Recursion/Divide and Conquer

Majority Element

An array $A[1, n]$ is said to have a majority element iff strictly more than half of its entries are the same.

Problem: Given an array of elements, design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element.

Example: 5 6 7 1 5 5 5 8 5 5 1 1 5 5 5 (Answer: 5)

Wrinkle: The elements of the array are not necessarily from some ordered domain like the integers. There can be no comparisons of the form "Is $A[i] > A[j]$?". However you can answer questions of the form: "Is $A[i] = A[j]$?" in constant time.

Majority Element Preliminaries

Monkey: $O(n^2)$

Lower bound: $\Omega(n)$

Applications: Data compression (Store the file once and keep a record of the locations where it appears.)

Majority Element: Analysis

Claim: If we split an array A with a majority element m into two parts A_1 and A_2, then m must be a majority element in either A_1 or A_2 or both.

Example: 5 6 7 1 5 5 5 8 5 5 1 1 5 5 5

Proof: Let there be x_1 m elements in A_1 and x_2 m elements in A_2.

$$2x_1 \leq |A_1| \text{ and } 2x_2 \leq |A_2| \Rightarrow 2(x_1+x_2) \leq |A_1|+|A_2| = |A|$$

Majority Element: Algorithm

$MAJORITY(A)$

- If $|A| = 1$, return A.
- Split A into two (roughly) equal halves, A_1 and A_2.
- Let $m_1 \equiv MAJORITY(A_1), m_2 \equiv MAJORITY(A_2)$.
- Check whether m_1 or m_2 is a majority element for A. If so, return it. Otherwise, return $\emptyset$.

Example: 5 6 7 1 5 5 5 8 5 5 1 1 5 5 5

Majority Element: Time Analysis

$MAJORITY(A)$

- If $|A| = 1$, return A.
- Split A into two (roughly) equal halves, A_1 and A_2.
- Let $m_1 \equiv MAJORITY(A_1), m_2 \equiv MAJORITY(A_2)$.
- Check whether m_1 or m_2 is a majority element for A. If so, return it. Otherwise, return $\emptyset$.

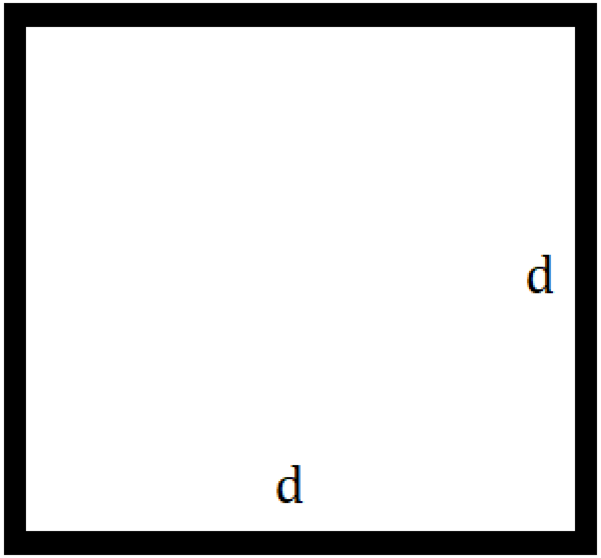
Let $T(n)$ be the time for $MAJORITY(A)$ if $|A| = n$.

$$T(n) = 2T(\frac{n}{2}) + O(n) \Rightarrow T(n) = \Theta(n \log n)$$

Computational Geometry

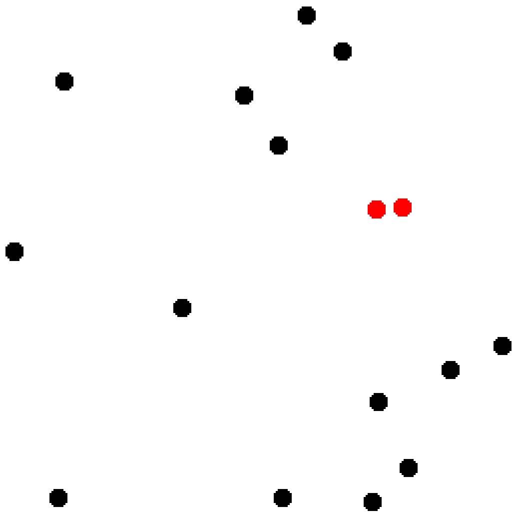
The field of computational geometry studies algorithmic solutions for problems in the field of geometry.

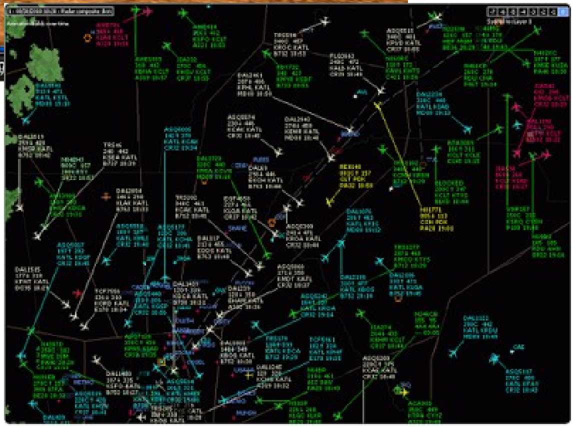
<u>Fact</u>: If all points are required to be at least distance d apart, then the maximum number of points that can fit into a square with dimensions $d \times d$ is 4.



Closest Point Pair

Problem: You are given a set of n points in the plane and you want to determine the closest pair of points.





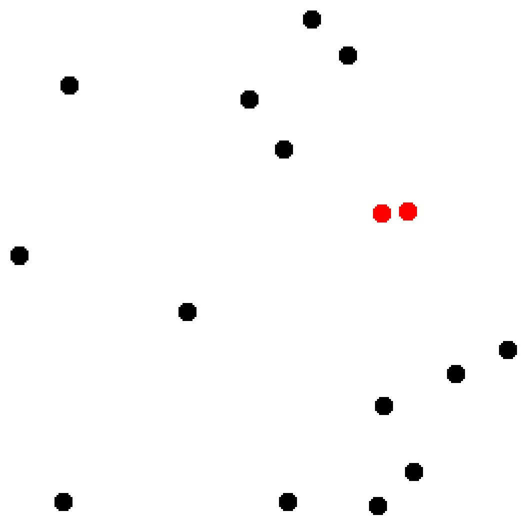
Monkey: $O(n^2)$

Lower bound: $\Omega(n)$

Applications: air-traffic controller, WoW

Closest Point Pair: Preliminary

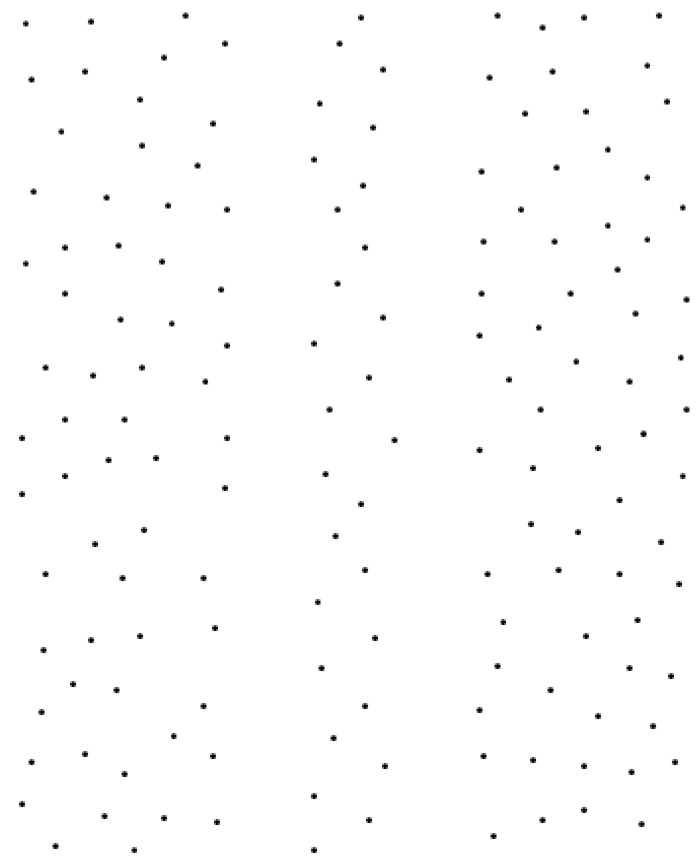
Before the algorithm starts, sort all points on both x and y coordinate. From this point onward, we can evaluate **above, below, left, in the middle**, etc. in time $O(1)$.



Closest Point Problem: Algorithm

$CPP(S)$

- If $|S| \leq 3$, brute force and return the answer.
- Divide S into two roughly equal-sized sets $LEFT$ and $RIGHT$.
- $CPP(LEFT) \equiv \{p_1, p_2\}$
- $CPP(RIGHT) \equiv \{q_1, q_2\}$
- $d \equiv \min\{d(p_1, p_2), d(q_1, q_2)\}$.
- Slide the window up to check for closer left/right pairs.

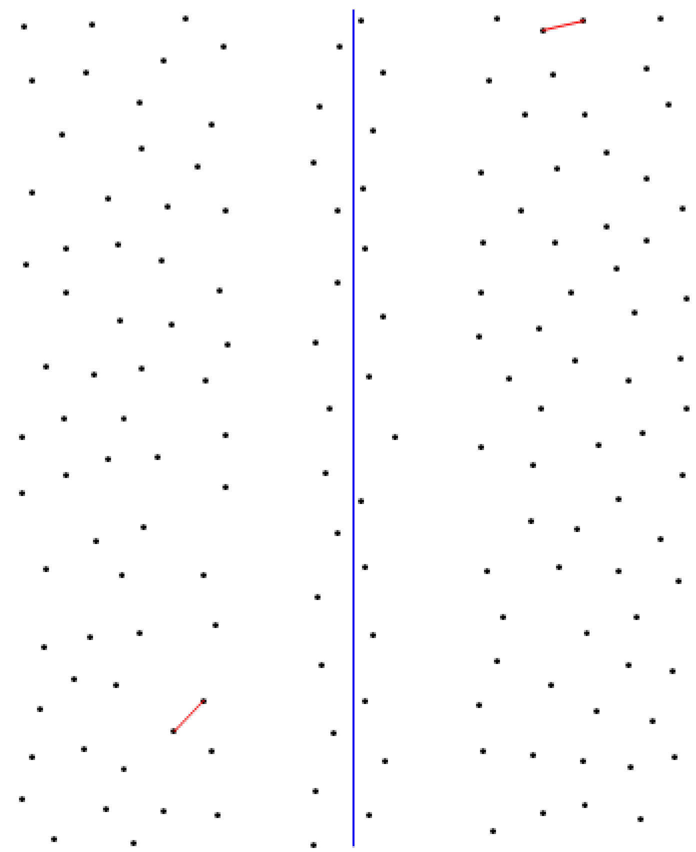


$\sim$ 100 points distributed inside the unit square.
Bottom left: (0,0) Upper right: (1,1)

Closest Point Problem: Algorithm

$CPP(S)$

- If $|S| \leq 3$, brute force and return the answer.
- Divide S into two roughly equal-sized sets $LEFT$ and $RIGHT$.
- $CPP(LEFT) \equiv \{p_1, p_2\}$
- $CPP(RIGHT) \equiv \{q_1, q_2\}$
- $d \equiv \min\{d(p_1, p_2), d(q_1, q_2)\}$.
- Slide the window up to check for closer left/right pairs.

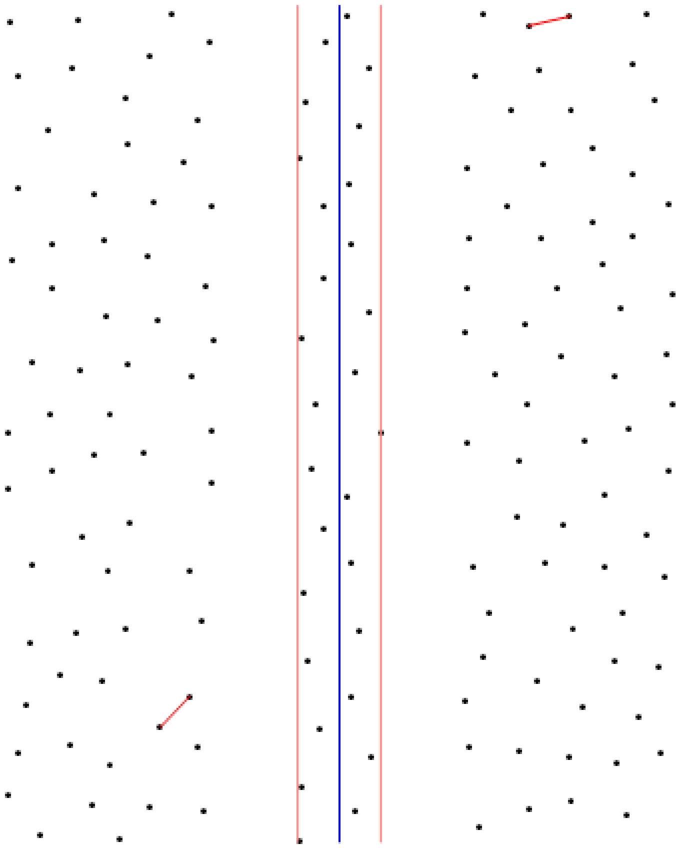


$$d((0.320515, 0.173867), (0.285685, 0.137414)) = 0.0504182$$
$$d((0.727689, 0.975927), (0.776453, 0.987135)) = 0.0500347$$

Closest Point Problem: Algorithm

$CPP(S)$

- If $|S| \leq 3$, brute force and return the answer.
- Divide S into two roughly equal-sized sets $LEFT$ and $RIGHT$.
- $CPP(LEFT) \equiv \{p_1, p_2\}$
- $CPP(RIGHT) \equiv \{q_1, q_2\}$
- $d \equiv \min\{d(p_1, p_2), d(q_1, q_2)\}$.
- Slide the window up to check for closer left/right pairs.



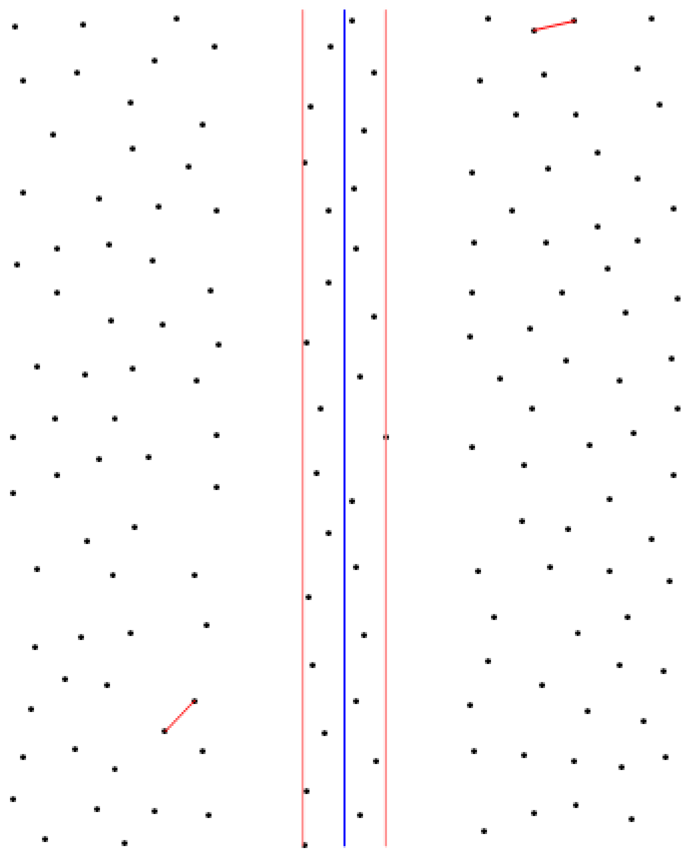
$d = 0.0500347$

Separators are at a distance d from the y-axis.

Closest Point Problem: Algorithm

$CPP(S)$

- If $|S| \leq 3$, brute force and return the answer.
- Divide S into two roughly equal-sized sets $LEFT$ and $RIGHT$.
- $CPP(LEFT) \equiv \{p_1, p_2\}$
- $CPP(RIGHT) \equiv \{q_1, q_2\}$
- $d \equiv \min\{d(p_1, p_2), d(q_1, q_2)\}$.
- Slide the window up to check for closer left/right pairs.

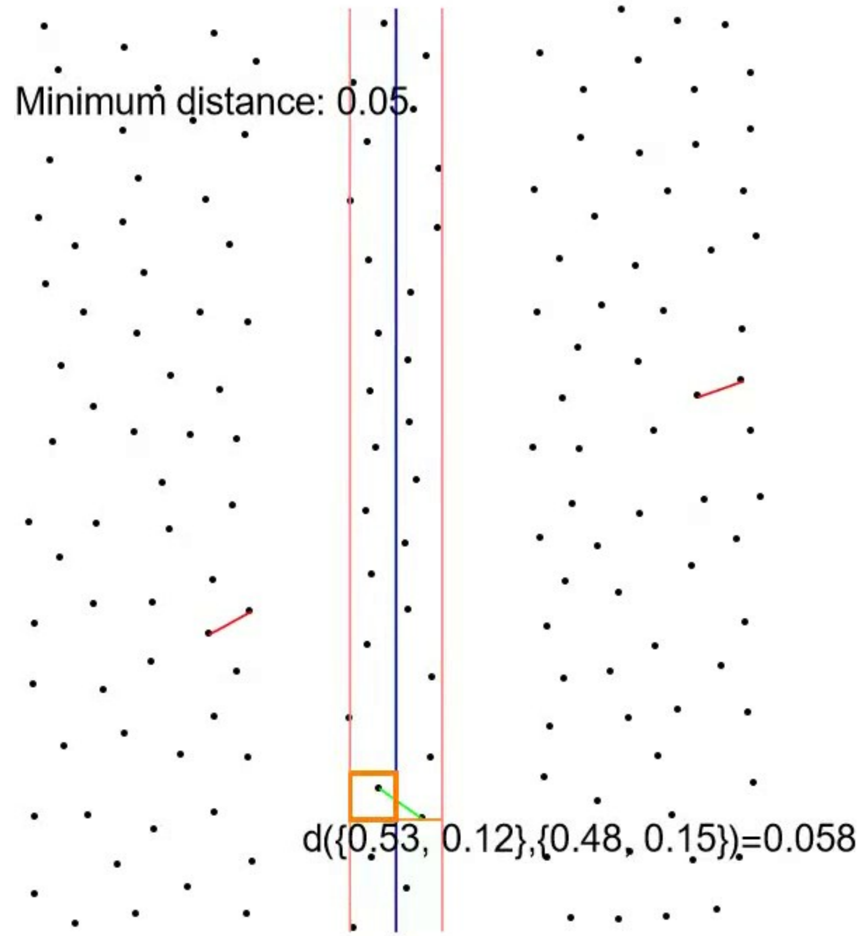


If there is going to be a pair of points closer together than d, where must they be?

Closest Point Problem: Algorithm

$CPP(S)$

- If $|S| \leq 3$, brute force and return the answer.
- Divide S into two roughly equal-sized sets $LEFT$ and $RIGHT$.
- $CPP(LEFT) \equiv \{p_1, p_2\}$
- $CPP(RIGHT) \equiv \{q_1, q_2\}$
- $d \equiv \min\{d(p_1, p_2), d(q_1, q_2)\}$.
- Slide the window up to check for closer left/right pairs.



The window is of size $d \times d$. At most how many points can be in that window?

Closest Point Problem: Time Analysis

$CPP(S)$

- If $|S| \leq 3$, brute force and return the answer.
- Divide S into two roughly equal-sized sets $LEFT$ and $RIGHT$.
- $CPP(LEFT) \equiv \{p_1, p_2\}$
- $CPP(RIGHT) \equiv \{q_1, q_2\}$
- $d \equiv \min\{d(p_1, p_2), d(q_1, q_2)\}$.
- Slide the window up to check for closer left/right pairs.

Let $T(n)$ be the time CPP takes if $|S| = n$.

$T(n) = 2T(\frac{n}{2}) + O(n) \Rightarrow$

$T(n) = \Theta(n \log n)$

Majority Element Revisited

Example: 6 5 7 1 5 5 5 8 5 5 1 1 5 5 5

Operation: If the list has an odd number of elements, choose the first element and check whether it is a majority element. If so, you're done. If not, throw it away. Now the list must have an even number of elements: Split the array into pairs of 2 elements. For each pair, if they are the same element, keep one of them; otherwise, throw away both.

$\Rightarrow$ 5 7 1 5 5 5 8 5 5 1 1 5 5 5

$\Rightarrow$ 5 7 | 1 5 | 5 5 | 8 5 | 5 1 | 1 5 | 5 5

$\Rightarrow$ 5 5

$\Rightarrow$ 5

Majority Element Revisited

Example: 6 5 7 1 5 5 5 8 5 5 1 1 5 5 5

$\Rightarrow$5 7 1 5 5 5 8 5 5 1 1 5 5 5

$\Rightarrow$5 7 | 1 5 | 5 5 | 8 5 | 5 1 | 1 5 | 5 5

$\Rightarrow$5 5

$\Rightarrow$5

If there is a majority element, will it remain a majority element after this operation?

Assume there are x elements out of n that are all the same and that $\frac{x}{n} > \frac{1}{2}$. The worst case would be if many majority elements matched up with nonmajority elements. Assume that this happens y times.

$\frac{x}{n} > \frac{1}{2} \Rightarrow 2x > n \Rightarrow 2x - 2y > n - 2y \Rightarrow \frac{x-y}{n-2y} > \frac{1}{2}$

But $\frac{x-y}{n-2y}$ is an upper bound on the fraction of majority elements remaining after the operation.

Majority Element Revisited

So if there was a majority element before the operation, it must remain a majority element after the operation. **If we repeat the operation and there is a majority element, it must survive until the very end!**

How long does the operation take on a list of size n?

$\Theta(n)$

At least what fraction of the list is removed after each operation?

$\frac{1}{2}$

How long does this tournament algorithm take?

$\Theta(n + \frac{n}{2} + \frac{n}{4} + \ldots) = \Theta(n)$