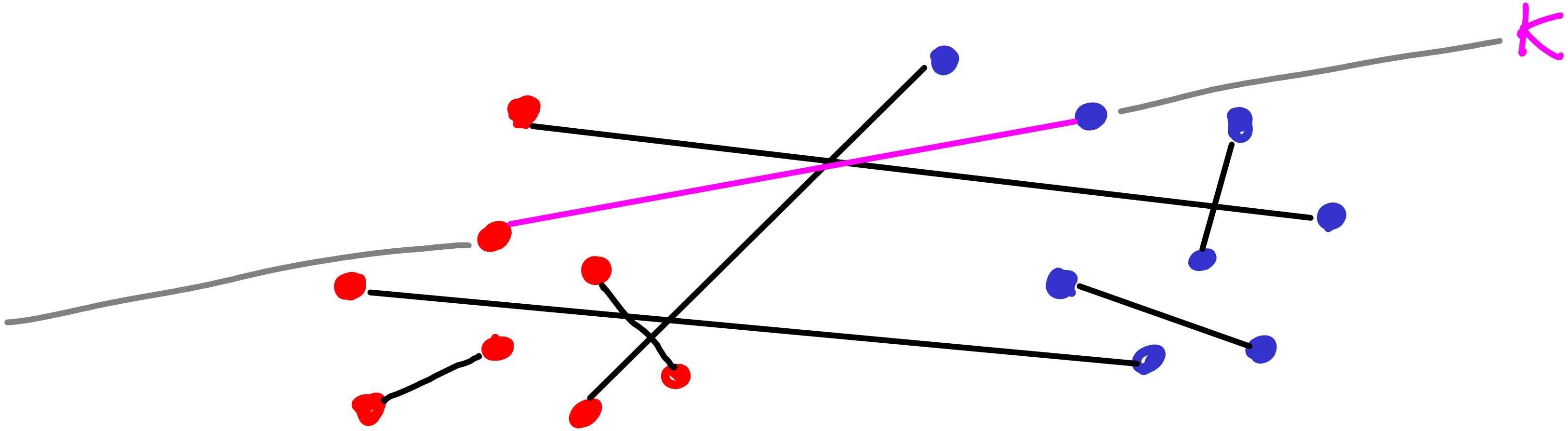



$a_x < b_x$

$K^*$ can't sweep below b

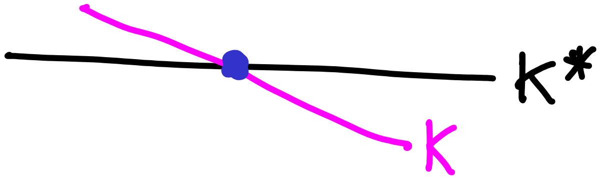
a can't be on bridge

(it could be on C.H.)

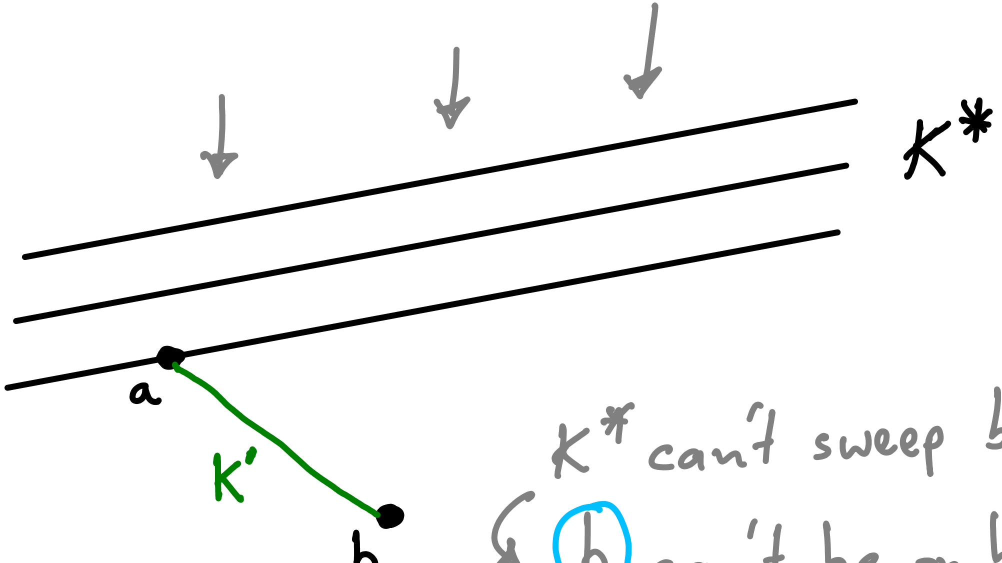



Case 2

$K < K^*$




Half of the pairs have slope $K' \le K$, so $K' < K^*$

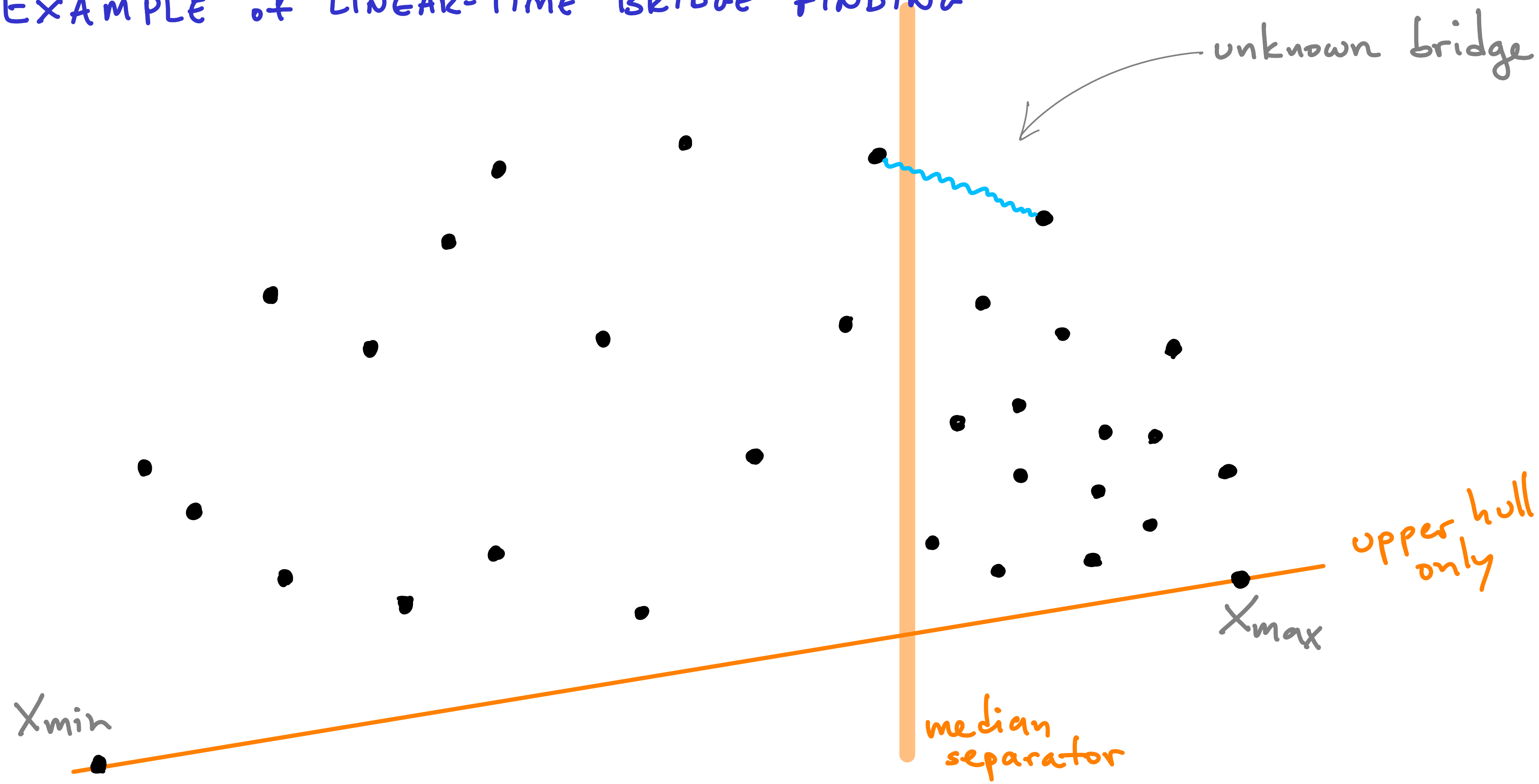



$K^*$ can't sweep below a

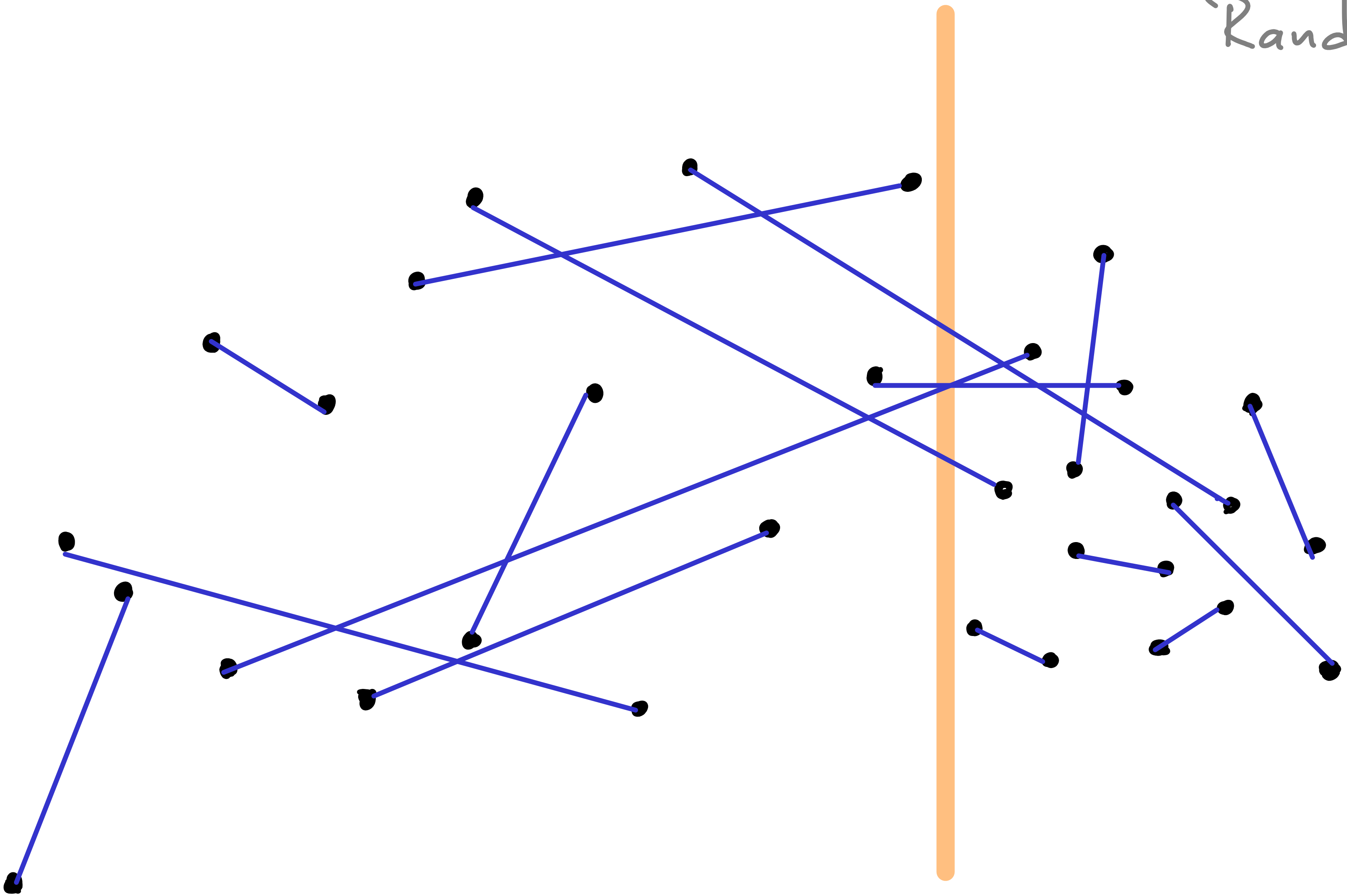
b can't be on bridge (it could be on C.H.)

$a_x < b_x$

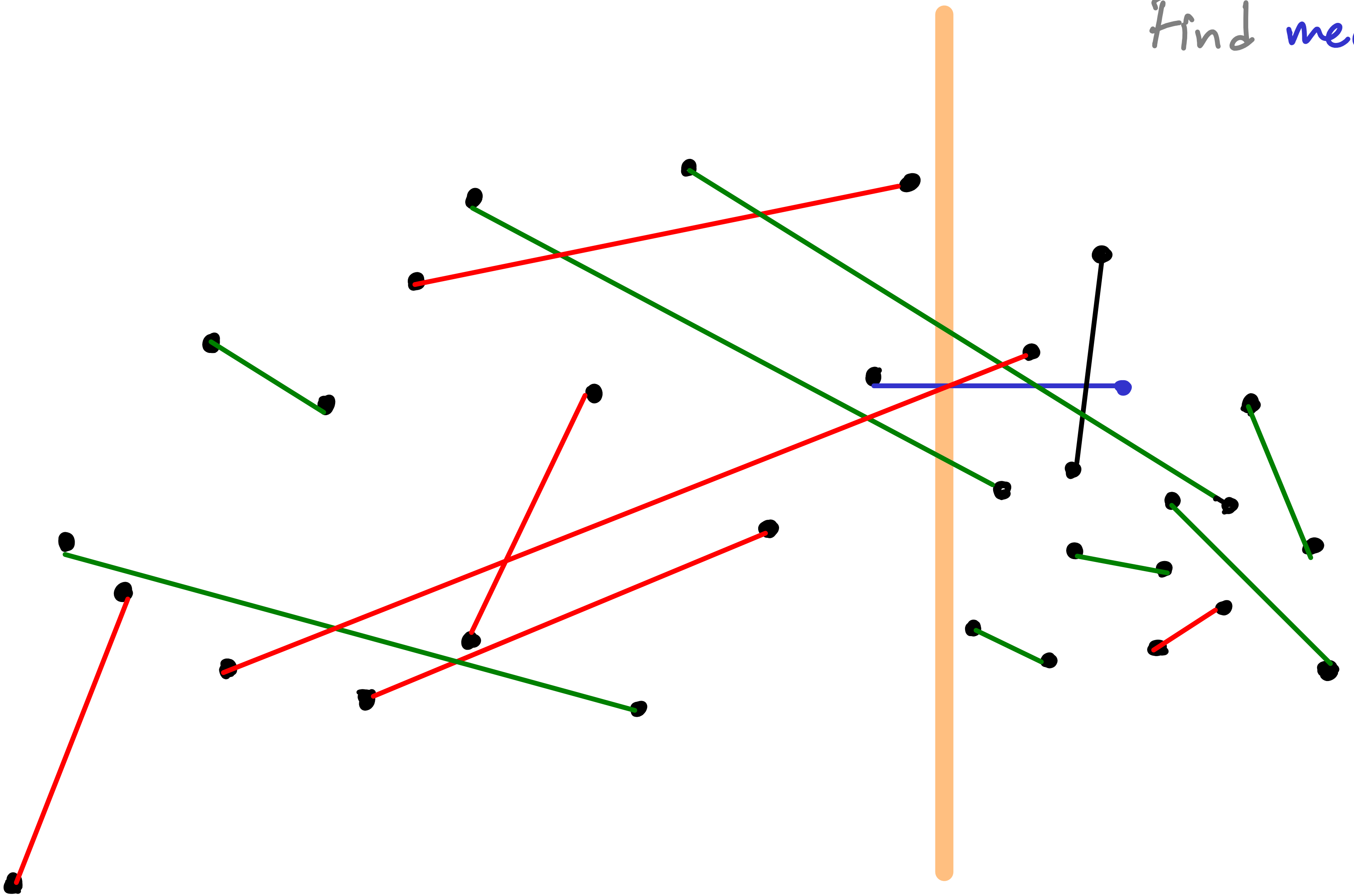
# EXAMPLE of LINEAR-TIME BRIDGE FINDING



Randomly pair points



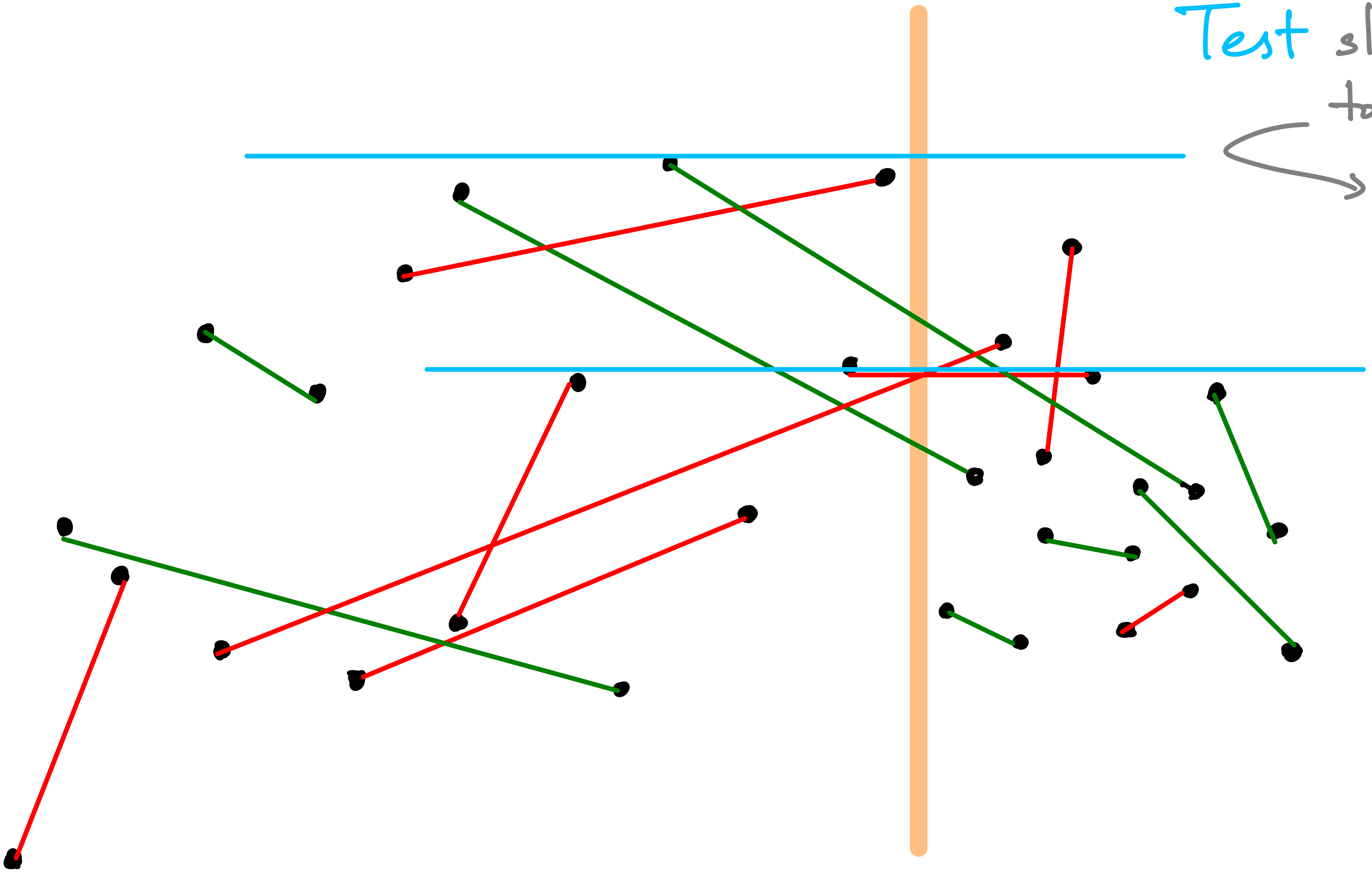
Find median slope



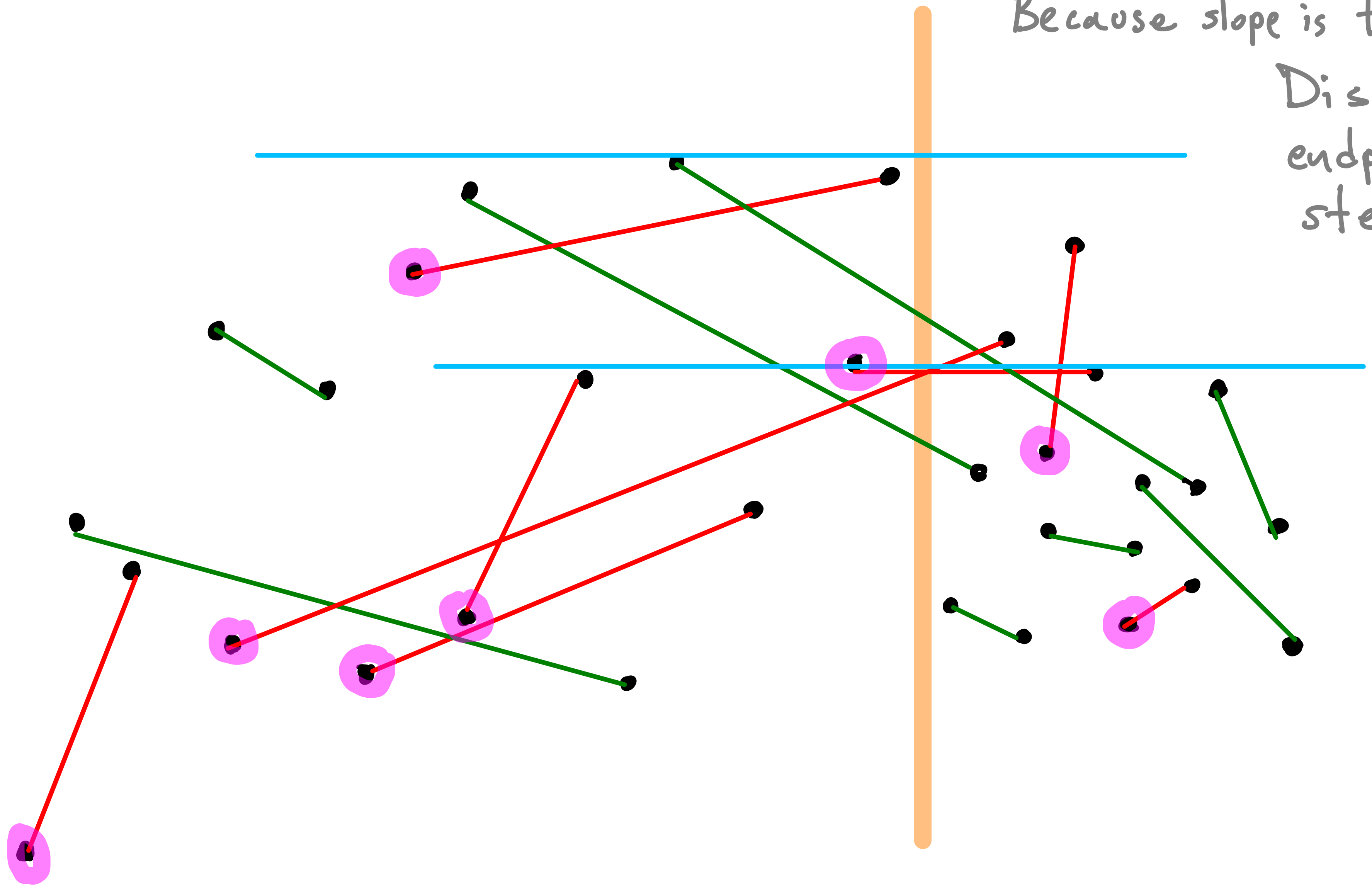
# Test slope:

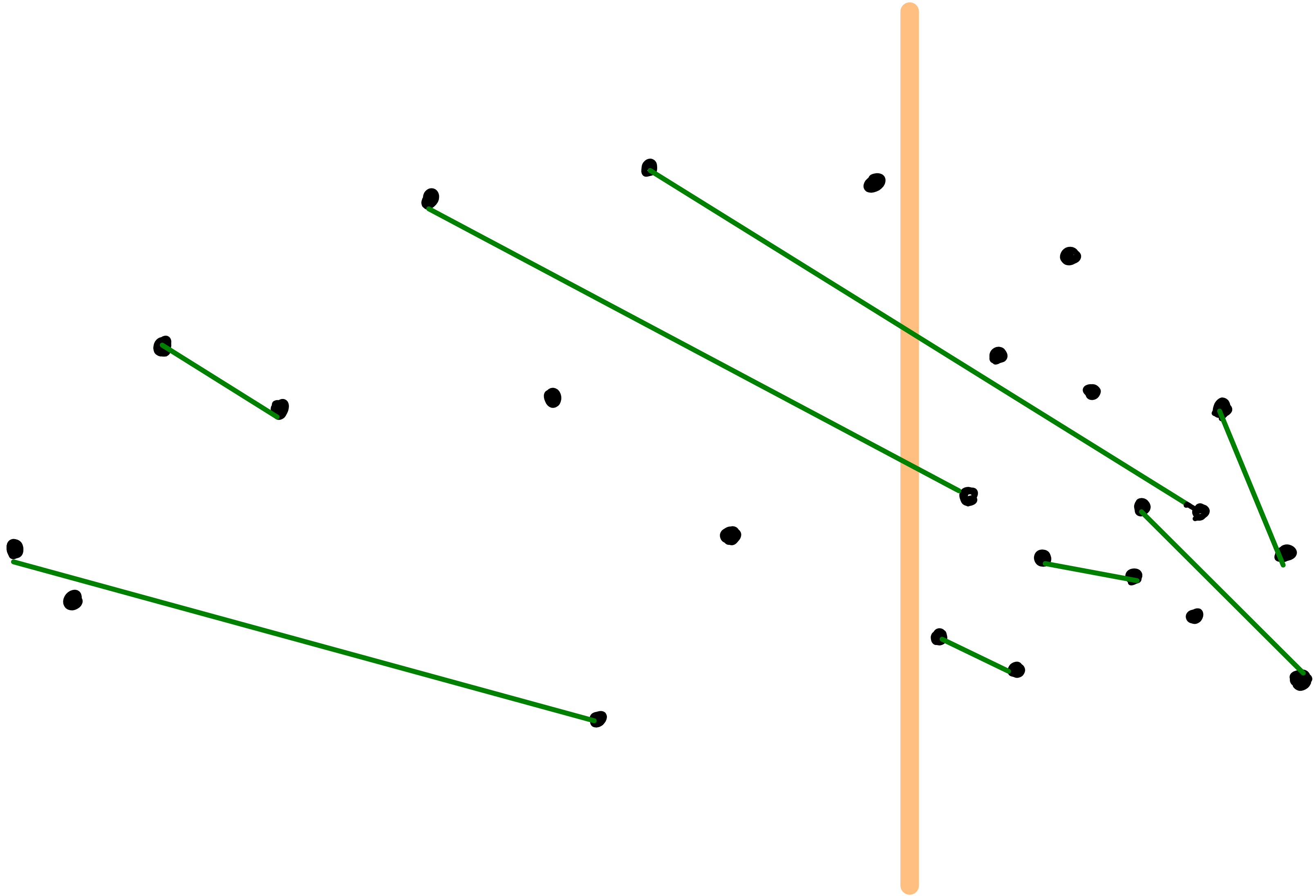
too steep

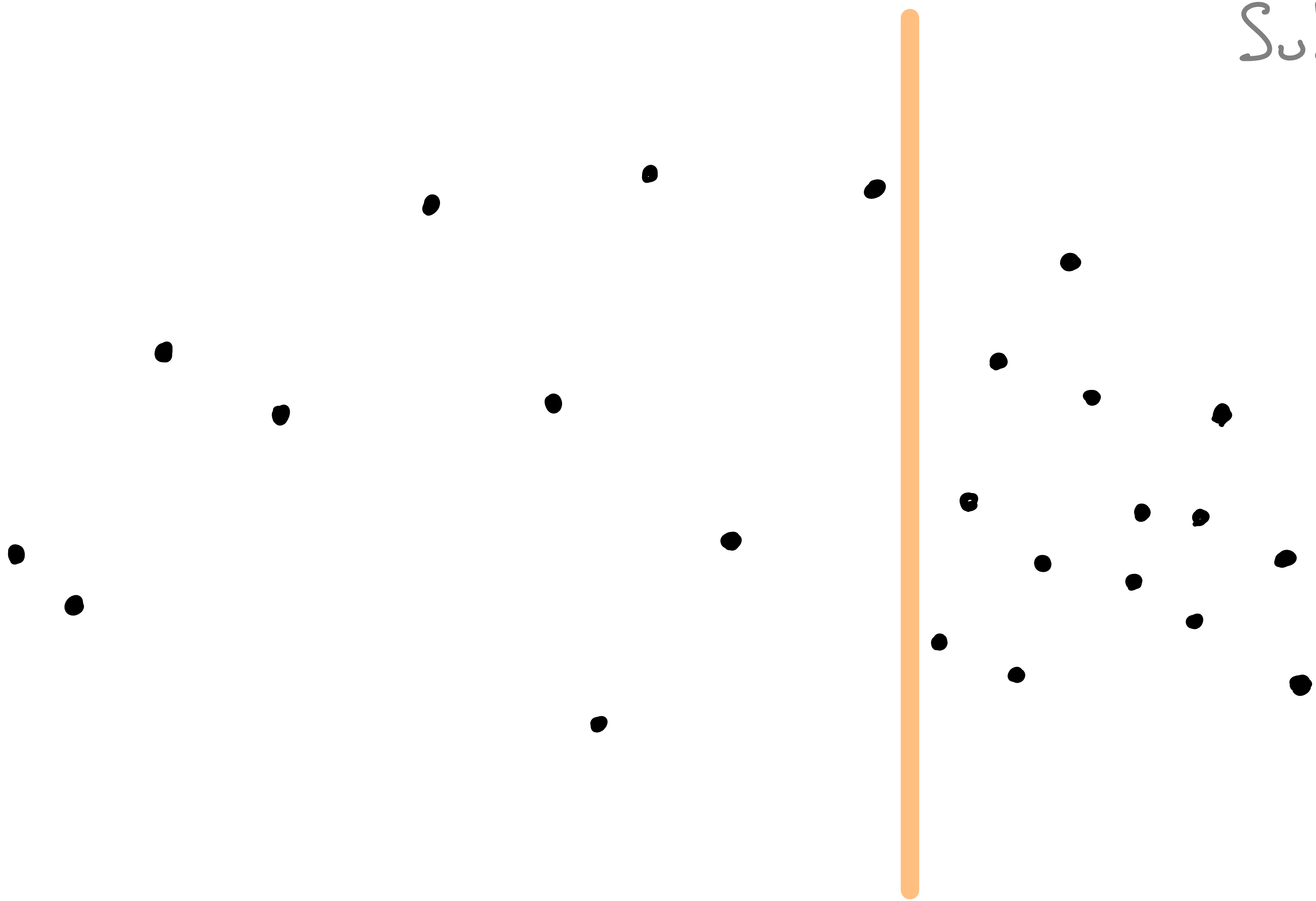
only left side is extremal



Because slope is too steep:
Discard left endpoints of steeper pairs

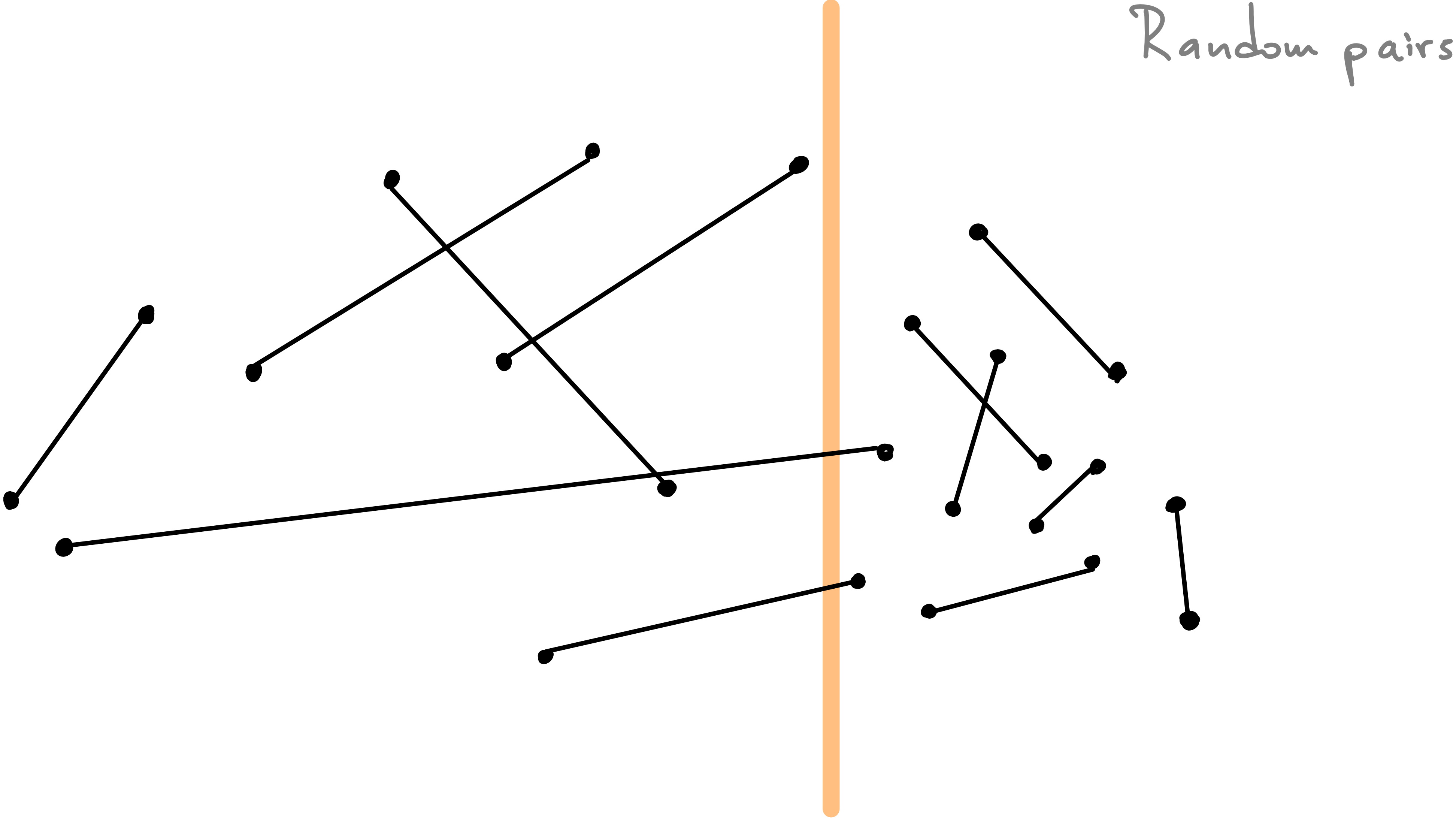




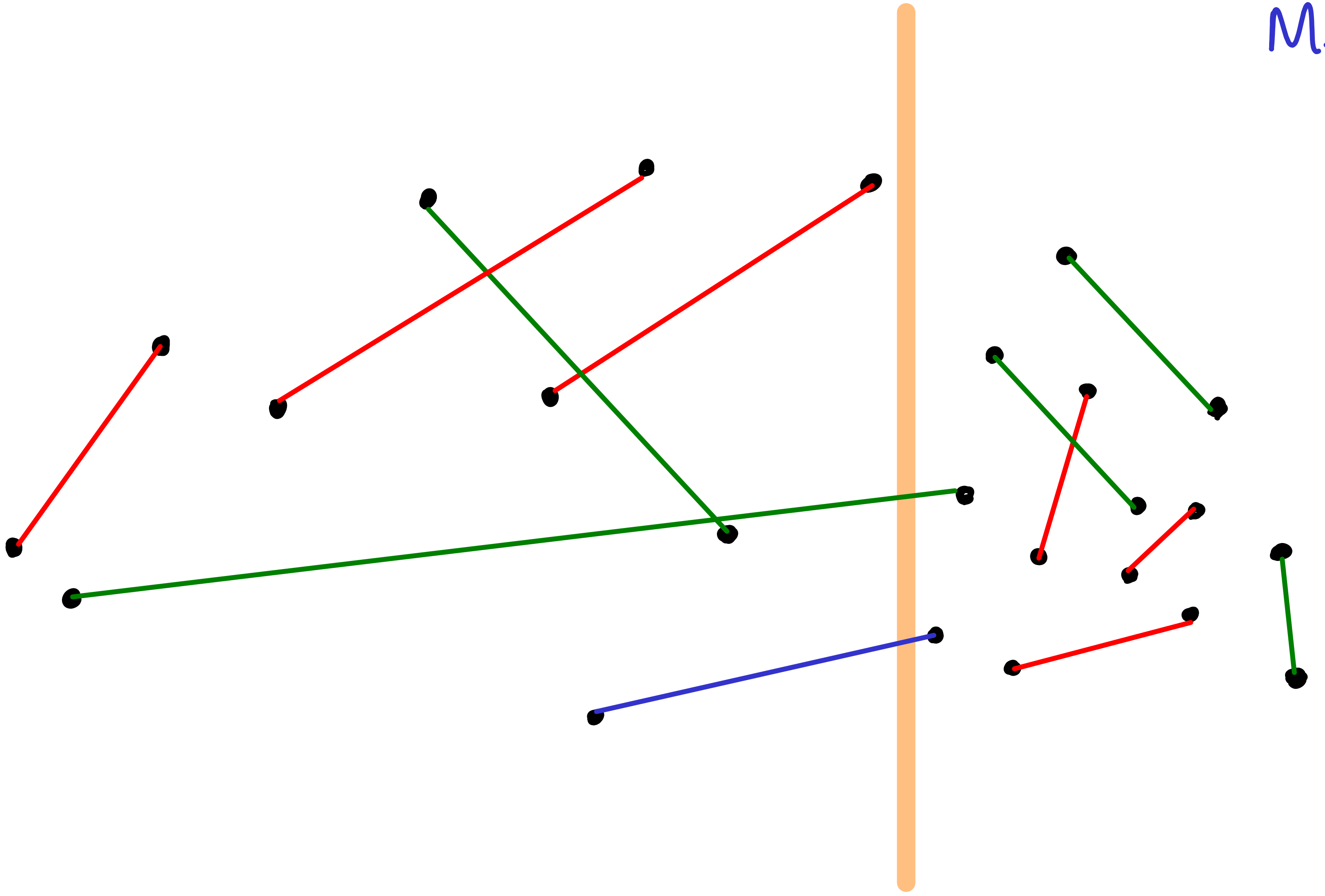


Subset :

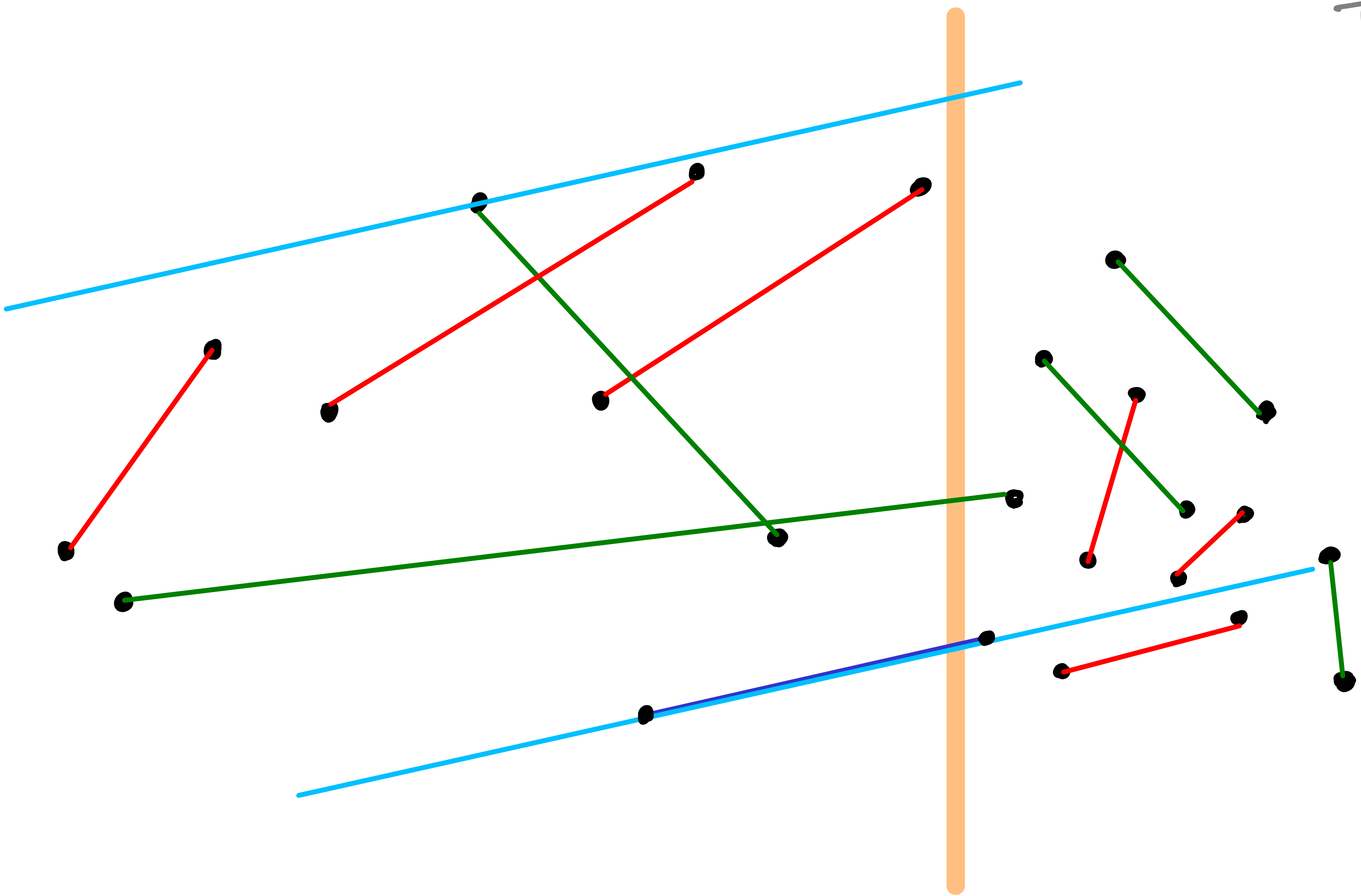
$\leq \frac{3}{4} \cdot \text{original}$

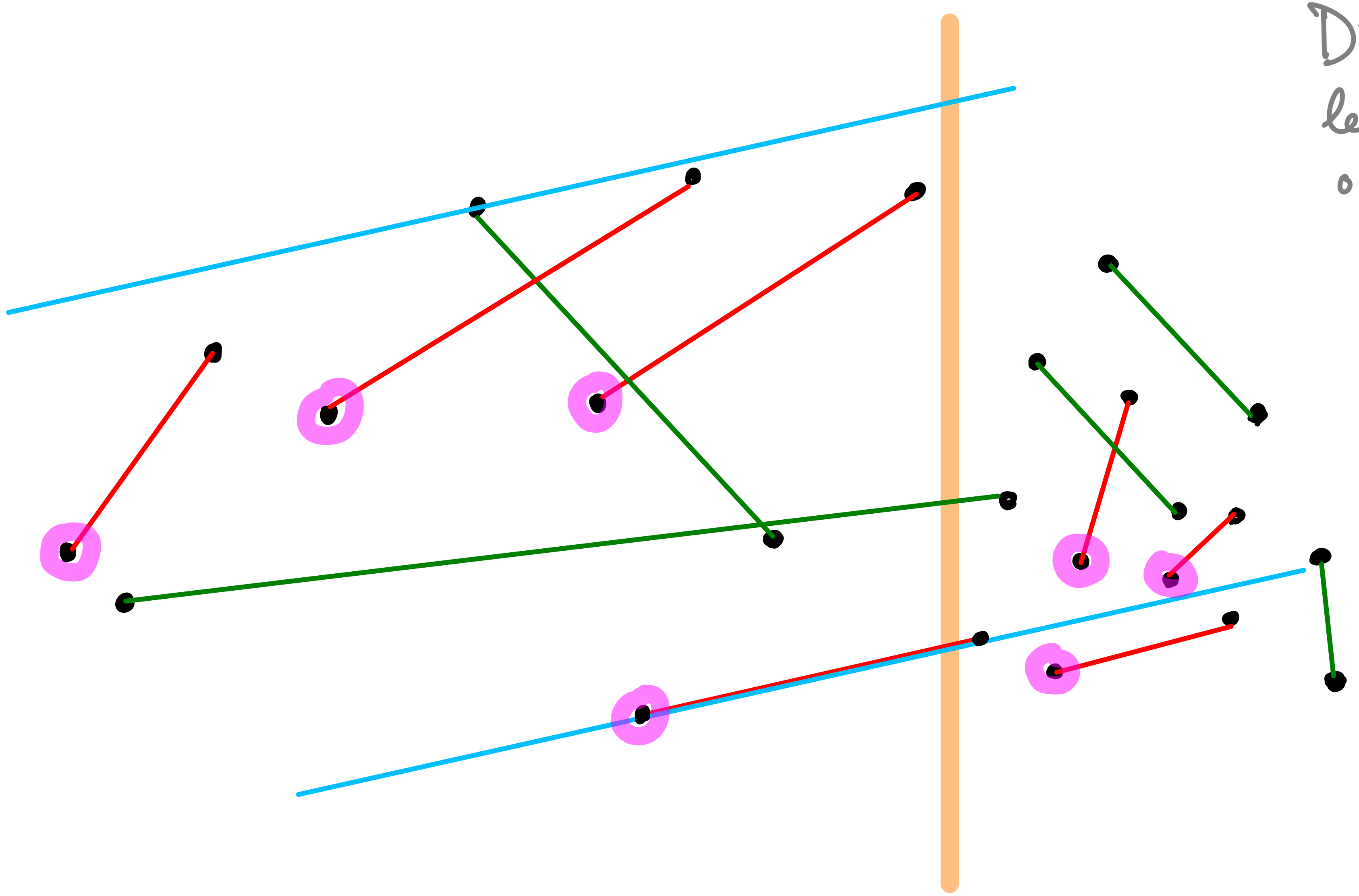

Random pairs

Median slope

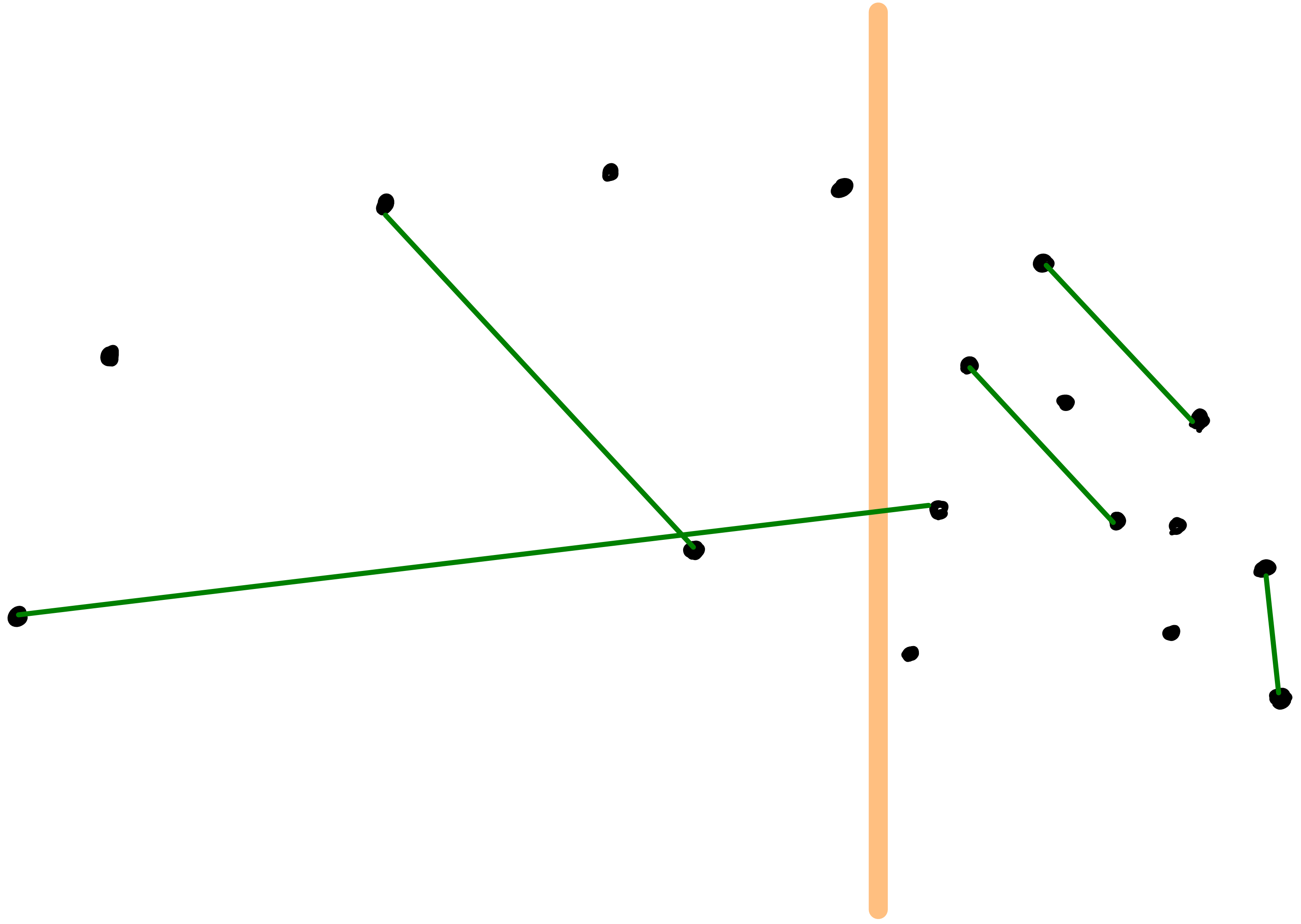


Too steep again



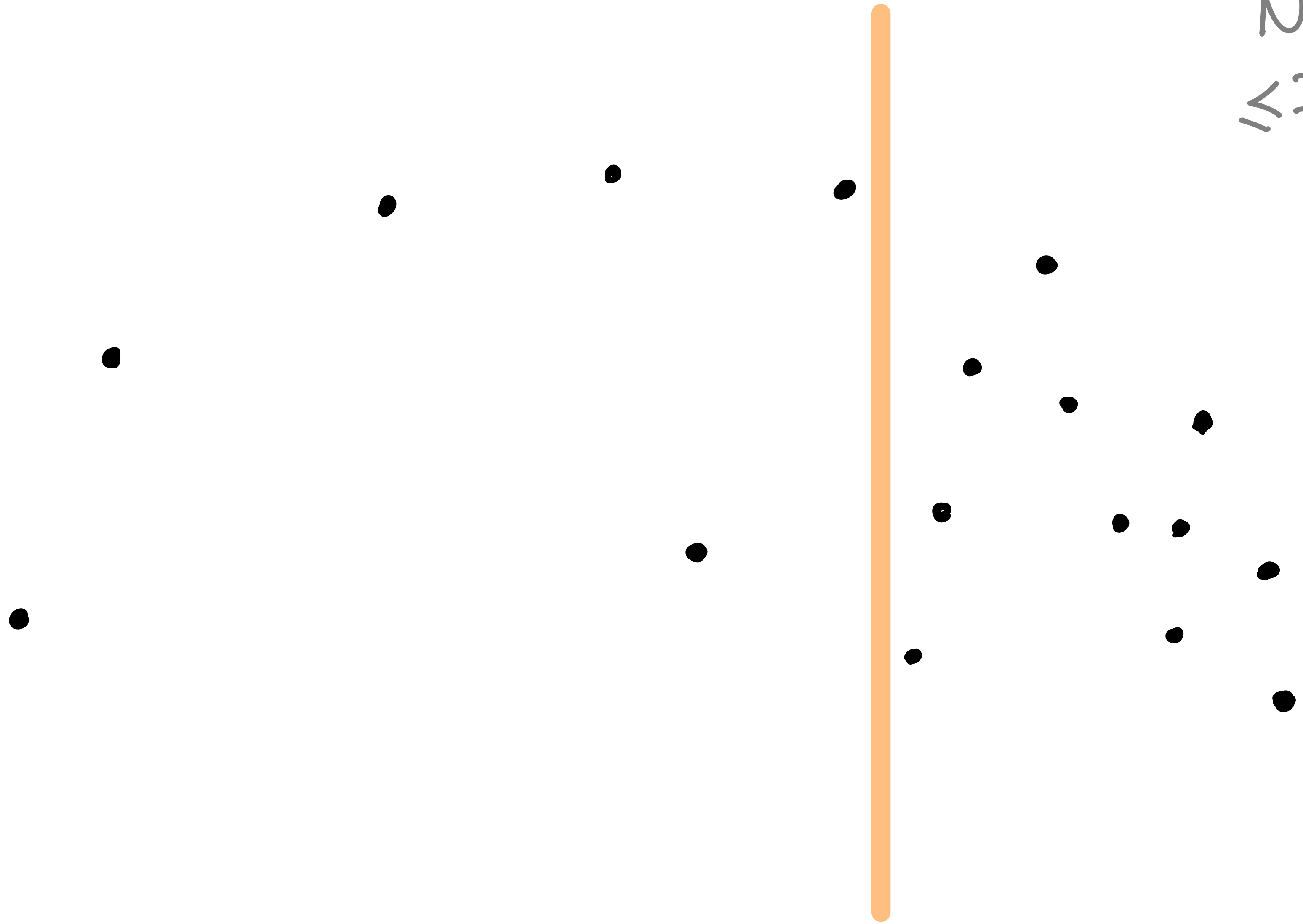


Discard
left endpoints
of steeper pairs

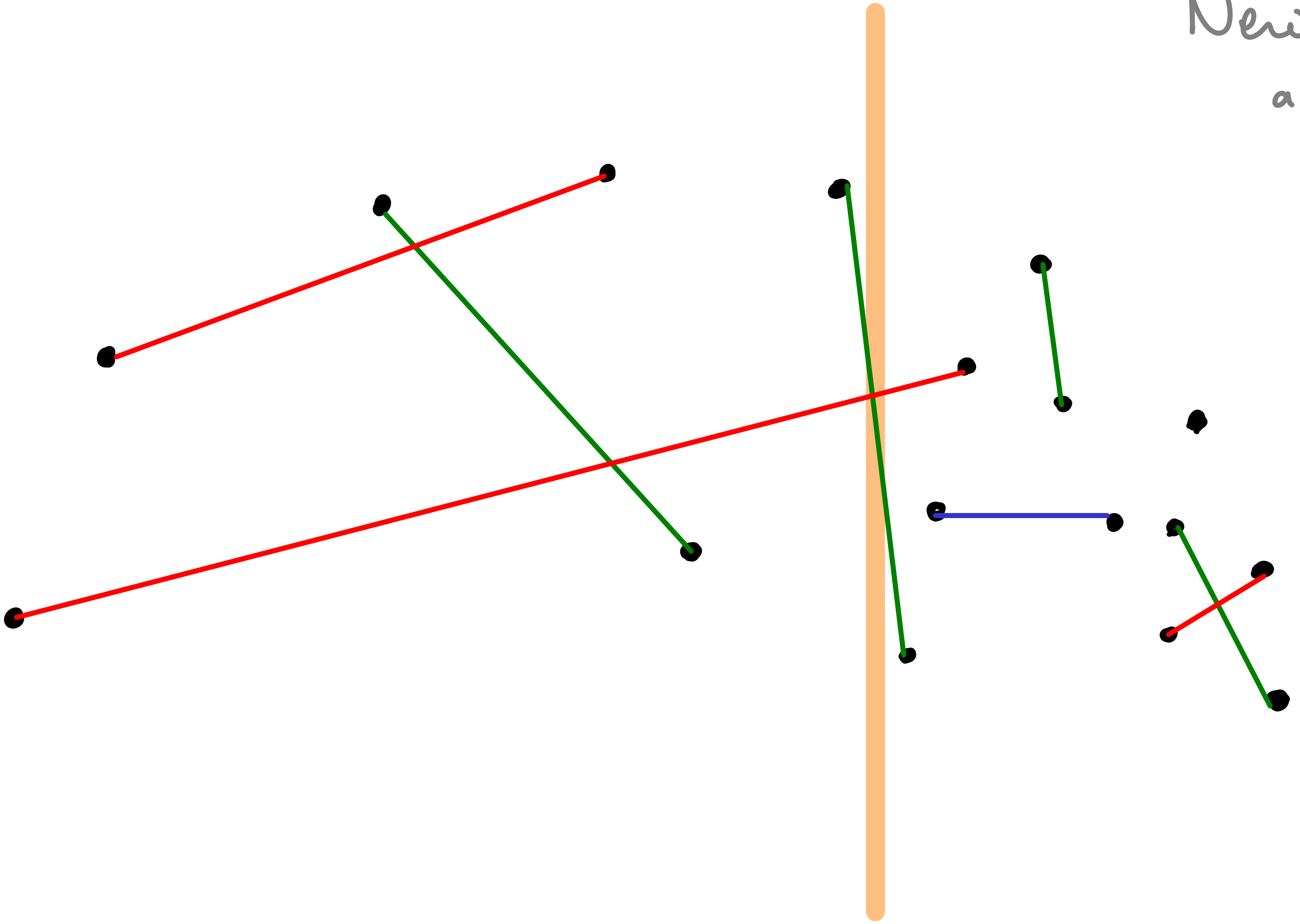


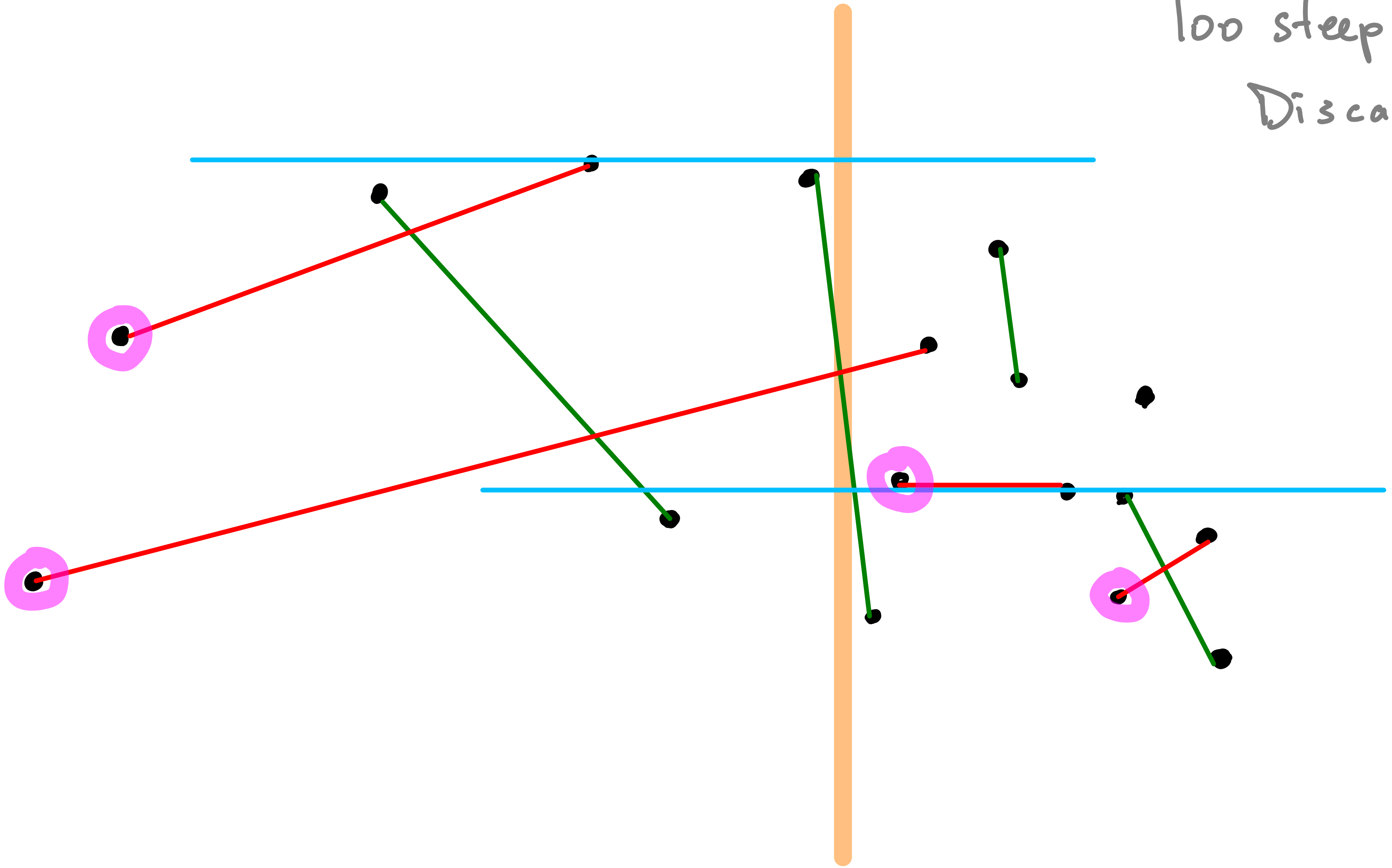
New subset

$\leq \frac{3}{4} \cdot \frac{3}{4} \cdot \text{original}$

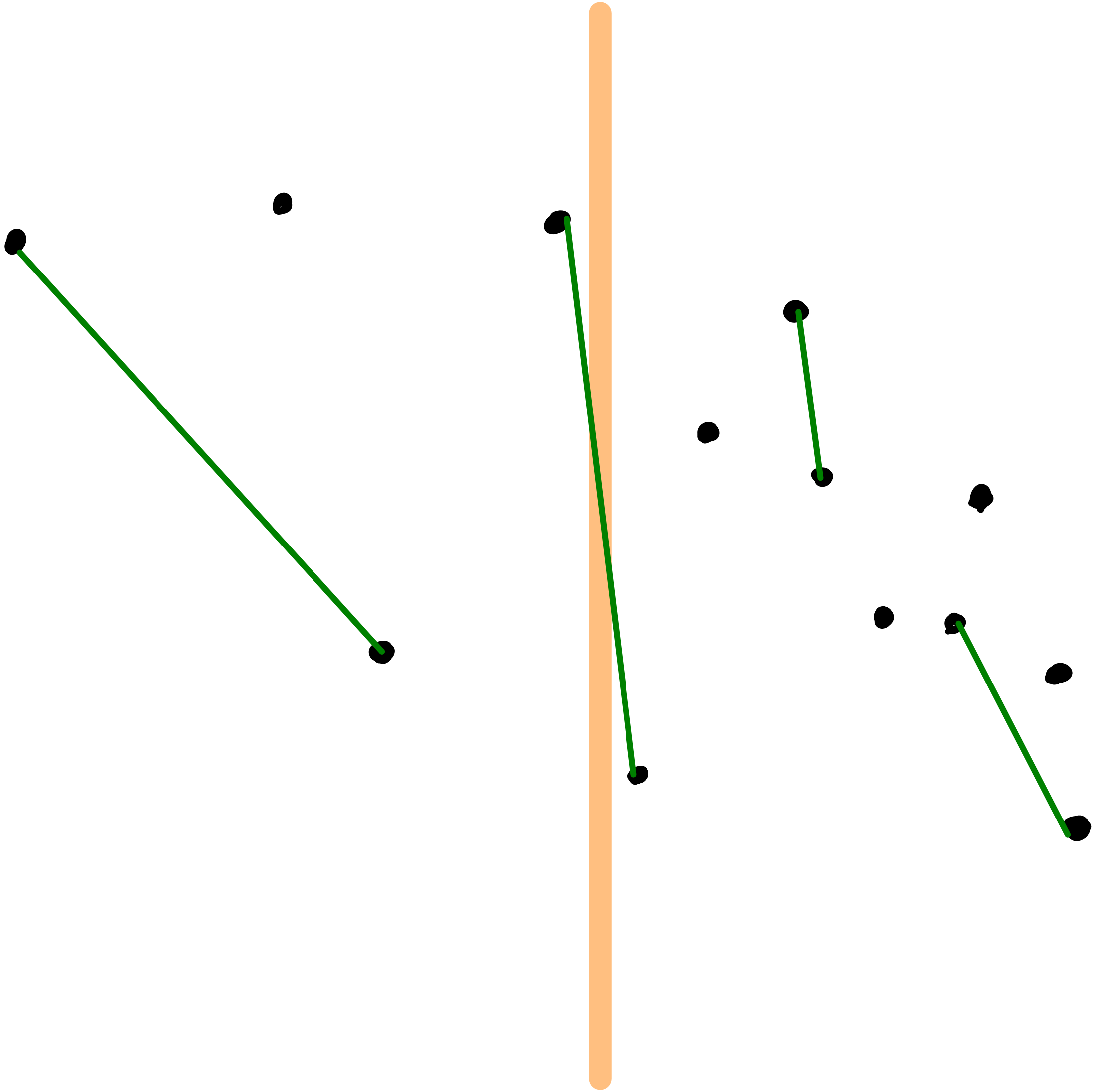


New random pairs
and median slope



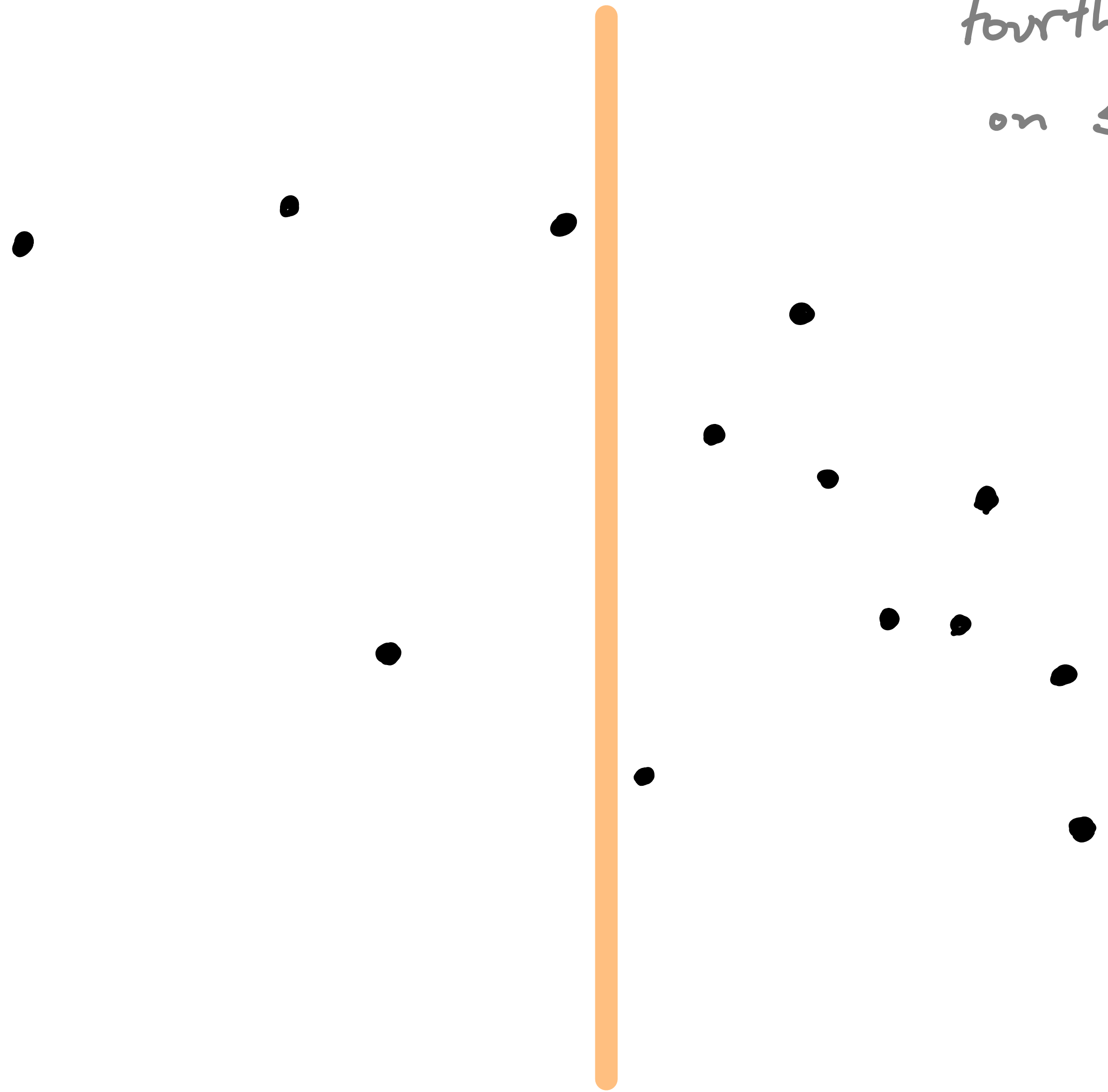


Too steep yet again
Discard...

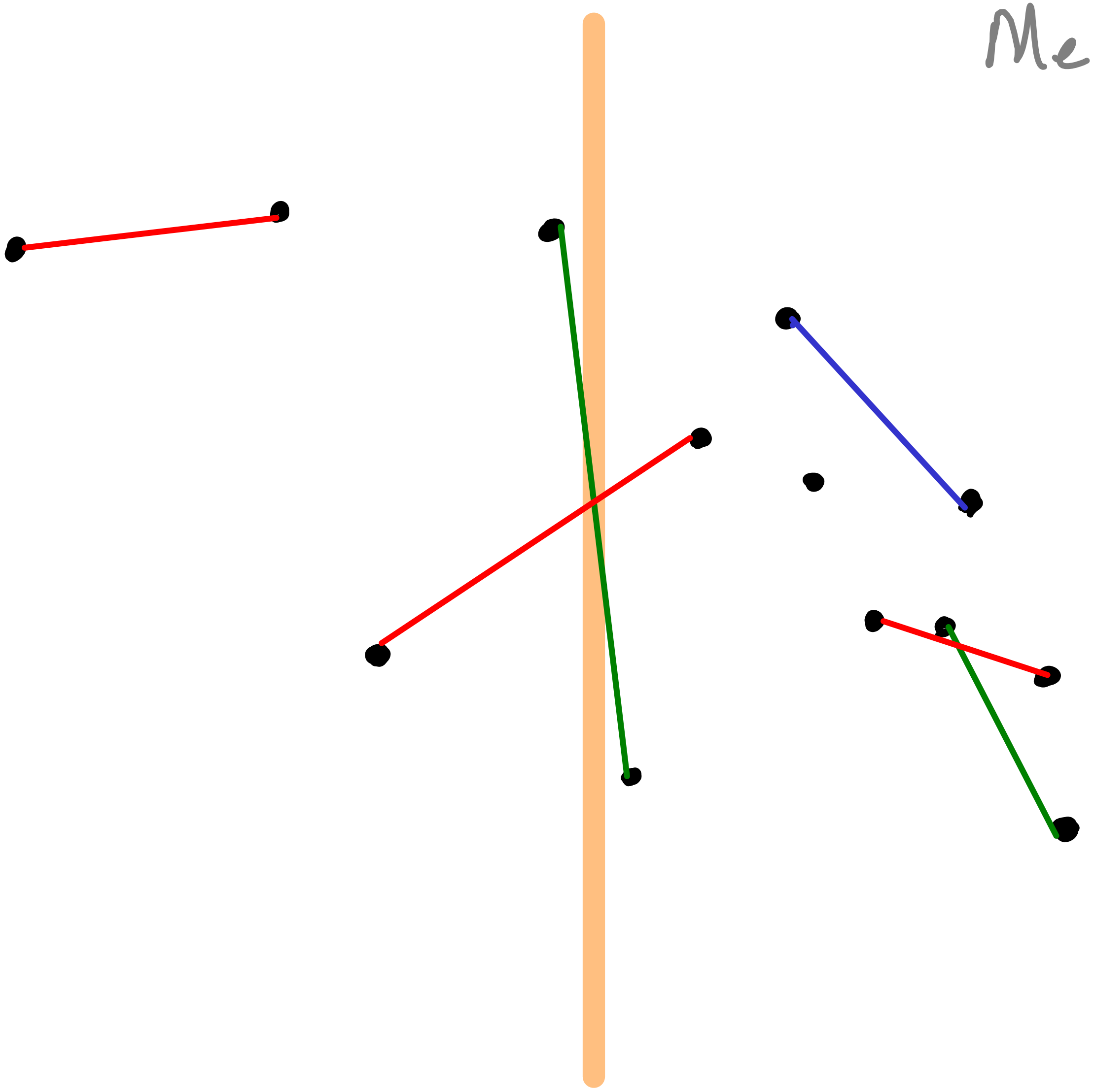


Fourth attempt, ...

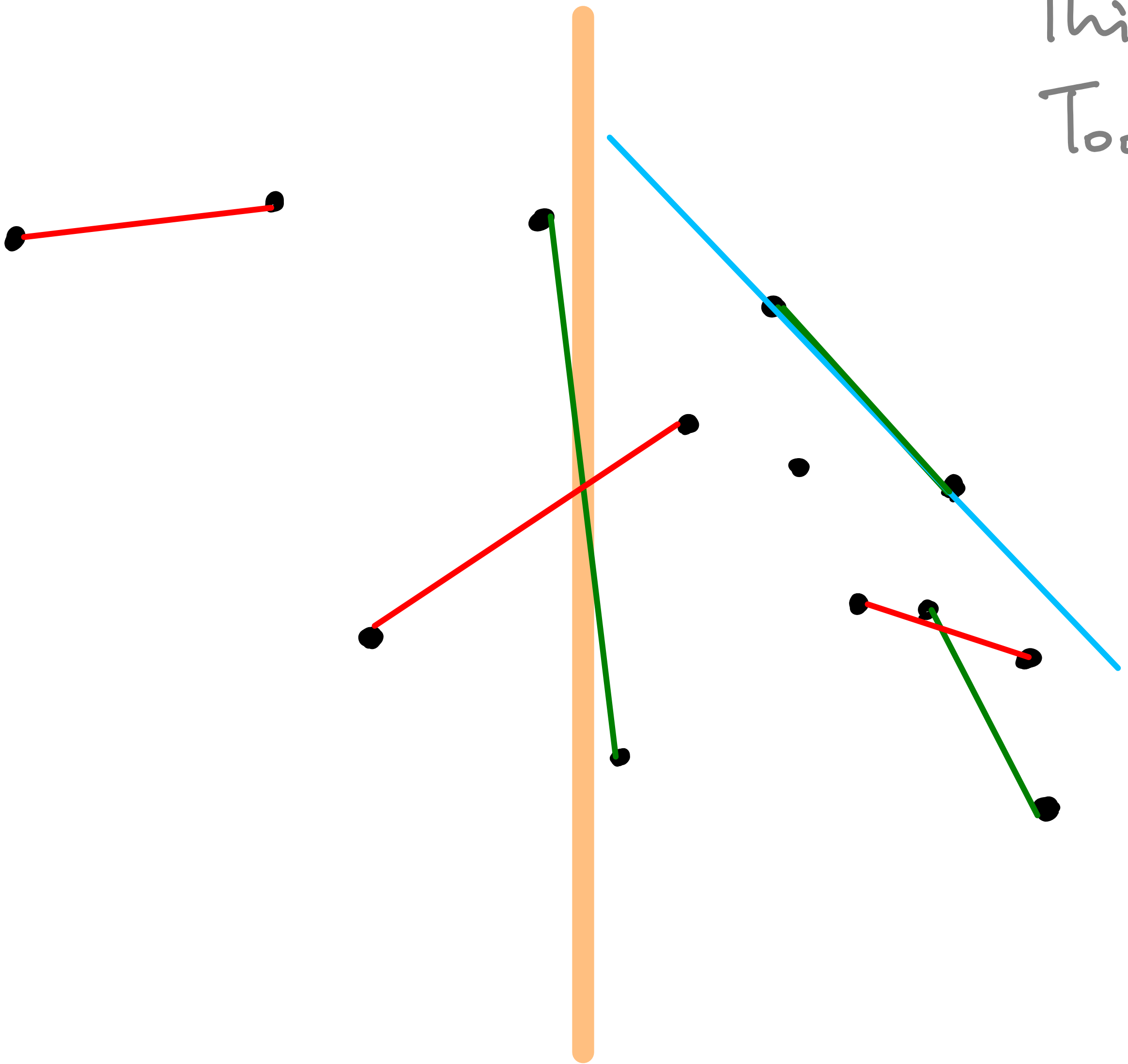
on $\leq \frac{3}{4} \cdot \frac{3}{4} \cdot \frac{3}{4} \cdot$ original



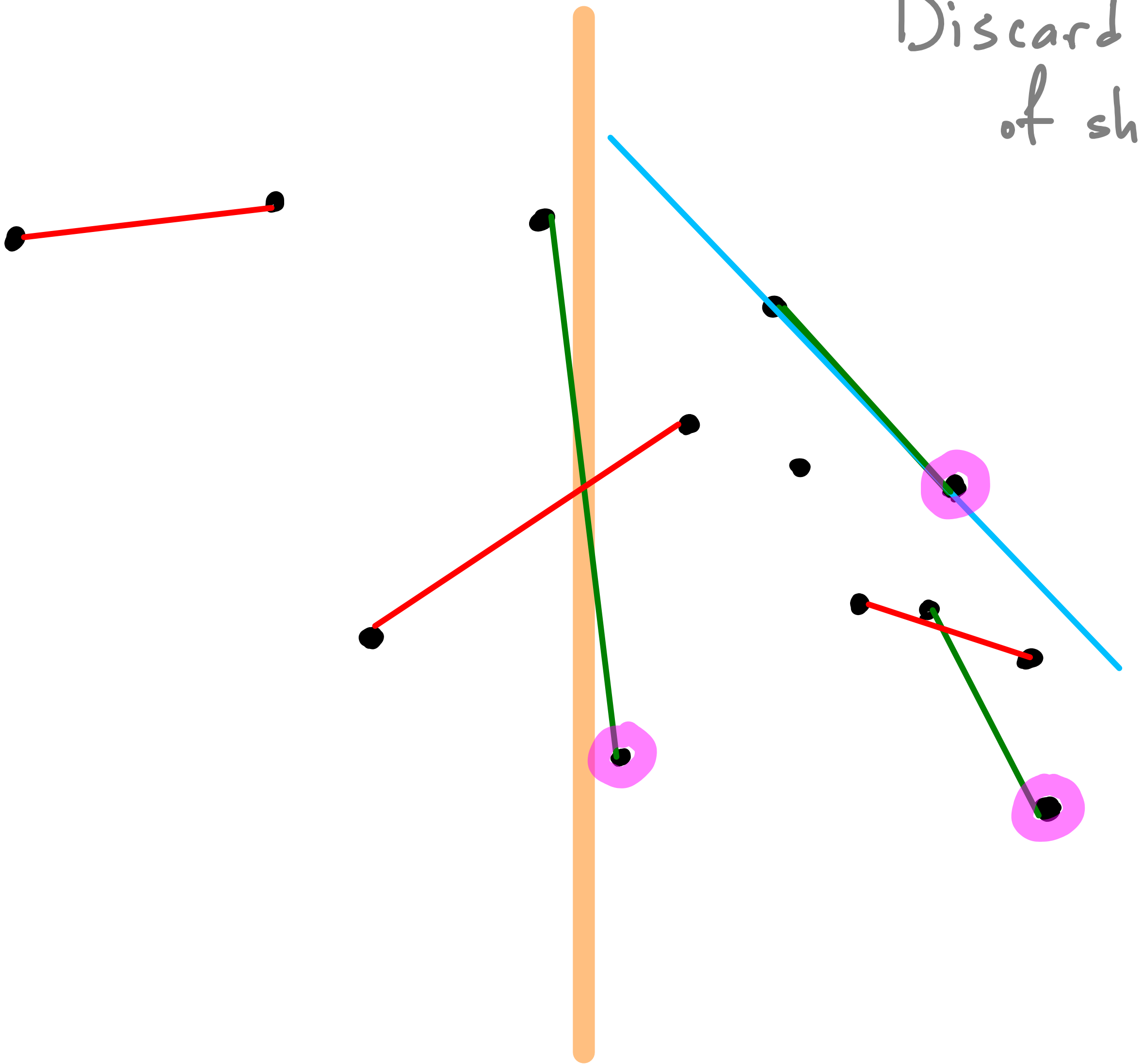
Median slope

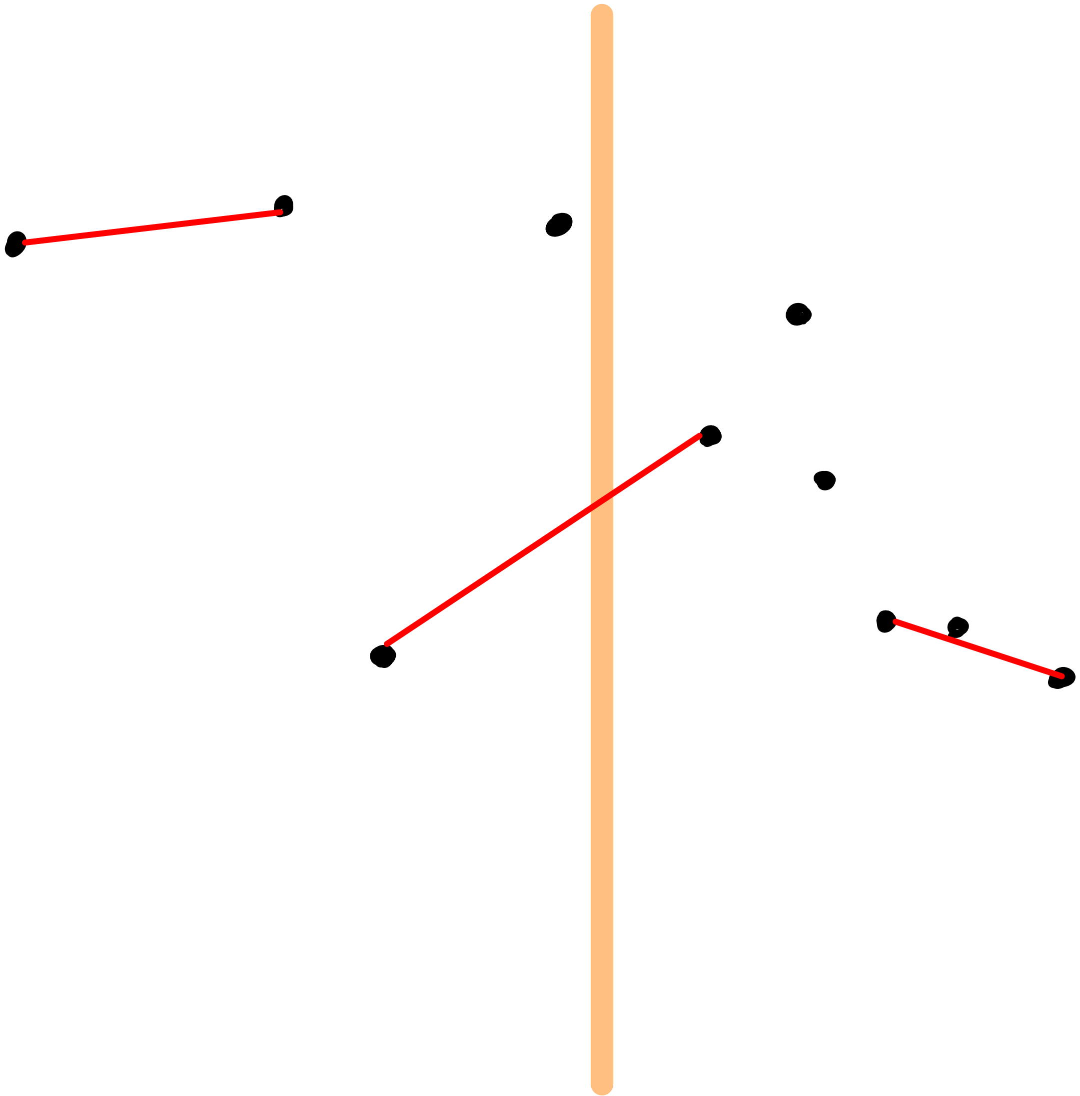


This time:
Too shallow



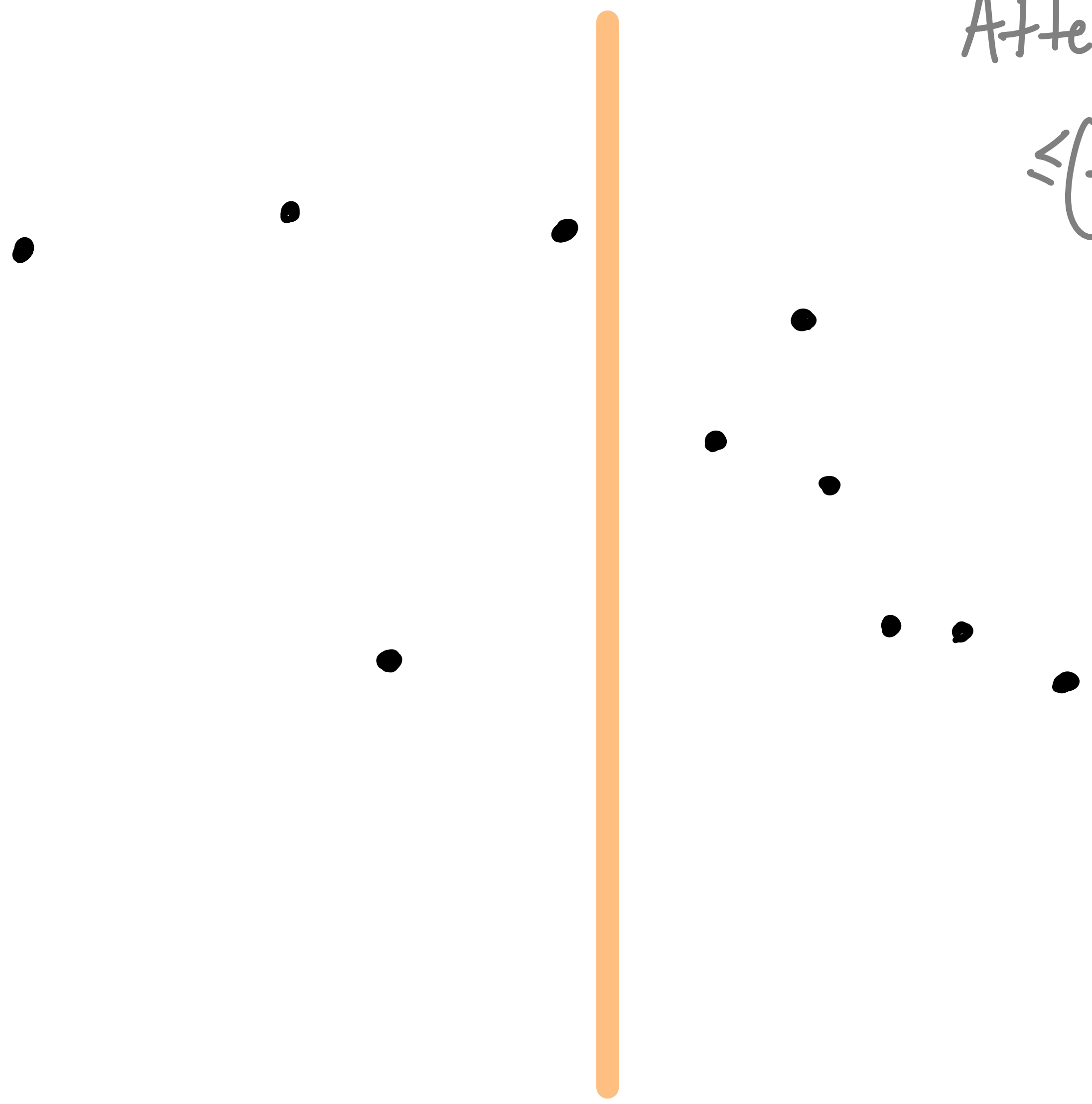
Discard right endpoints
of shallower pairs

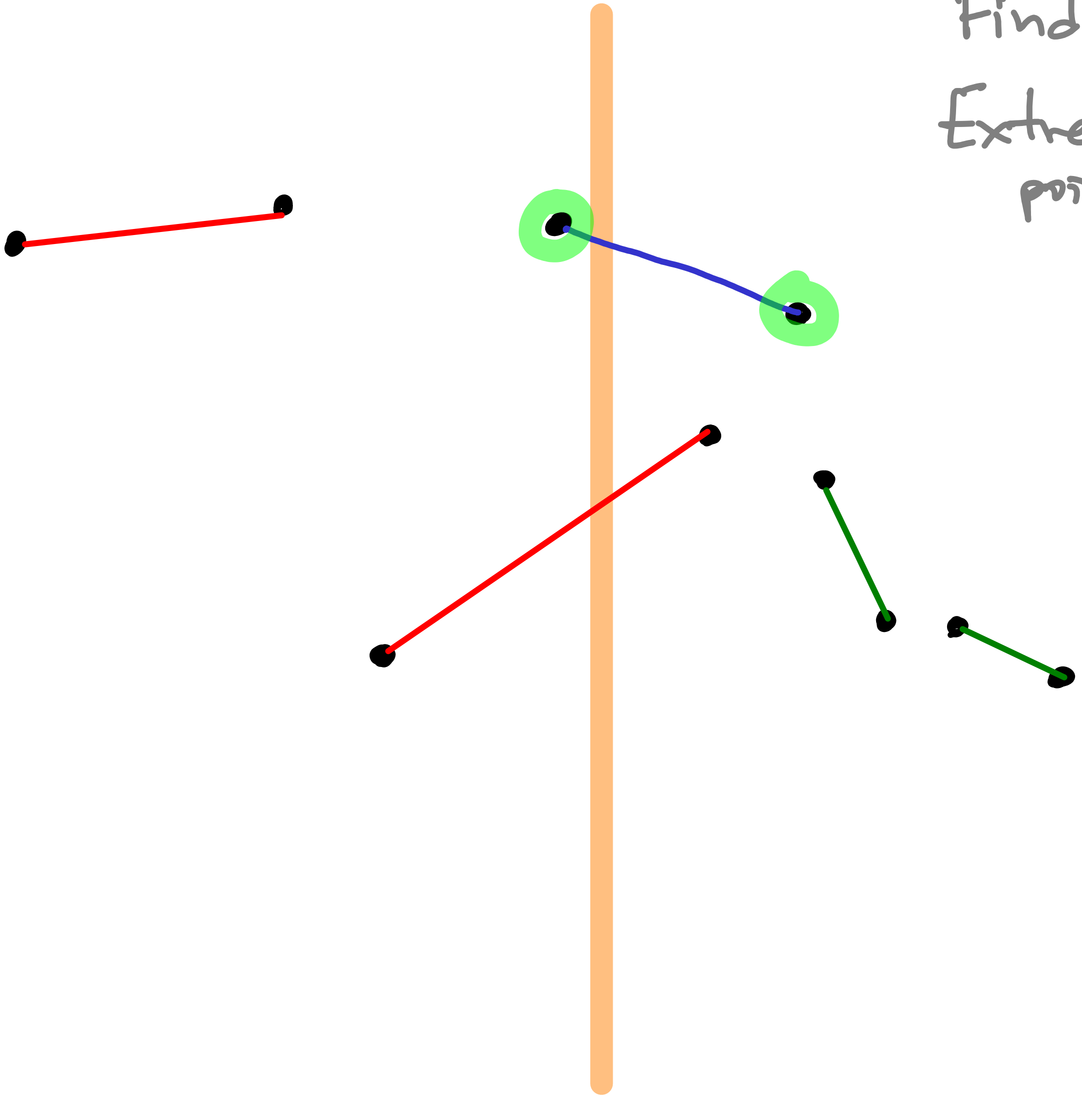




Attempt #5

$\leq \left(\frac{3}{4}\right)^4 \cdot \text{original}$



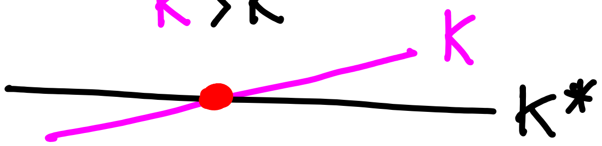


Find median slope.

Extreme finds one point on each side.

↳ DONE

**Case 1** $k > k^*$

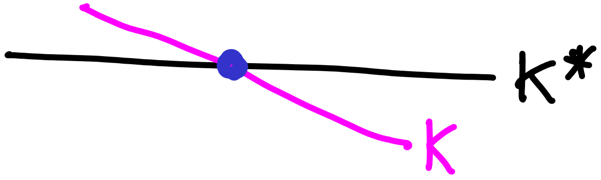



Half of the pairs have slope $k' \geq k$, so

$k' > k^*$

(a) can't be on bridge (it could be on C.H.)

**Case 2** $k < k^*$




Half of the pairs have slope $k' \leq k$, so

$k' < k^*$

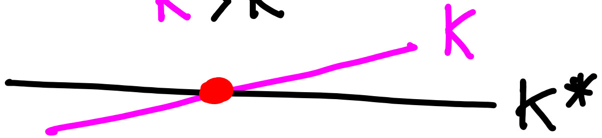
(b) can't be on bridge (it could be on C.H.)

THROW AWAY
ONE POINT (a or b)
FROM HALF THE PAIRS

$\Downarrow$

1/4 points

Case 1 $K > K^*$

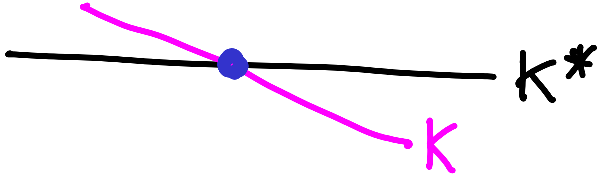



Half of the pairs
have slope $K' \geq K$, so

$K' > K^*$

a can't be on bridge
(it could be on C.H.)

Case 2 $K < K^*$




Half of the pairs
have slope $K' \leq K$, so

$K' < K^*$

b can't be on bridge
(it could be on C.H.)

If we guess wrong :

THROW AWAY
ONE POINT (a or b)
FROM HALF THE PAIRS

If we guess wrong : THROW AWAY ONE POINT (a or b) FROM HALF THE PAIRS

Then arbitrarily pair remaining points & "guess" again

If we guess wrong : THROW AWAY ONE POINT (a or b) FROM HALF THE PAIRS

Then arbitrarily pair remaining points & "guess" again

Time: $c \cdot n$ for first wrong guess

$c \cdot \frac{3n}{4}$ for second " "

$c \cdot \frac{3}{4} \cdot \frac{3}{4}n$ for third.

If we guess wrong: THROW AWAY ONE POINT (a or b) FROM HALF THE PAIRS

Then arbitrarily pair remaining points & "guess" again

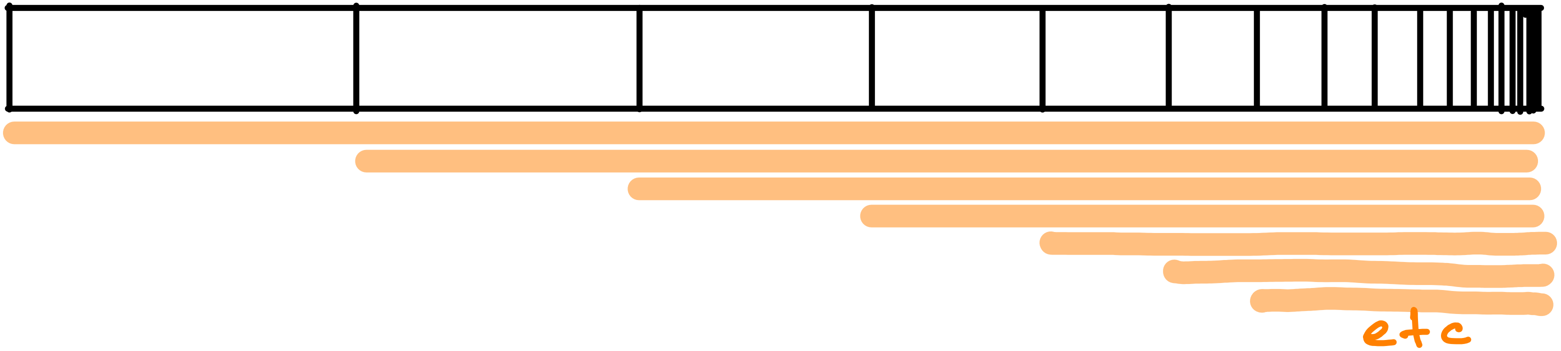
Time: $c \cdot n$ for first wrong guess

$c \cdot \frac{3n}{4}$ for second " "

$c \cdot \frac{3}{4} \cdot \frac{3n}{4}$ for third.

etc

total: $O(n)$



# "PRUNE & SEARCH"

If you can throw out a constant fraction of your input whenever you fail, then you will still have a good algorithm.

$$T(n) = F(n) + T\left(\frac{n}{c}\right) \qquad [c > 1]$$

# "PRUNE & SEARCH"

If you can throw out a constant fraction of your input whenever you fail, then you will still have a good algorithm.

$$T(n) = F(n) + T\left(\frac{n}{c}\right) \qquad [c > 1]$$

$$\downarrow$$

$$O(1) \qquad [c=2]$$

# "PRUNE & SEARCH"

If you can throw out a constant fraction of your input whenever you fail, then you will still have a good algorithm.

$$T(n) = F(n) + T\left(\frac{n}{c}\right) \qquad [c > 1]$$

$O(\log n)$ $\;O(1)$ : binary search $[c=2]$

$O(n)$ $\qquad [c = \frac{4}{3}]$

# "PRUNE & SEARCH"

If you can throw out a constant fraction of your input whenever you fail, then you will still have a good algorithm.

$$T(n) = F(n) + T\left(\frac{n}{c}\right) \qquad [c > 1]$$

$O(\log n)$ $\quad O(1)$ : binary search $[c=2]$

$O(n)$ $\quad O(n)$ : finding a bridge $[c=\frac{4}{3}]$

# "PRUNE & SEARCH"

If you can throw out a constant fraction of your input whenever you fail, then you will still have a good algorithm.

$$T(n) = F(n) + T\left(\frac{n}{c}\right) \qquad [c > 1]$$

$O(\log n)$ $\quad O(1)$ : binary search $[c=2]$

$O(n)$ $\quad O(n)$ : finding a bridge $[c=\frac{4}{3}]$

$O(n^k)$ $\quad O(n^k)$ $\quad n^k + \frac{n^k}{2^k} + \frac{n^k}{4^k} + \cdots \frac{n^k}{2^{ik}}$ $\quad [c=2]$

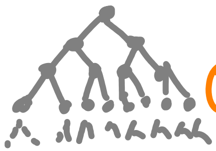
# "PRUNE & SEARCH"

If you can throw out a constant fraction of your input whenever you fail, then you will still have a good algorithm.

$$T(n) = F(n) + T\left(\frac{n}{c}\right) \qquad [c > 1]$$

| | | |
|---|---|---|
| $O(\log n)$ | $O(1)$ | : binary search $[c=2]$ |
| $O(n)$ | $O(n)$ | : finding a bridge $[c=\frac{4}{3}]$ |
| $O(n^k)$ | $O(n^k)$ | $n^k + \frac{n^k}{2^k} + \frac{n^k}{4^k} + \dots \frac{n^k}{2^{ik}}$ $[c=2]$ |
| $O(2^n)$ | $O(2^n)$ | $2^n + 2^{n-1} + 2^{n-2} + \dots + 2$ $[c=2]$ |

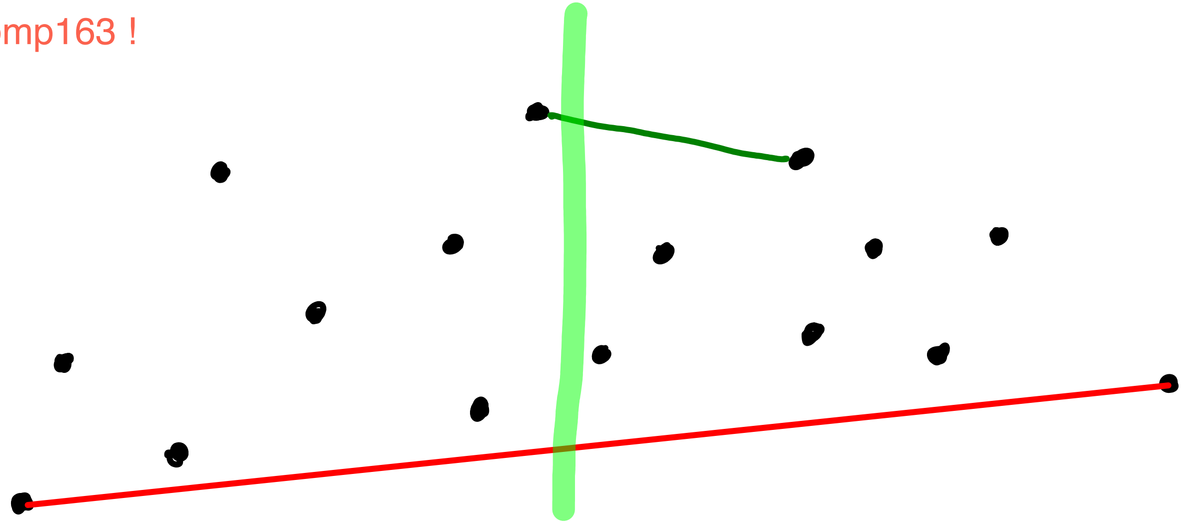
search leaves,
if "fail" then search parents
etc



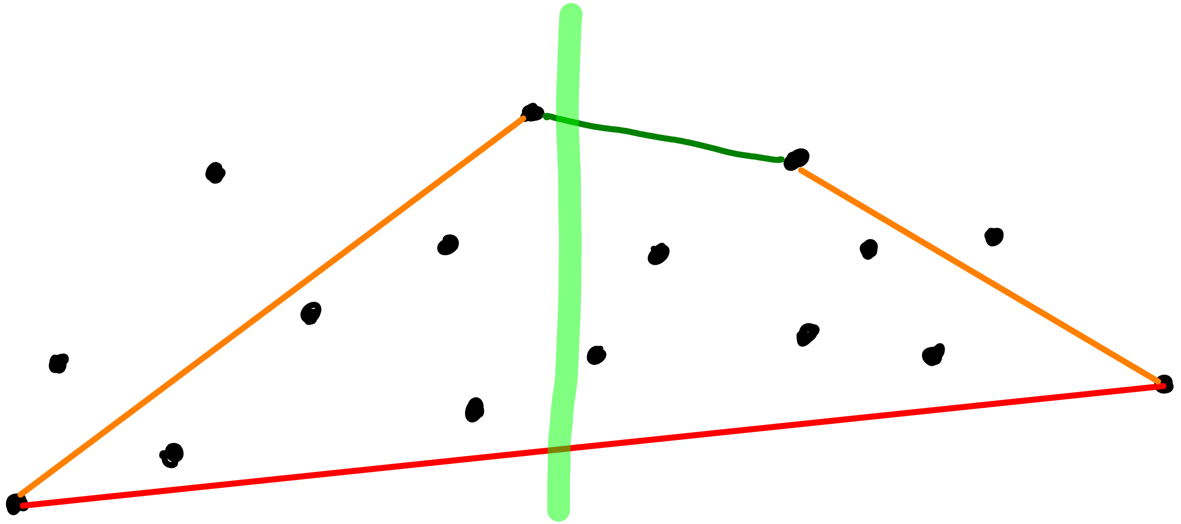
A little more context, for comp260. Wasn't discussed in class.

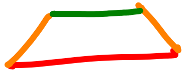
Take comp163 !

We know how to find a bridge in linear time

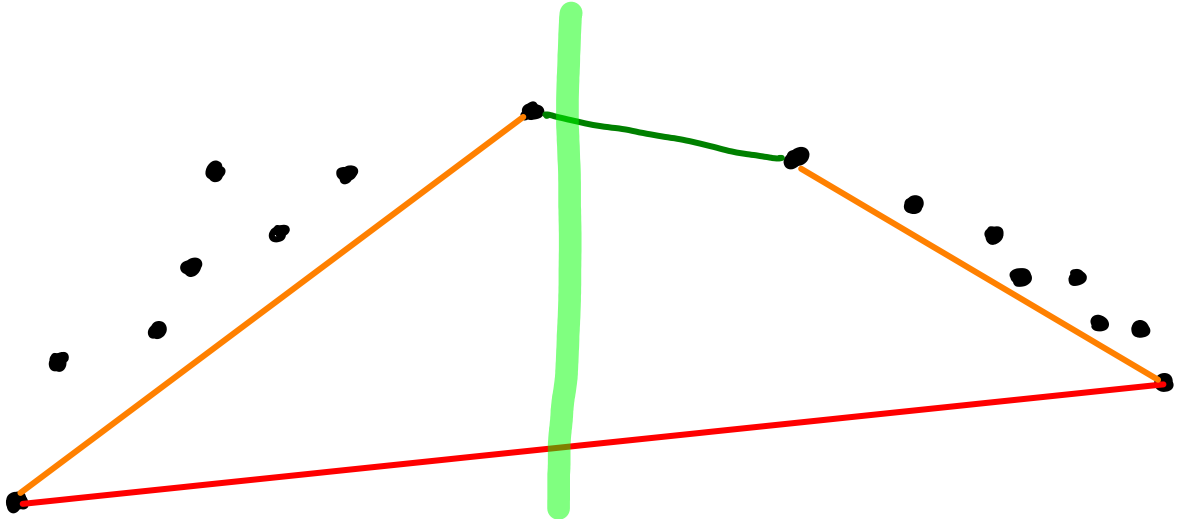


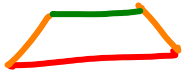
We know how to find a bridge in linear time



Might as well throw out potential non-C.H. pts inside  ... it's "free"

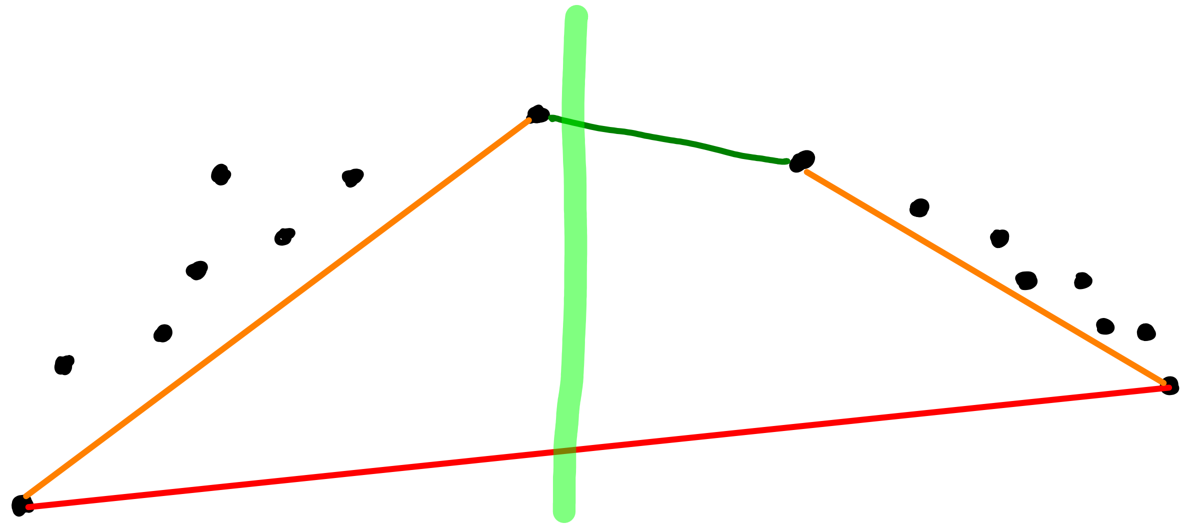
We know how to find a bridge in linear time



Might as well throw out potential non-C.H. pts inside  ... it's "free"

Of course we might not throw anything out.

We know how to find a bridge in linear time

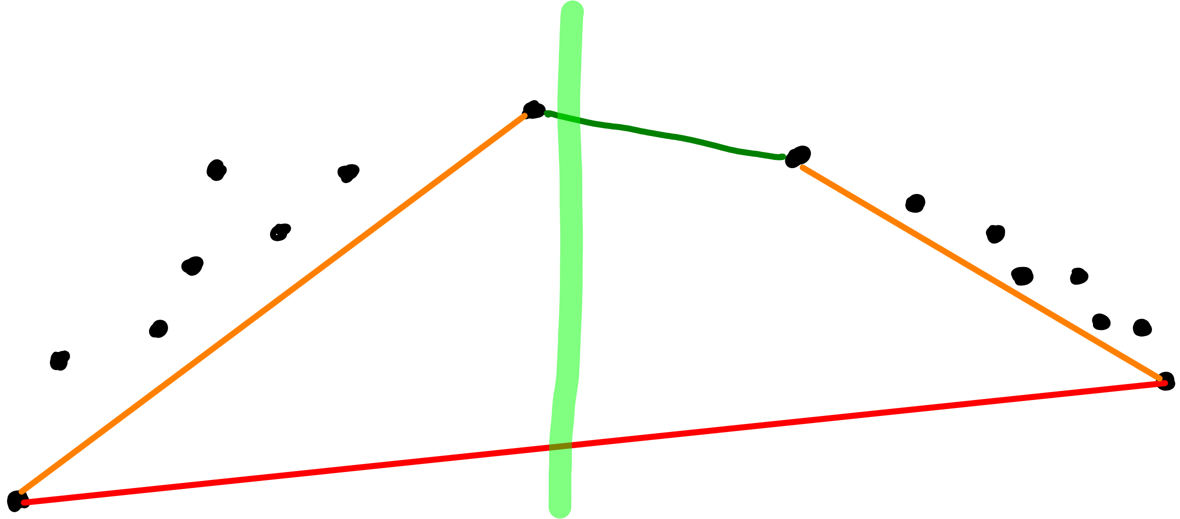


Solve 2 smaller problems with ~half points each.

That still only gives us $O(n \log n)$

Do we have to find a bridge that "splits" the hull evenly?

We know how to find a bridge in linear time



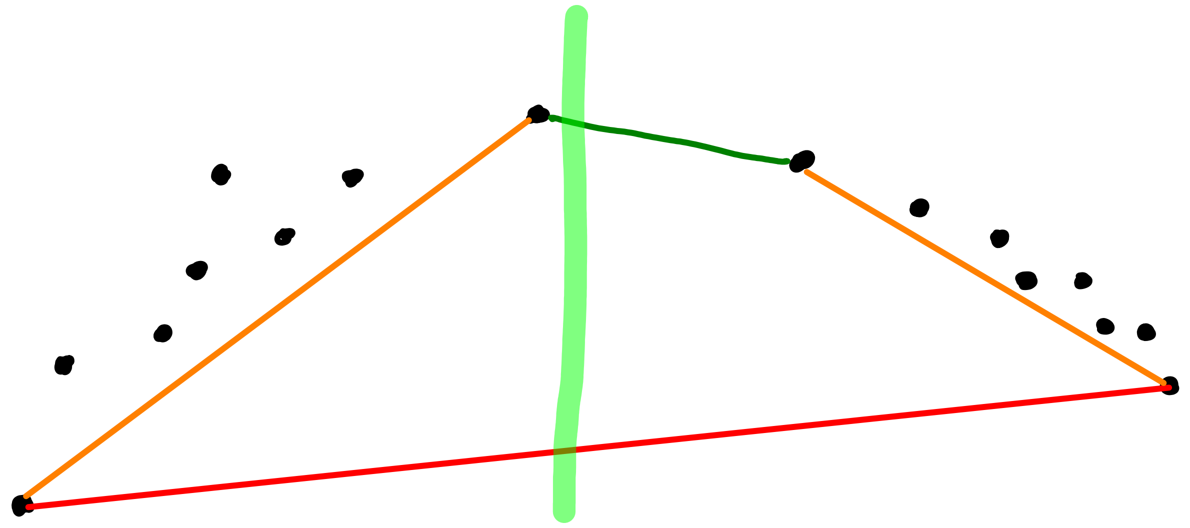
Solve 2 smaller problems with ~half points each.

That still only gives us $O(n \log n)$

Do we have to find a bridge that "splits" the hull evenly?

If we at least find one new bridge on both sides then we get $O(\log h)$ depth

We know how to find a bridge in linear time



Solve 2 smaller problems with ~half points each.

That still only gives us $O(n \log n)$

Do we have to find a bridge that "splits" the hull evenly?

If we at least find one new bridge on both sides then we get $O(\log h)$ depth

If we don't find a bridge on one side, we must have thrown out $n/2$ pts.