CS 218 Design and Analysis of Algorithms

Nutan Limaye

Indian Institute of Technology, Bombay
nutan@cse.iitb.ac.in

Module 1: Basics of algorithms

Divide, Delegate and Combine (Divide and Conquer)

You cannot do everything and be efficient!

Integer multiplication

Problem Description

Input:	Two n-digit non-negative integers x, y
Compute:	$x \times y$

We know that this has a simple algorithm (we studied in school).

What is the time complexity of that algorithm?

Primitive operations:

Adding two single digit numbers takes $O(1)$ time.

Multiplying two single digit numbers takes $O(1)$ time.

Inserting a zero at the end of a number takes $O(1)$ time.

We designed a $O(n^{\log 3})$ time algorithm for this.

Closest points in a plane

Given: n points, $p_1 = (x_1, y_1), p_2 = (x_2, y_2), \ldots, p_n = (x_n, y_n)$
Output: i, j such that the distance between p_i, p_j is the minimum

```
min ← (x_1 − x_2)^2 + (y_1 − y_2)^2
for i = 1 to n do
    for j = i + 1 to n do
        d ← (x_i − x_j)^2 + (y_i − y_j)^2
        if min > d then
            min ← d
        end if
    end for
end for
Output min
```

$O(n^2)$ comparisons. Can we do better?

On a journey to find an $O(n \log n)$ algorithm

Consider the one-dimensional case.

Here an $O(nlogn)$ algorithm seems easy.

Sort the points based on their co-ordinate.

The closest pair must be consecutive in this ordering.

Can this work for 2-D?

The divide and conquer approach

If we divide the points into two halves.

Find recursively the closest pair in one half.

Similarly, in the second half.

Using these answers, combine.

If we hope for $O(n \log n)$, then combination step must take $O(n)$ time.

Split across the middle, there are still $\Omega(n^2)$ distances which are not computed!

The divide and conquer approach

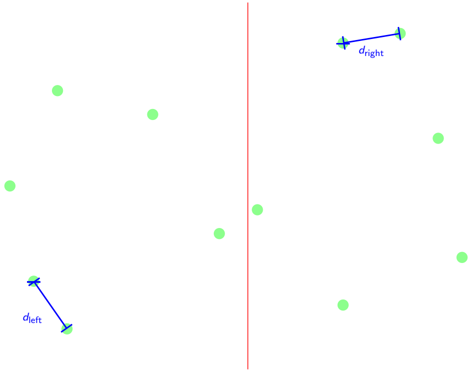
Given all the points in a plane.

The divide and conquer approach

Given all the points in a plane.

Divide them into 2 halves based on their x-coordinates.

The divide and conquer approach

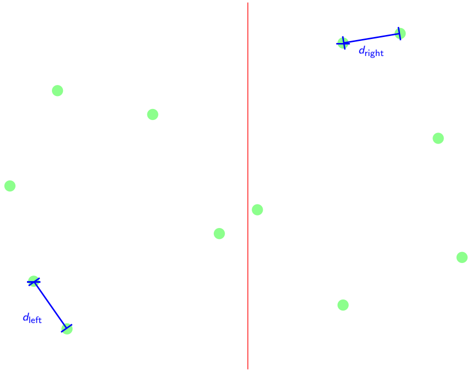


Given all the points in a plane.

Divide them into 2 halves based on their x-coordinates.

recursively compute d_{left} and d_{right}.

The divide and conquer approach



Given all the points in a plane.

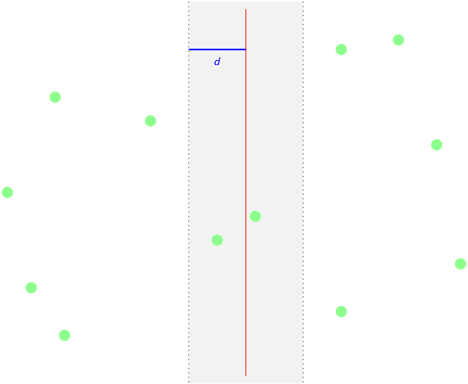
Divide them into 2 halves based on their x-coordinates.

recursively compute d_{left} and d_{right}.

Let $d = \min\{d_{\text{left}}, d_{\text{right}}\}$.

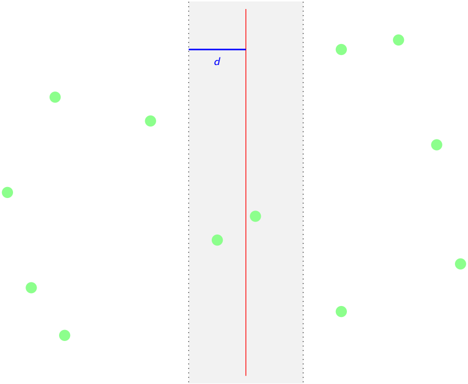
If the first division does not separate the closest pair, then d is the correct answer.

The divide and conquer approach



If the closest pair is separated by the red line

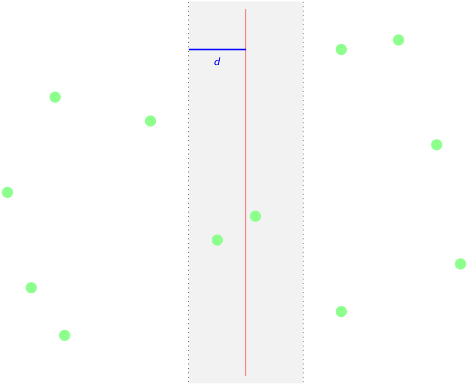
The divide and conquer approach



If the closest pair is separated by the red line

Both the points must lie within distance d from the the red line.

The divide and conquer approach

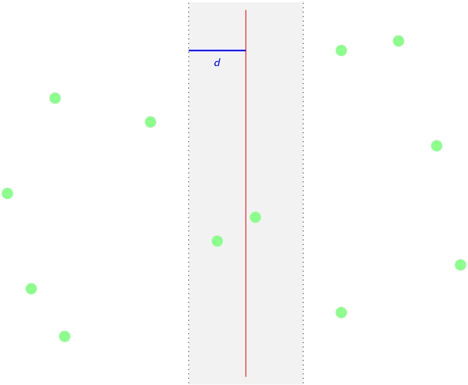


If the closest pair is separated by the red line

Both the points must lie within distance d from the the red line.

Why?

The divide and conquer approach



If the closest pair is separated by the red line

Both the points must lie within distance d from the the red line.

Why?

If one of these points is further away, then the distance between them will have to be $> d$.

Hence cannot be the closest pair.

Divide and conquer approach

Lemma

Let S_y be the points in the distance d region from the red line sorted in decreasing order of their y-coordinates. Say $S_y = \langle q_1, \ldots, q_m \rangle$. If the distance between some q_i and q_j is $< d$ then $j - i \leq 15$.

Points to be noted about the lemma.

How long does it take to compute S_y?

$O(n)$ time. Why?

Recall the 1-D problem.

There two closest points were consecutive.

Here not quite the same, but there is some resemblance.

Assuming the lemma, are we done?

Algorithm for finding the closest pairs

Given: n points, $p_1 = (x_1, y_1), p_2 = (x_2, y_2), \ldots, p_n = (x_n, y_n)$
Output: i, j such that the distance between p_i, p_j is the minimum

Let P_x be the array of points sorted in increasing of x-coordinates.
Let P_y be the array of points sorted in increasing of y-coordinates.
`ClosestPair`(P_x, P_y)

if $|P_x| = 2$ **then**
  Output the distance between points in P_x.
end if
$d_{left} \leftarrow$ `ClosestPair`(FirstHalf(P_x, P_y)).
$d_{right} \leftarrow$ `ClosestPair`(SecondHalf(P_x, P_y)).
$d = \min(d_{left}, d_{right})$.
Let S_y be the points in P_y within distance d from the red line.
for $i = 1$ to $|S_y|$ **do**
  for $j = 1$ to 15 **do**
    $d \leftarrow \min\{(\text{distance between } S_y(i), S_y(j)), d\}$.
  end for
end for
Output d.

Running time analysis of `ClosestPair`

Running time analysis

Sorting the points based on their x and y co-ordinates takes time $O(n \log n)$.

Computing the first or second half of P_x or P_y takes time $O(n)$.

Combining the answers by comparing with the middle band takes time $O(n)$.

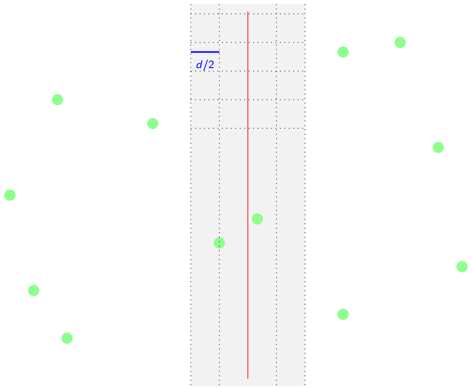
$$T(n) \leq 2 \cdot T(n/2) + O(n)$$

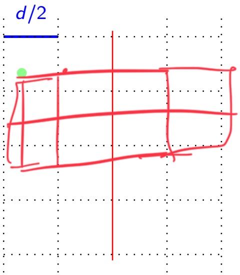
$T(n) = O(n \log n)$.

Divide and conquer approach

Lemma

Let S_y be the points in the distance d region from the red line sorted in decreasing order of their y-coordinates. Say $S_y = \langle q_1, \ldots, q_m \rangle$. If the distance between some q_i and q_j is $< d$ then $j - i \leq 15$.



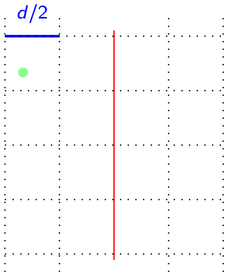


Divide and conquer approach

Lemma

Let S_y be the points in the distance d region from the red line sorted in decreasing order of their y-coordinates. Say $S_y = \langle q_1, \ldots, q_m \rangle$. If the distance between some q_i and q_j is $< d$ then $j - i \leq 15$.

Divide the region up into squares of size $d/2$.



Each square can contain at most 1 point.
Why?

Each square is completely contained in one side of the red line.

Two points on either side are at least distance d apart.

Divide and conquer approach

Lemma

Let S_y be the points in the distance d region from the red line sorted in decreasing order of their y-coordinates. Say $S_y = \langle q_1, \ldots, q_m \rangle$. If the distance between some q_i and q_j is $< d$ then $j - i \leq 15$.

Suppose two points are separated by > 15 indices.

At least 3 full rows separate them.

But the height of 3 rows is $\geq 3d/2$, i.e. $> d$

Hence such two points are at least distance d apart.