Announcements

- Homework 3 online, due Friday
- Exam 1 solutions online

HW2 Q2

Remember:

- Looking for sink/source SCCs
- Compute Metagraph
  - SCCs are a vertex v and *every* other vertex that can both reach and be reached from v
- Find sources/sinks
  - A sink is a vertex with no outgoing edges
  - A source is a vertex with no incoming edges
  - An isolated vertex is *both*!

Last Time

- Divide and Conquer
- Master Theorem
- MergeSort
- Order statistics

Divide and Conquer

This is the first of our three major algorithmic techniques.

1. Break problem into pieces
2. Solve pieces recursively
3. Recombine pieces to get answer

Master Theorem

<u>Theorem:</u> Let T(n) be given by the recurrence:

$$T(n) = \begin{cases} O(1) & \text{if } n = O(1) \\ aT(n/b + O(1)) + O(n^d) & \text{otherwise} \end{cases}$$

Then we have that

$$T(n) = \begin{cases} O(n^{\log_b(a)}) & \text{if } a > b^d \\ O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \end{cases}$$

Today

- Order statistics
- Binary search
- Closest pair of points

Order Statistics

Suppose that we just want to find the median element of a list, or the largest, or the 10^{th} smallest.

Order Statistics

Suppose that we just want to find the median element of a list, or the largest, or the 10^{th} smallest.

<u>Problem:</u> Given a list L of numbers and an integer k, find the kth largest element of L.

Order Statistics

Suppose that we just want to find the median element of a list, or the largest, or the 10^{th} smallest.

<u>Problem:</u> Given a list L of numbers and an integer k, find the kth largest element of L.

<u>Naïve Algorithm:</u> Sort L and return kth largest. O(n log(n)) time.

Divide Step

Select a *pivot* x ∈ L. Compare it to the other elements of L.

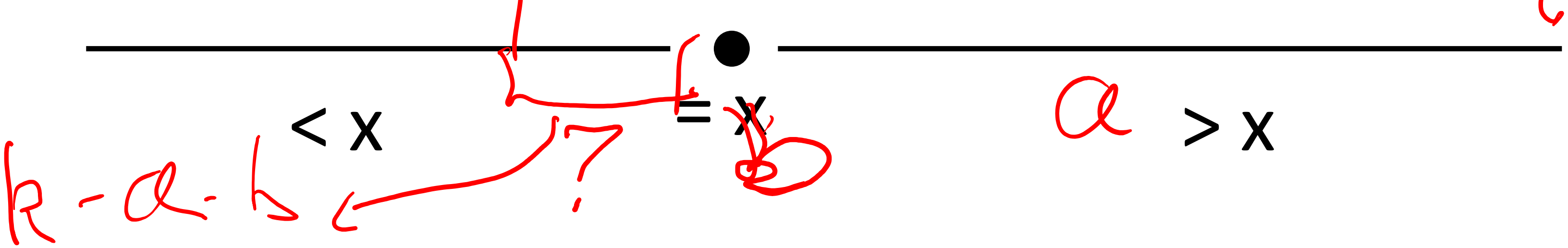
k

? b a

k - a - b

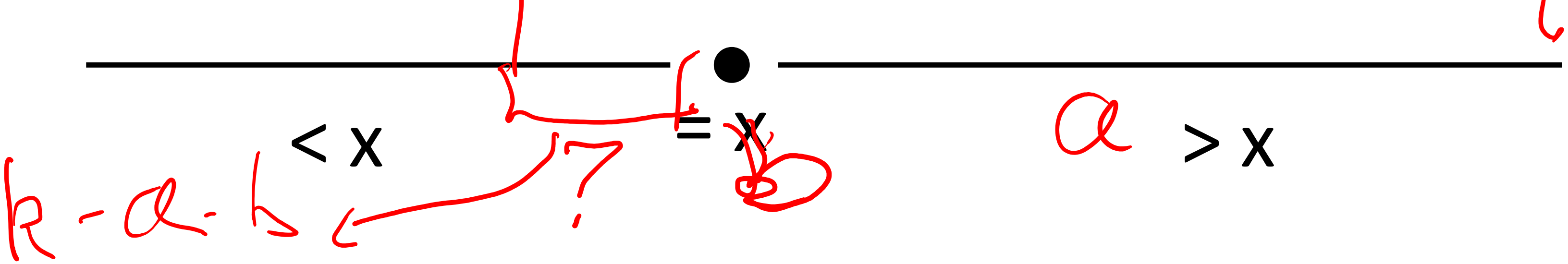
Divide Step

Select a *pivot* x ∈ L. Compare it to the other elements of L.



Divide Step

Select a *pivot* x ∈ L. Compare it to the other elements of L.

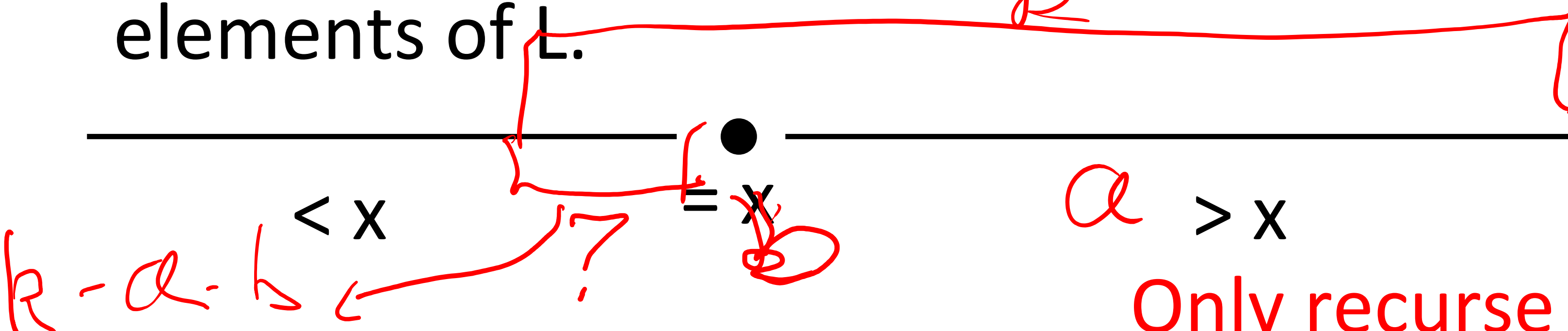


Which list is our answer in?

- Answer is > x if there are ≥ k elements bigger than x.
- Answer is x if there are < k elements bigger and ≥ k elements bigger than or equal to x.
- Otherwise answer is less than x.

Divide Step

Select a *pivot* x ∈ L. Compare it to the other elements of L.



Only recurse on one list

Which list is our answer in?

- Answer is > x if there are ≥ k elements bigger than x.
- Answer is x if there are < k elements bigger and ≥ k elements bigger than or equal to x.
- Otherwise answer is less than x.

Order Statistics

```
Select(L,k)
  Pick x ∈ L
  Sort L into L>x, L<x, L=x
  If Len(L>x) ≥ k
    Return Select(L>x,k)
  Else if Len(L>x)+Len(L=x) ≥ k
    Return x
  Return
    Select(L<x,k-Len(L>x)-Len(L=x))
```

Order Statistics

```
Select(L,k)
  Pick x ∈ L
  Sort L into L>x, L<x, L=x
  If Len(L>x) ≥ k
    Return Select(L>x,k)
  Else if Len(L>x)+Len(L=x) ≥ k
    Return x
  Return
   Select(L<x,k-Len(L>x)-Len(L=x))
```

O(n)

Order Statistics

```
Select(L,k)
  Pick x ∈ L
  Sort L into L>x, L<x, L=x      } O(n)
  If Len(L>x) ≥ k
    Return Select(L>x,k)         } ???
  Else if Len(L>x)+Len(L=x) ≥ k
    Return x
  Return
    Select(L<x,k-Len(L>x)-Len(L=x))   } ???
```

Runtime

Runtime recurrence

$$T(n) = O(n) + T(\text{sublist size})$$

Runtime

Runtime recurrence

$$T(n) = O(n) + T(\text{sublist size})$$

Problem: The sublist we recurse on could have size as big as n-1. If so, runtime is $O(n^2)$.

Runtime

Runtime recurrence

$$T(n) = O(n) + T(\text{sublist size})$$

Problem: The sublist we recurse on could have size as big as n-1. If so, runtime is $O(n^2)$.

Need to ensure this doesn't happen.

Randomization

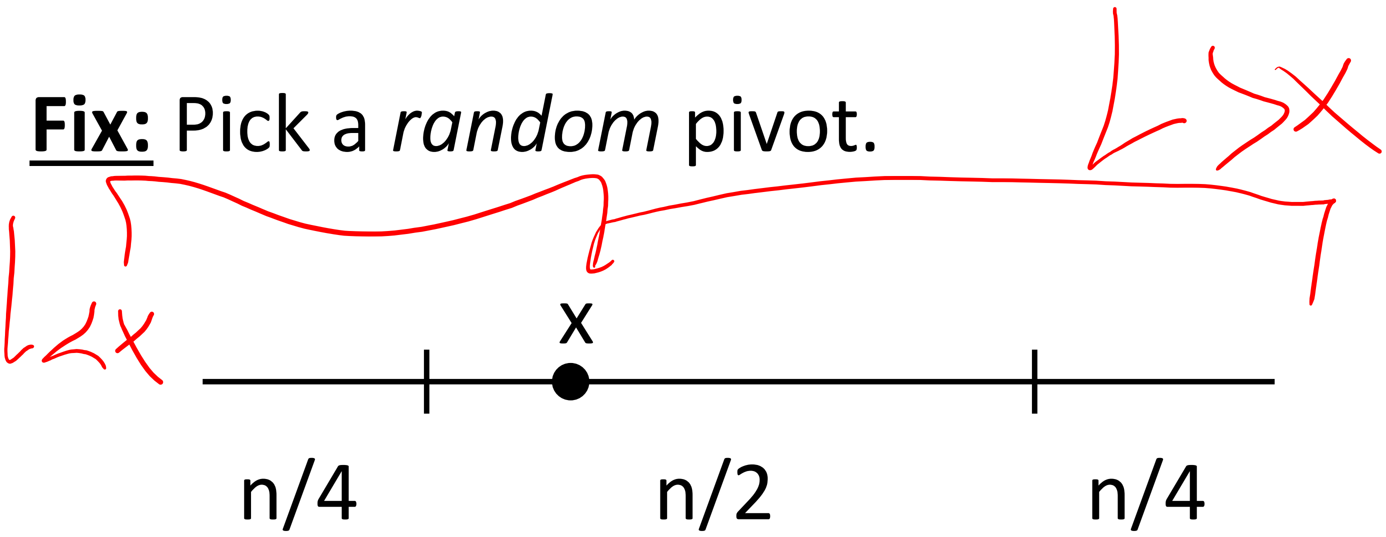
$> x$

Fix: Pick a *random* pivot.

$< x$

Randomization

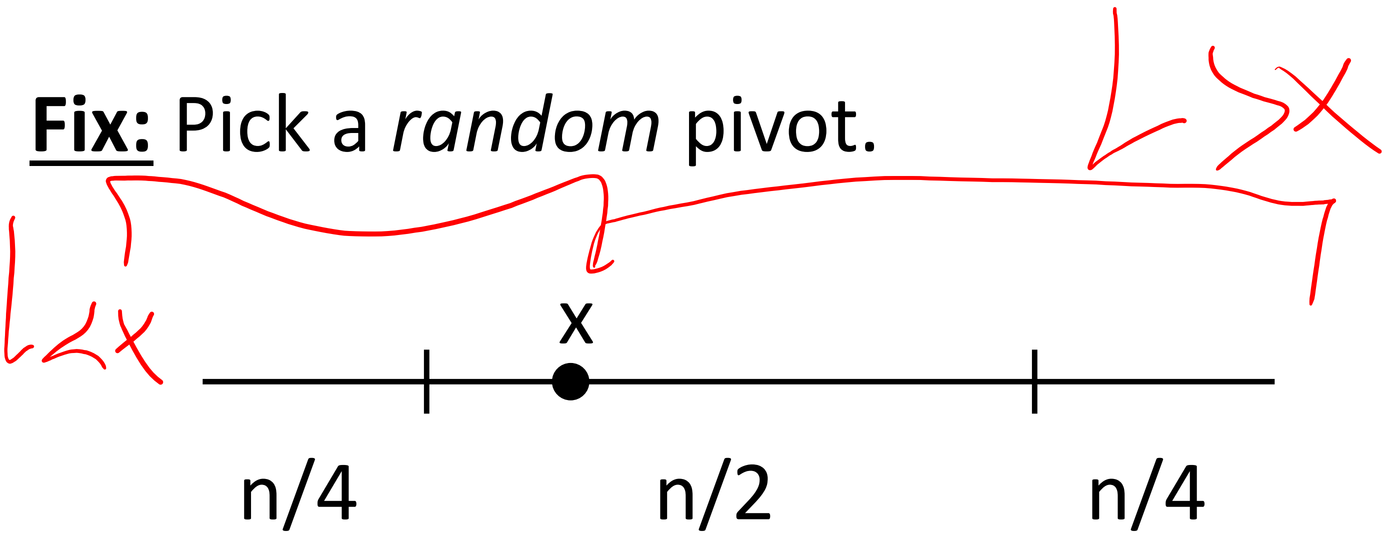
Fix: Pick a *random* pivot.



- There's a 50% chance that x is selected in the middle half.

Randomization

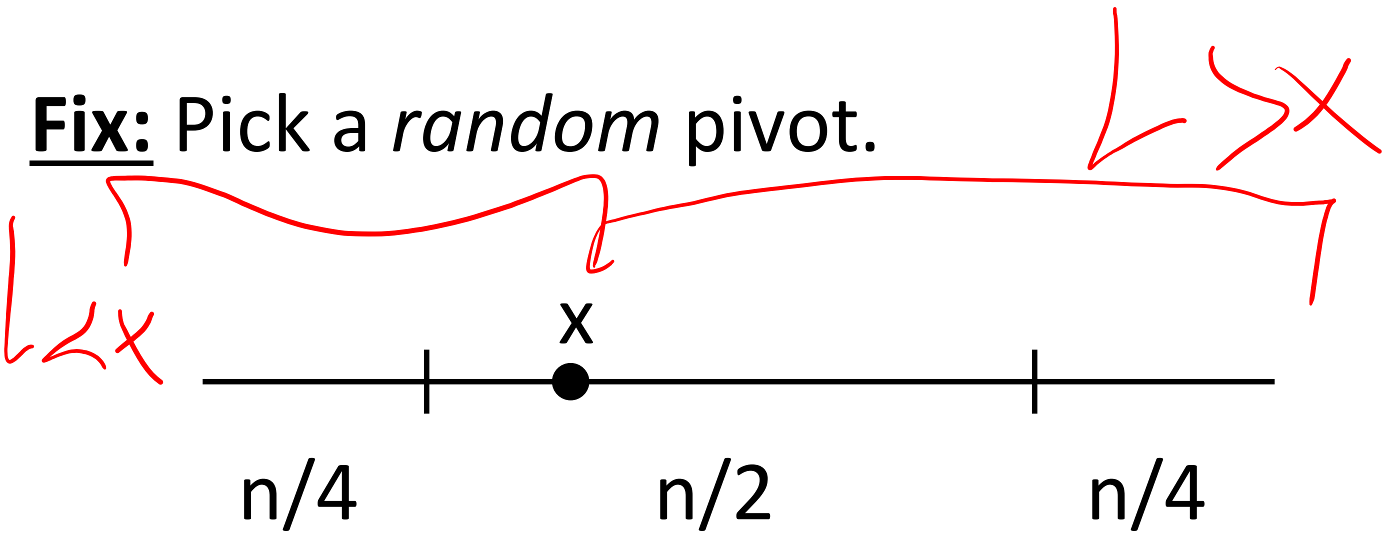
Fix: Pick a *random* pivot.



- There's a 50% chance that x is selected in the middle half.
- If so, no matter where the answer is, recursive call of size at most 3n/4.

Randomization

Fix: Pick a *random* pivot.



- There’s a 50% chance that x is selected in the middle half.
- If so, no matter where the answer is, recursive call of size at most 3n/4.
- On average need two tries to reduce call.

Question: Runtime

We get a runtime recurrence:

$T(n) = O(n) + T(3n/4)$

What is $T(n)$?

A) $O(\log(n))$
B) $O(n^{3/4})$
C) $O(n)$
D) $O(n \log(n))$
E) $O(n^2)$

Question: Runtime

We get a runtime recurrence:

$T(n) = O(n) + T(3n/4)$

What is T(n)?

A) $O(\log(n))$
B) $O(n^{3/4})$
C) $O(n)$
D) $O(n \log(n))$
E) $O(n^2)$

<u>Master Theorem:</u>

$a = 1$, $b = 4/3$, $d = 1$

$a < b^d$

$O(n^d) = O(n)$

Note

The algorithm discussed does give the correct answer in *expected* O(n) time.

There are deterministic O(n) algorithms using similar ideas, but they are substantially more complicated.

Search

Problem: Given a sorted list L and a number x, find the location of x in L.

Search

<u>Problem:</u> Given a sorted list L and a number x, find the location of x in L.

<u>Naïve Algorithm:</u> Try every element of L. O(n) time.

Search

Problem: Given a sorted list L and a number x, find the location of x in L.

Naïve Algorithm: Try every element of L. O(n) time.

Usually, you cannot beat O(n) because any algorithm needs to read the entire input. However, since the list is guaranteed to be sorted, we can do better here.

Divide

Split L into two lists.

Divide

Split L into two lists.

- Could search for x in each
  $T(n) = 2T(n/2)+O(1)$ → Too slow

Divide

Split L into two lists.

- Could search for x in each
  $T(n) = 2T(n/2)+O(1)$ → Too slow
- Use sorting to figure out which list to check.

Divide

Split L into two lists.

- Could search for x in each
  $T(n) = 2T(n/2)+O(1)$ → Too slow
- Use sorting to figure out which list to check.

If $L[i] > x$, location must be before i.

If $L[i] < x$, location must be after i.

If $L[i] = x$, we found it.

Binary Search

```
BinarySearch(L,i,j,x)
\\Search between L[i] and L[j]
  If j < i, Return 'error'
  k ← [(i+j)/2]
  If L[k] = x, Return k
  If L[k] > x
    Return BinarySearch(L,i,k-1,x)
  If L[k] < x
    Return BinarySearch(L,k+1,j,x)
```

Binary Search

```
BinarySearch(L,i,j,x)
\\Search between L[i] and L[j]
  If j < i, Return ‘error’
  k ← [(i+j)/2]
  If L[k] = x, Return k
  If L[k] > x
    Return BinarySearch(L,i,k-1,x)
  If L[k] < x
    Return BinarySearch(L,k+1,j,x)
```

O(1)

Binary Search

```
BinarySearch(L,i,j,x)
\\Search between L[i] and L[j]
  If j < i, Return 'error'
  k ← [(i+j)/2]
  If L[k] = x, Return k
  If L[k] > x
    Return BinarySearch(L,i,k-1,x)
  If L[k] < x
    Return BinarySearch(L,k+1,j,x)
```

O(1)

T(n/2)

Question: Runtime

We get a runtime recurrence:

$T(n) = O(1) + T(n/2)$

What is $T(n)$?

A) $O(\log(n))$
B) $O(n^{1/2})$
C) $O(n^{1/2} \log(n))$
D) $O(n)$
E) $O(n \log(n))$

Question: Runtime

We get a runtime recurrence:

$T(n) = O(1) + T(n/2)$

What is $T(n)$?

A) $O(\log(n))$
B) $O(n^{1/2})$
C) $O(n^{1/2} \log(n))$
D) $O(n)$
E) $O(n \log(n))$

<u>Master Theorem:</u>

$a = 1, b = 2, d = 0$

$a = b^d$

$O(n^d \log(n)) = O(\log(n))$

Binary Search Puzzles

You have 27 coins one of which is heavier than the others, and a balance. Determine the heavy coin in 3 weightings.

Lots of puzzles have binary search-like answers. As long as you can spend constant time to divide your search space in half (or thirds). You can use binary search in O(log(n)) time.

Closest Pair of Points (Ex 2.32)

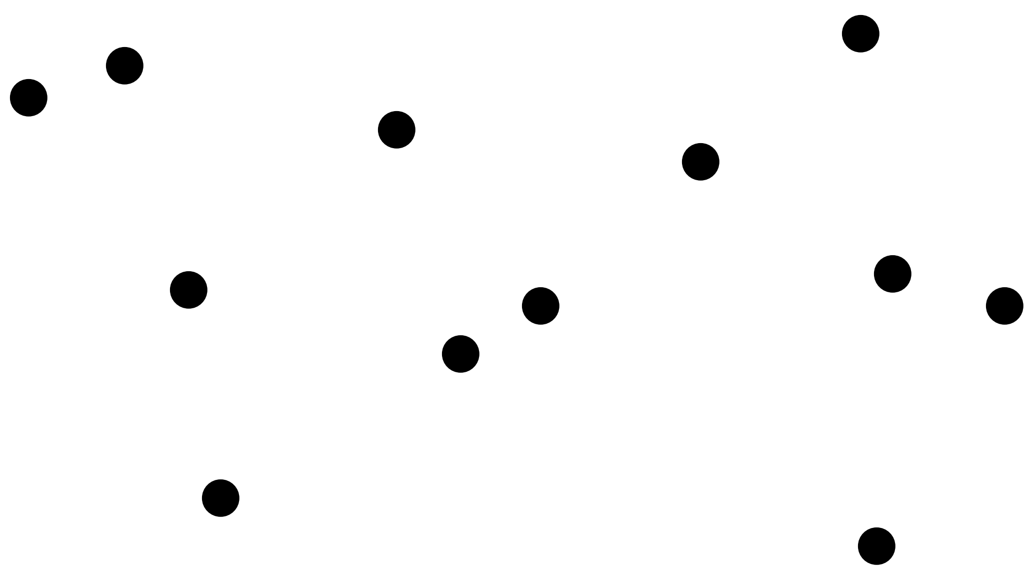
Problem: Given n points in the plane $(x_1,y_1)\ldots(x_n,y_n)$ find the pair (x_i,y_i) and (x_j,y_j) whose Euclidean distance is as small as possible.

Closest Pair of Points (Ex 2.32)

Problem: Given n points in the plane $(x_1,y_1)\ldots(x_n,y_n)$ find the pair (x_i,y_i) and (x_j,y_j) whose Euclidean distance is as small as possible.

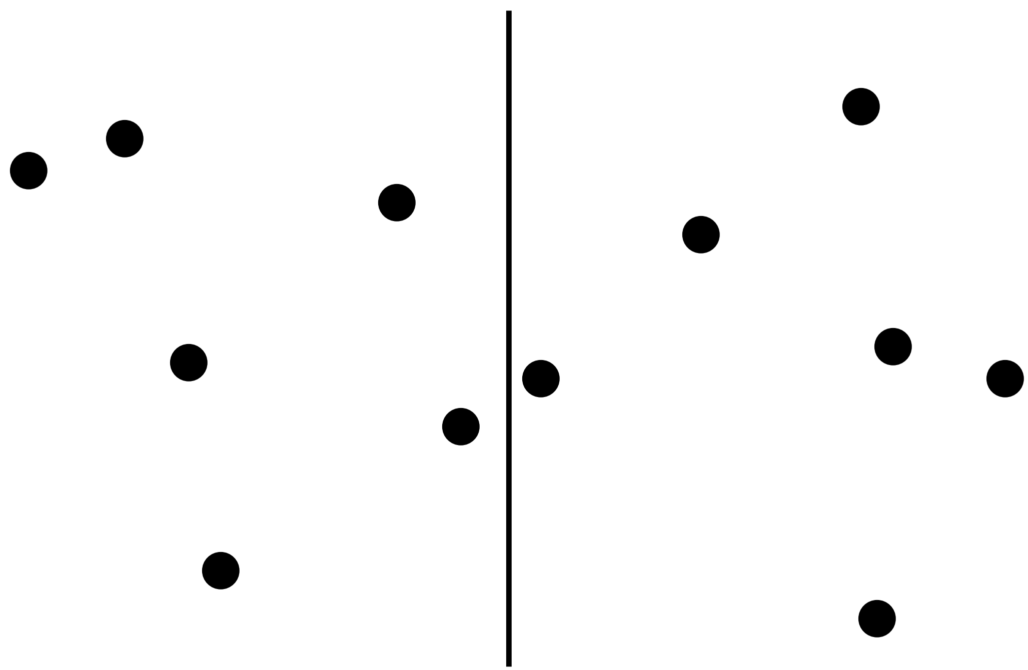
Naïve Algorithm: Try every pair of points. $O(n^2)$ time.

Divide and Conquer Outline



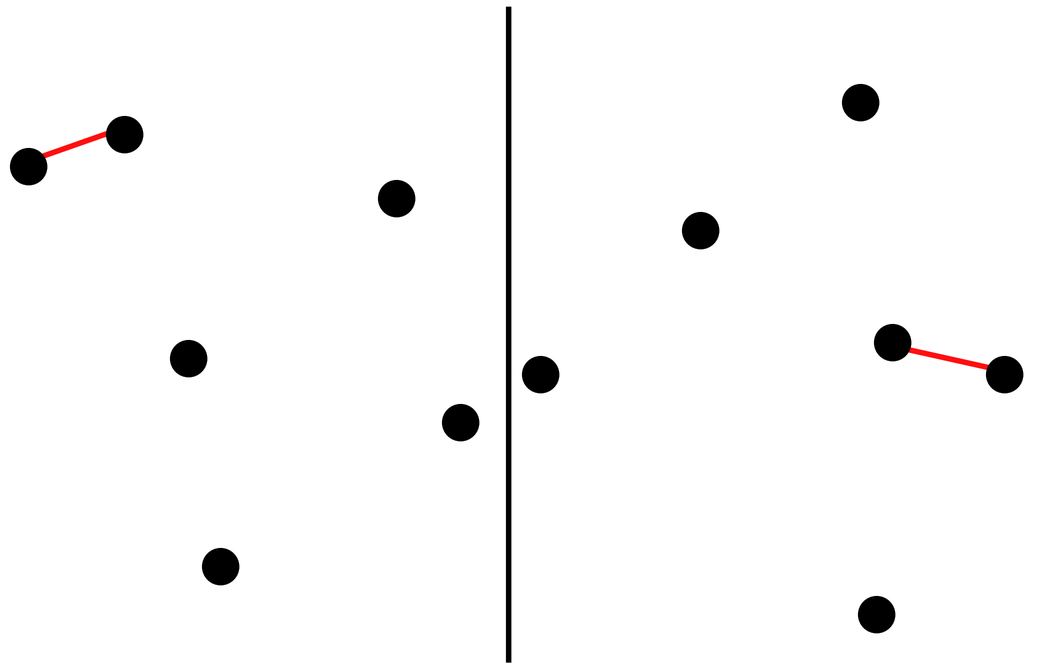
Divide and Conquer Outline

- Divide points into two sets by drawing a line.



Divide and Conquer Outline

- Divide points into two sets by drawing a line.
- Compute closest pair on each side.



Divide and Conquer Outline

- Divide points into two sets by drawing a line.
- Compute closest pair on each side.
- What about pairs that cross the divide?