Announcements

- Homework 3 Due on Friday
- Please fill out feedback survey: https://forms.gle/Rs1Jz4XZpaQqag13A

Last Time

- Divide & Conquer
  - Split into smaller subproblems
  - Solve Recursively
  - Recombine to get answer
- Sorting
- Order Statistics

Today

- Binary Search
- Closest Pair of Points

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.

- 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)
```

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) $O(n \log(n))$

Master Theorem:

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

$a = b^d$

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

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.

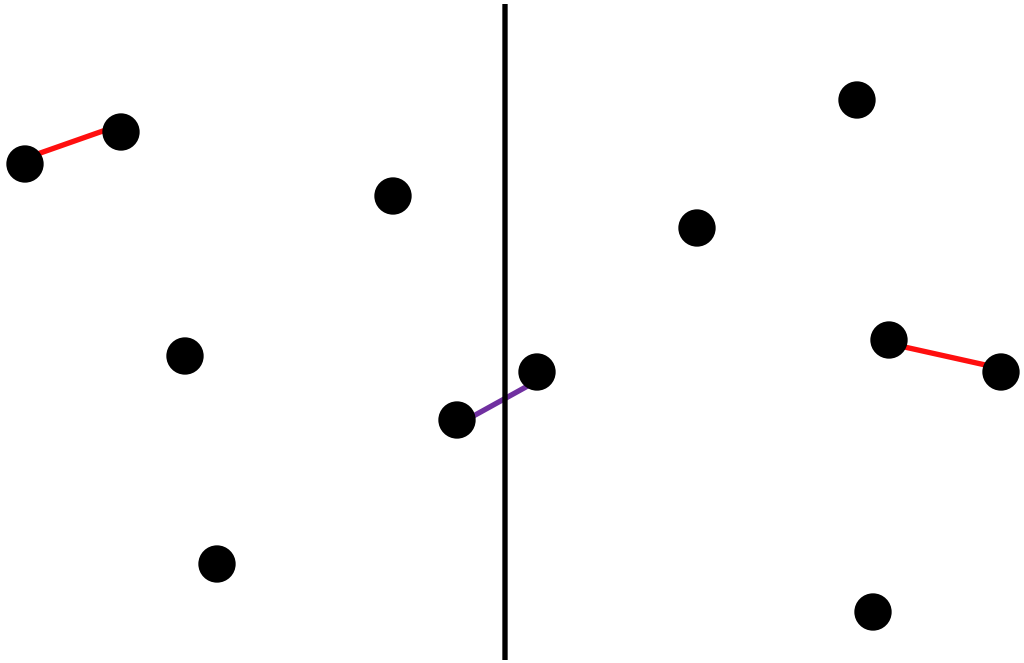
Closest Pair of Points (Ex 2.32)

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.

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

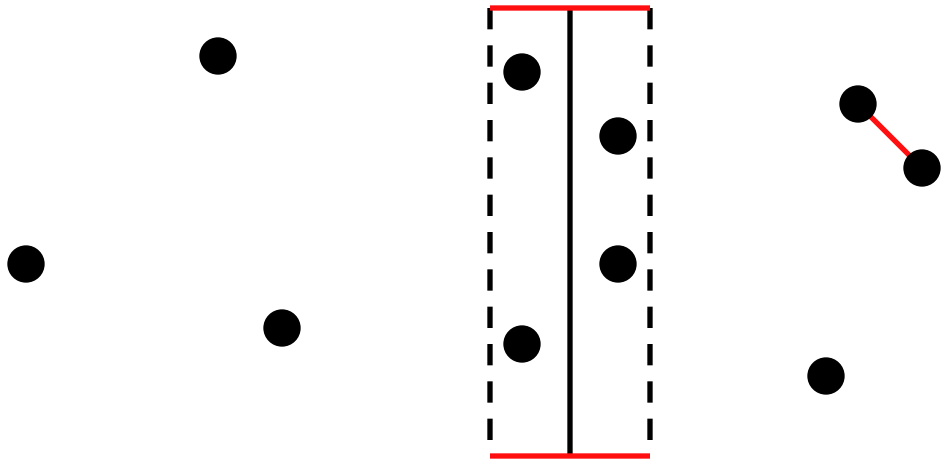
Divide and Conquer Outline

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



Observation

- Suppose closest pair on either side at distance δ.
- Only need to care about points within δ of dividing line.
- Need to know if some pair closer than δ.



Main Idea

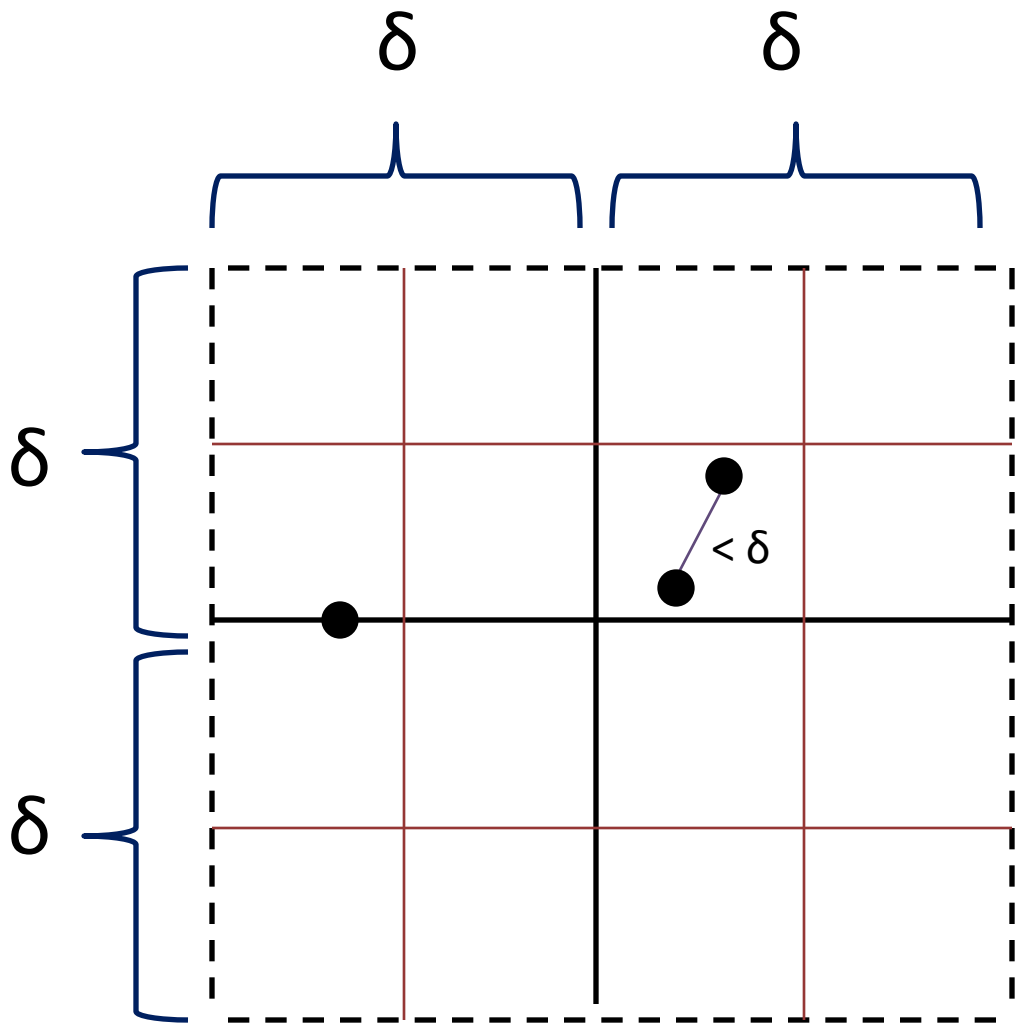
Proposition: Take the points within δ of the dividing line and sort them by y-coordinate. Any one of these points can only be within δ of the 8 closest points on either side of it.

This means that we only need to check a few pairs of points crossing the line.

Idea: Points on each side separated by δ. Not enough room for many of them nearby.

Proof

- Nearby points must have y-coordinate within δ.
- Subdivide region into δ/2-sided squares.
- At most one point in each square.



Algorithm

```
CPP(S)
  If |S| ≤ 3
    Return closest distance
  Find line L evenly dividing points
  Sort S into S_left, S_right
  δ ← min(CPP(S_left),CPP(S_right))
  Let T be points within δ of L
  Sort T by y-coordinate
  Compare each element of T to 8 closest
    on either side. Let min dist be δ'.
  Return min(δ,δ')
```

O(n) (median of x-coordinates)

O(n)

2T(n/2)

O(n log(n))

O(n)

Runtime

We have a recurrence

$$T(n) = O(n \log(n)) + 2\,T(n/2).$$

This is not quite covered by the Master Theorem, but can be shown to give

$$T(n) = O(n \log^2(n)).$$

Alternatively, if you are more careful and have `CPP` take points already sorted by y-coordinate, you can reduce to $O(n \log(n))$.

Fast Fourier Transform

- We will *not* cover this in class.
- It is a great example of a divide and conquer algorithm.
- Very important algorithm. Allows you to:
  - Multiply n-bit numbers in $O(n \log^2(n))$ time.
  - Decompose a signal into frequencies.
  - Remove noise from signals.
- It is *highly* recommended that you look it up in the book if you:
  - Have familiarity with complex numbers
  - Already know what a Fourier transform is
  - Are motivated to put in some extra time