ICCS312 Algorithms and Tractability (Term 1/2021-22)

Lecture 4: Divide and Conquer (II)

Sunsern Cheamanunkul

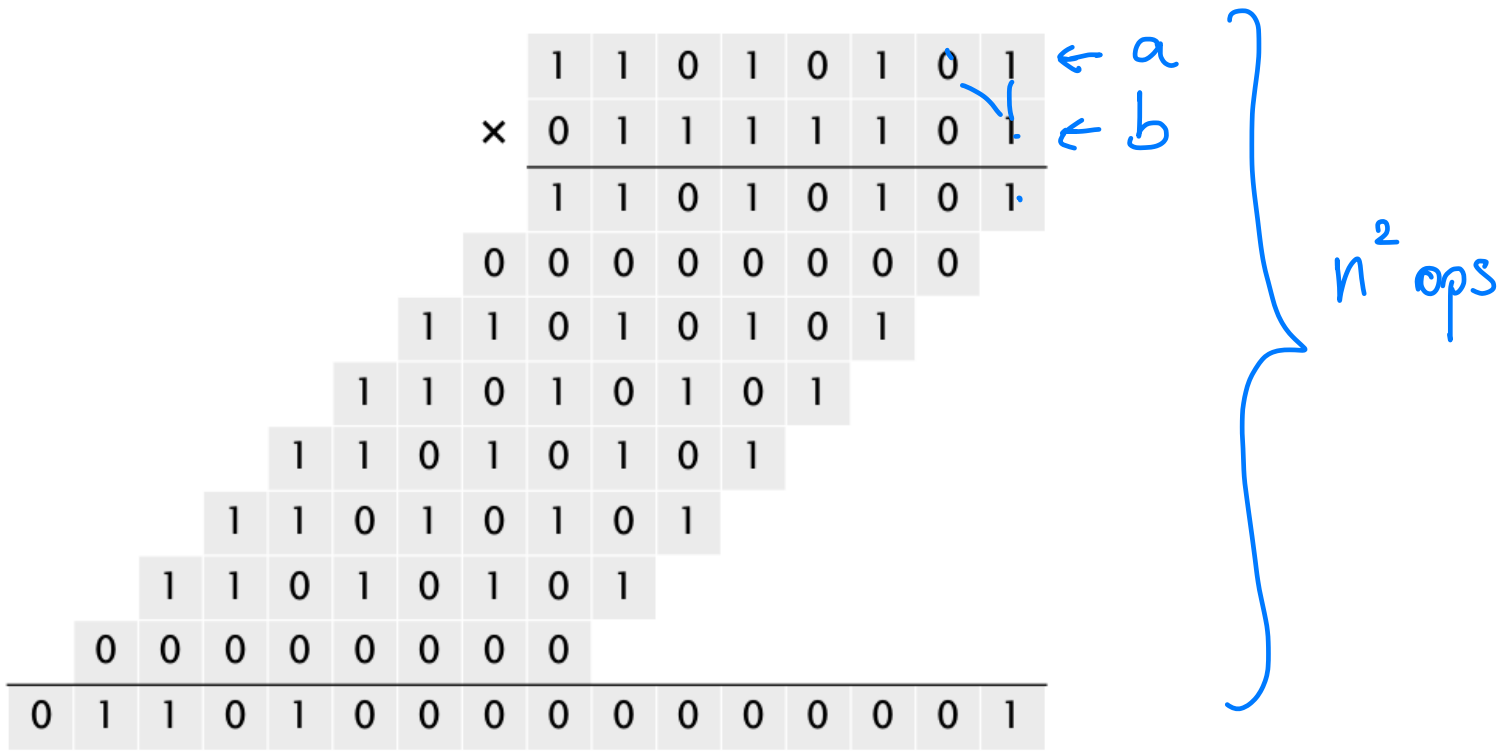
Mahidol University International College

September 23, 2021

1. Fast Multiplication
2. Master Theorem
3. Closest pair of points

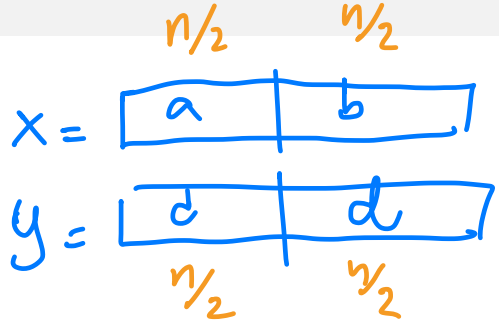
Integer Multiplication

Given two n-bit integers a and b, compute $a \times b$



- What is the running time? $O(n^2)$

Divide-and-Conquer multiplication



- Idea: To multiply two n-bit integers x and y
  - Divide x and y into low- and high-order bits
  - Multiply **four** $\frac{n}{2}$-bit integers, recursively
  - Add and shift to obtain result.

$$m = \lceil n/2 \rceil$$
$$a = \lfloor x/2^m \rfloor \qquad b = x \mod 2^m$$
$$c = \lfloor y/2^m \rfloor \qquad d = y \mod 2^m$$

$$xy = (2^m a + b)(2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd$$

Divide-and-Conquer multiplication

$x = 1101$ 0011 $x >> 2 = 11$ $x \,\&\, (2^2 - 1) = 01$

Algorithm: Multiply(x,y,n)

```
If (n == 1): return  x*y
m = n // 2
a = x >> m                          O(1)
b = y >> m                          O(1)
c = x & (2^m - 1)                   O(1)
d = y & (2^m - 1)                   O(1)
e = Multiply(a,c,m)                 T(n/2)
f = Multiply(b,d,m)                 T(n/2)
g = Multiply(b,c,m)                 T(n/2)
h = Multiply(a,d,m)                 T(n/2)
return 2^(2*m)*e + 2^m * (g+h) + f      O(n)
```

How fast is Multiply?

The recurrence is

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 4T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$

- What is the running time?

Master Theorem

Divide-and-conquer algorithms often follow a generic pattern: they tackle a problem of size n by recursively solving, say, a subproblems of size n/b and then combining these answers in $O(n^d)$ time, for some $a, b, d > 0$.

For example, in Multiply, we have $a = 4$, $b = 2$, and $d = 1$, or

$$T(n) = 4T(n/2) + O(n^1)$$

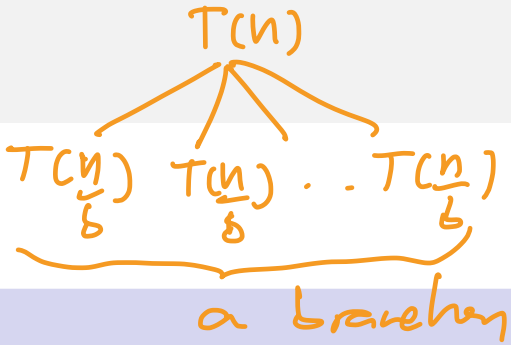
Master Theorem

Master Theorem

If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a > 0$, $b > 1$ and $d \geqslant 0$, then

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

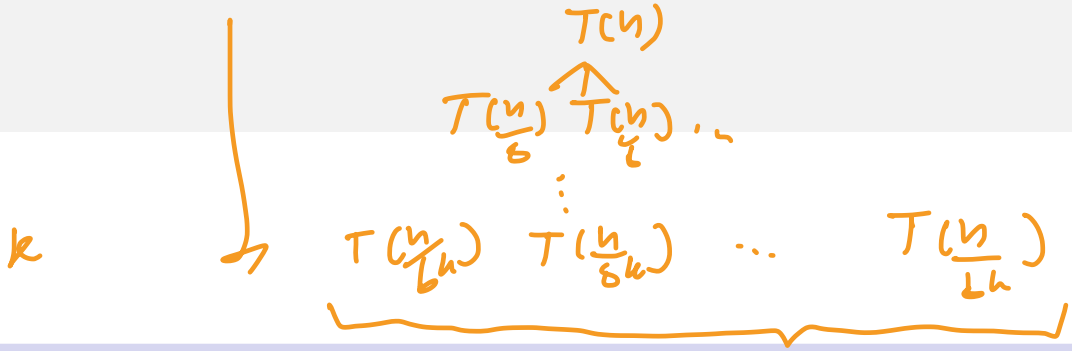
Master Theorem



Proof. [Master Theorem]

- Assuming for the sake of convenience that n is a power of b.
- After all, n is at most a multiplicative factor of b away from some power of b.
- This allows us to ignore the rounding effect in $\lceil n/b \rceil$.
- Next, notice that the size of the subproblems decreases by a factor of b with each level of recursion.
- The base case will be reached after $\log_b n$ levels.
- Its branching factor is a, so the k-th level of the tree is made up of a^k subproblem, search of size n/b^k.

Master Theorem



Proof. [Master Theorem]

So the total work at level k is given by:

$$a^k \times O((\frac{n}{b^k})^d) = O(n^d) \times (\frac{a}{b^d})^k$$

As k goes from 0 to $\log_b n$, the overall work done can be written as:

$$\sum_{k=1}^{\log_b n} O(n^d) \times (\frac{a}{b^d})^k$$

Master Theorem

$$\sum_{k=1}^{\log_b n} O(n^d) \times (\frac{a}{b^d})^k$$

Proof. [Master Theorem]

When consider following cases:

- When $a < b^d$, we have a decreasing geometric series. The sum can be bounded by its first term: $O(n^d)$.
- When $a > b^d$, we have a increasing geometric series. The sum can be bounded by its last term: $O(n^{\log_b a})$.

$$n^d(\frac{a}{b^d})^{\log_b n} = n^d(\frac{a^{\log_b n}}{(b^{\log_b n})^d}) = a^{\log_b n} = a^{(\log_a n)(\log_b a)} = n^{\log_b a}$$

- When $a = b^d$, all $O(\log n)$ terms are equal to $O(n^d)$.

Running time of Multiply

Master Theorem

If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a > 0$, $b > 1$ and $d \geqslant 0$, then

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

For Multiply,

$a=4$ $b=2$ $d=1$.

$\log_b a = \log_2 4 = 2$.

$$T(n) = 4T(n/2) + O(n^1)$$

We have $a = 4$, $b = 2$ and $d = 1$. Thus, $T(n) = O(n^{\log_b a}) = O(n^2)$.

- No improvement from brute-force algorithm!

Example: Power

Assume that the cost of multiplication here is $O(1)$. Consider the following pseudo-code:

```
long power(long x, long n) {
  if (n == 0)
    return 1;
  else
    return x * power(x, n-1);
}
```

- **Question:** What is the running time?

Example: Power

The recurrence for Power is:

$$T(n) = \begin{cases} T(n-1) + O(1) & \text{if n} > 0 \\ O(1) & \text{otherwise} \end{cases}$$

- Can we apply Master theorem here?

Example: Power

$T(n) = (T(n-2)+O(1)) + c$

$= (T(n-3) + c + c$

$= \cdots$

$= c + c + c + \cdots + c$

$n = c \cdot n$

$= O(n)$

The recurrence for Power is:

$$T(n) = \begin{cases} T(n-1) + O(1) & \text{if n} > 0 \\ O(1) & \text{otherwise} \end{cases}$$

- Can we apply Master theorem here?
- No, we cannot. Master theorem helps only when the recurrence relation matches with the pattern.
- Master theorem is *not* a cure for all diseases.

Back to Multiply

- Karatsuba's trick
- To multiply two n-bit integers x and y

$$xy = (2^m a + b)(2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd$$

- The middle term can be written as:

$$bc + ad = ac + bd - (a - b)(c - d)$$

- Now, we only need *three* multiplications!

Karatsuba-Multiply

Algorithm: Karatsuba-Multiply(x,y,n)

```
If (n == 1): return  x*y
m = n // 2
a = x >> m
b = y >> m
c = x & (2^m - 1)
d = y & (2^m - 1)
e = Karatsuba-Multiply(a,c,m)
f = Karatsuba-Multiply(b,d,m)
g = Karatsuba-Multiply(a-b,c-d,m)
return 2^(2*m)*e + 2^m * (e+f-g) + f
```

O(1)

T(n/2)

T(n/2)

T(n/2)

O(n)

Karatsuba-Multiply

The recurrence of Karatsuba-Multiply is:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 3T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$

- We have $a = 3$, $b = 2$ and $d = 1$.
- By Master theorem, we know that

$$T(n) = O(n^{\log_2 3}) = O(n^{1.585})$$

Algorithms for integer multiplication

K&T book

year	algorithm	bit operations
12xx	grade school	$O(n^2)$
1962	Karatsuba–Ofman	$O(n^{1.585})$
1963	Toom-3, Toom-4	$O(n^{1.465})$, $O(n^{1.404})$
1966	Toom–Cook	$O(n^{1+\varepsilon})$
1971	Schönhage–Strassen	$O(n \log n \cdot \log \log n)$
2007	Fürer	$n \log n \, 2^{O(\log^* n)}$
2019	Harvey–van der Hoeven	$O(n \log n)$
		$O(n)$

number of bit operations to multiply two n-bit integers

$$T(n) = 7T(n/4) + O(n^2)$$

$a = 7$

$b = 4$

$$\log_4 7 = \frac{\log_2 7}{\log_2 4} = \frac{2.8x}{2} \sim 1.4\,0\,4$$

$d = 2$

So, $T(n)$ is $O(n^2)$

Master Theorem

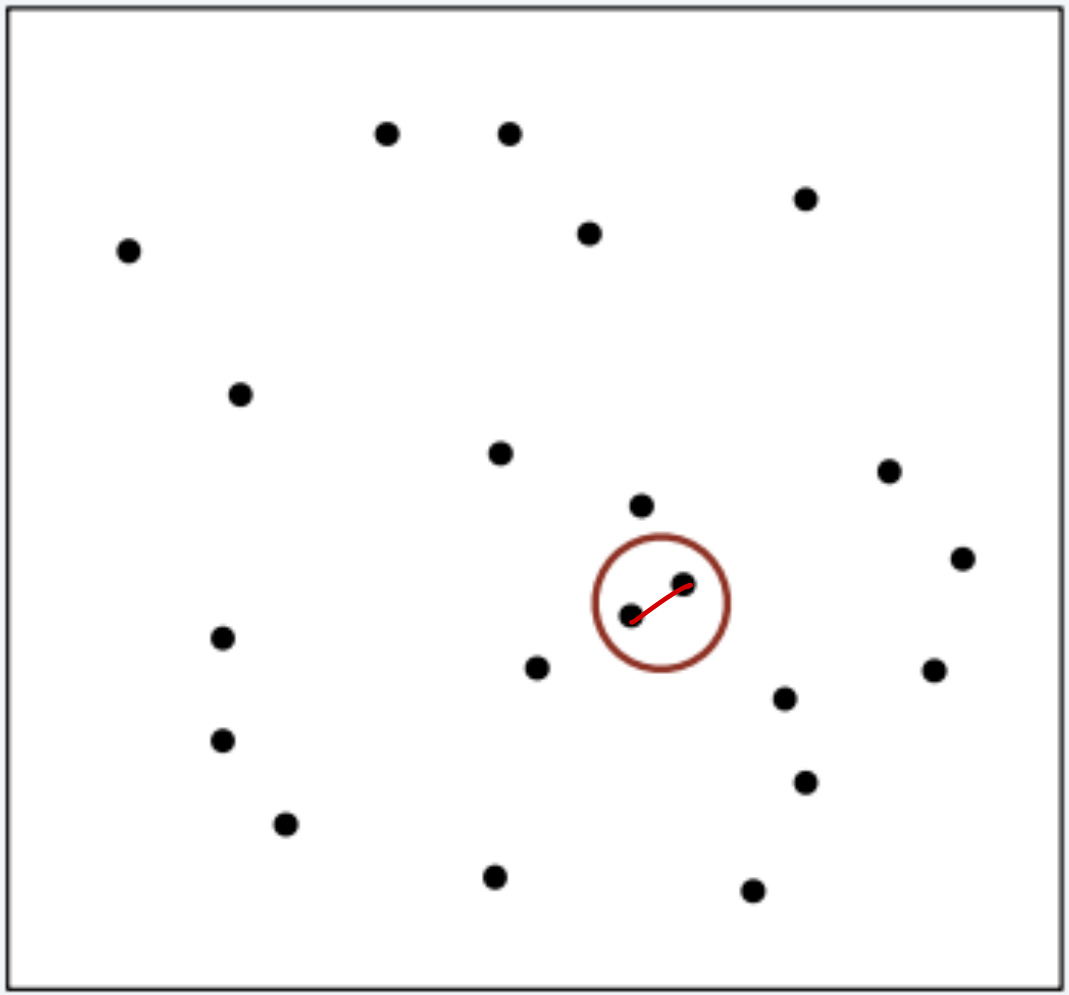
If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a > 0$, $b > 1$ and $d \geqslant 0$, then

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

Closest pair of points

- Given n points in the plane, find a pair of points with the smallest Euclidean distance between them.

L_2 distance

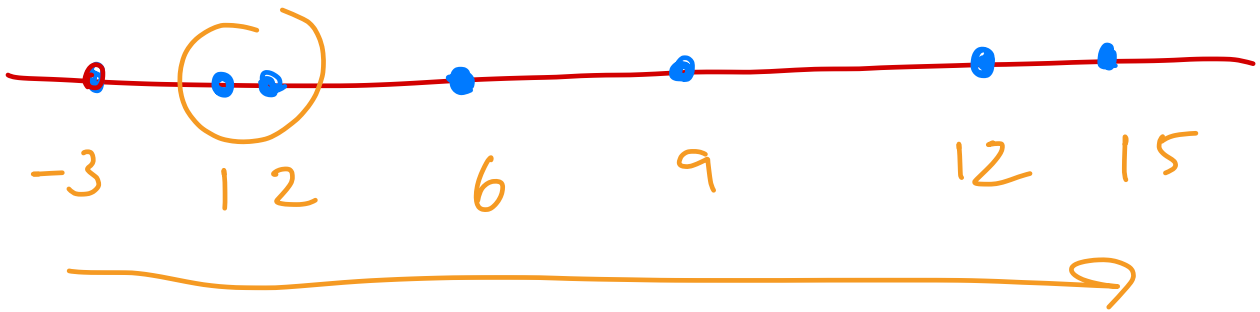


Closest pair of points

Input $\{6, 9, -3, 1, 12, 2, 15\}$

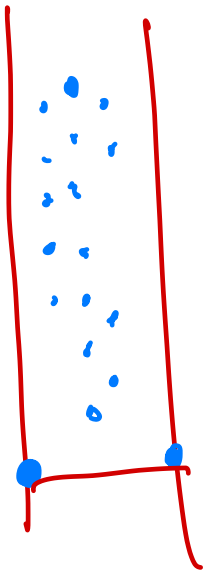
Output $(1, 2)$

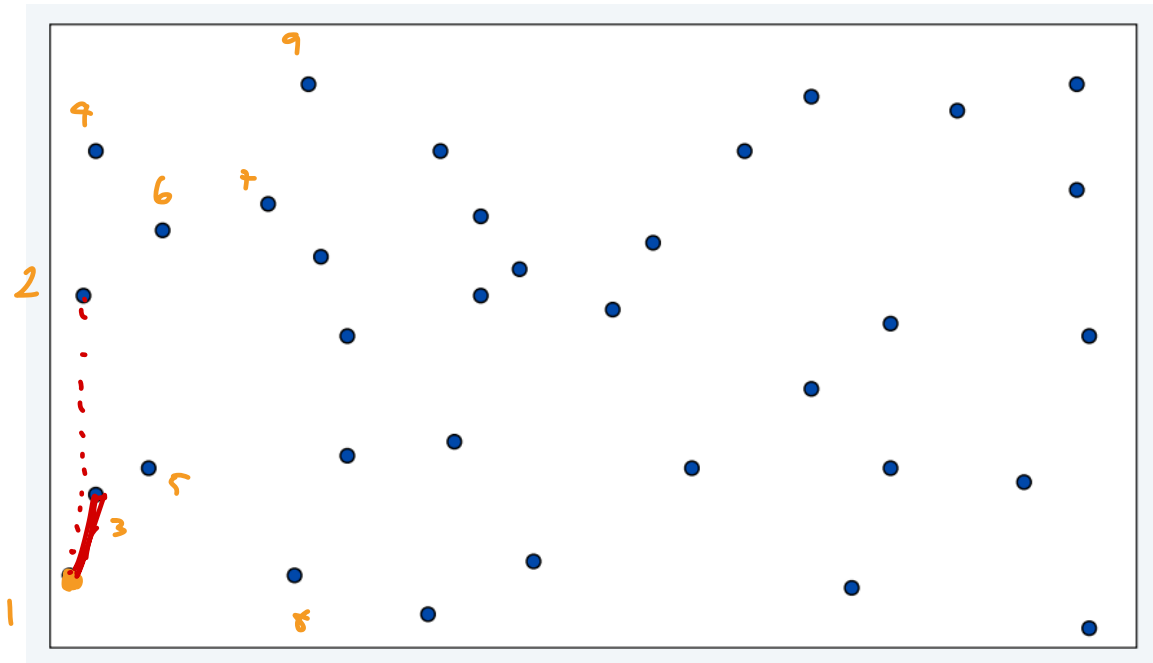
- Brute force: Check all pairs with $\Theta(n^2)$ distance calculations.
- Simpler variants:
  - 1D Version: all points lie on a line. $O(n \log n)$



First attempt

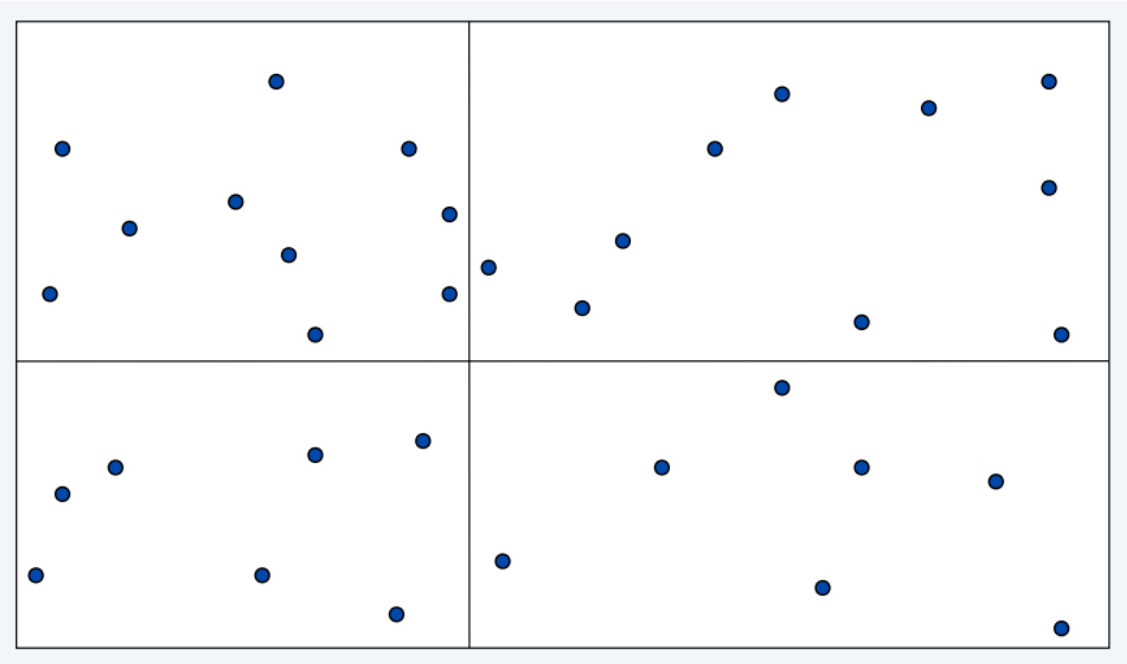
- Idea:
  - Sort by x-coordinate and consider nearby points.
  - Sort by y-coordinate and consider nearby points.

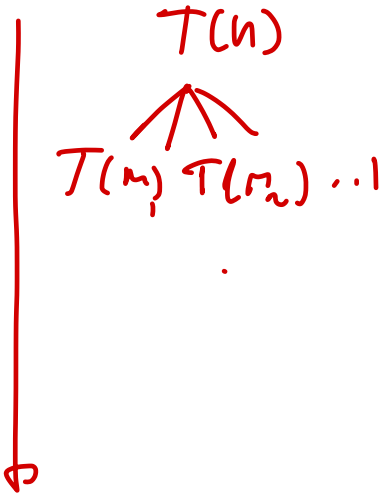




Second attempt

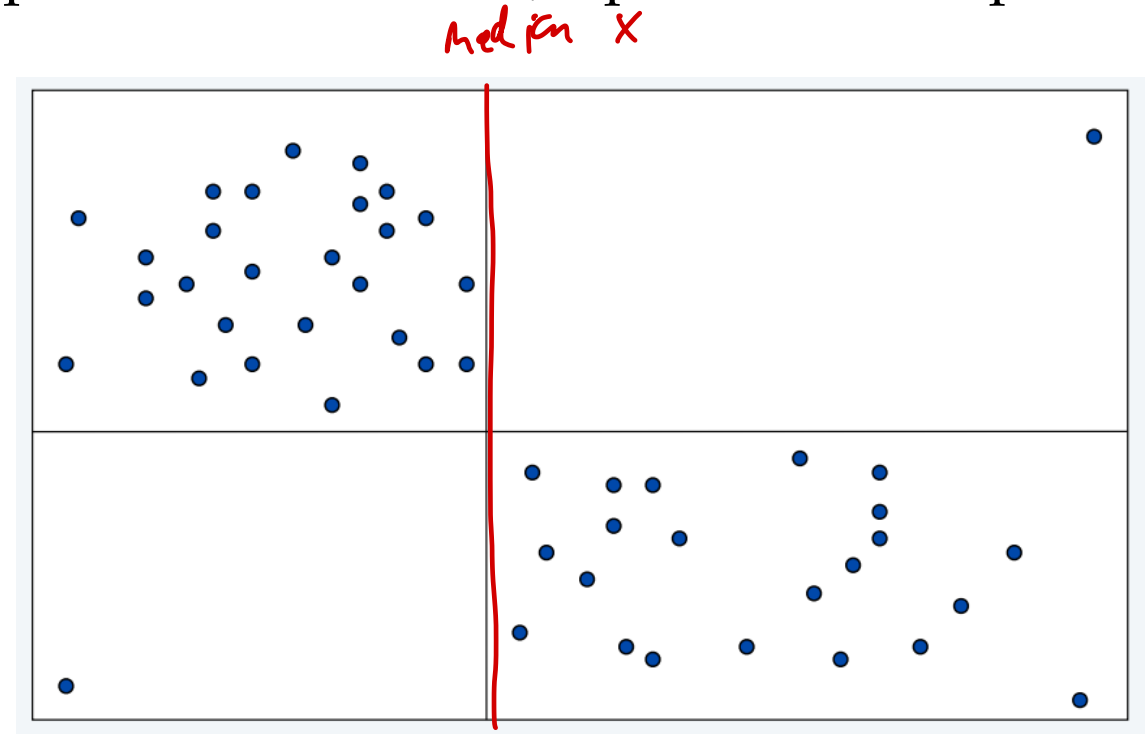
- Idea: Subdivide region into 4 quadrants





Second attempt

- Idea: Subdivide region into 4 quadrants
- Issue: Impossible to ensure $n/4$ points in each piece

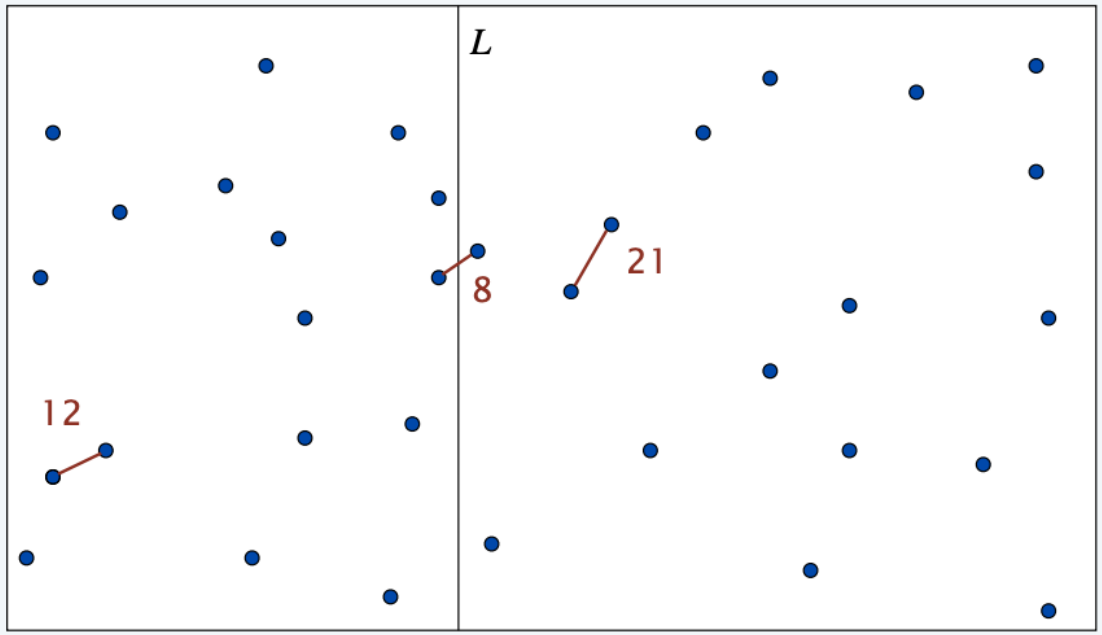


Divide-and-conquer

- Divide: draw vertical line L so that $n/2$ points on each side.
- Conquer: find the closest pair in each side recursively.
- Combine: find the closest pair with one point in each side.
- Return best of 3 solutions.

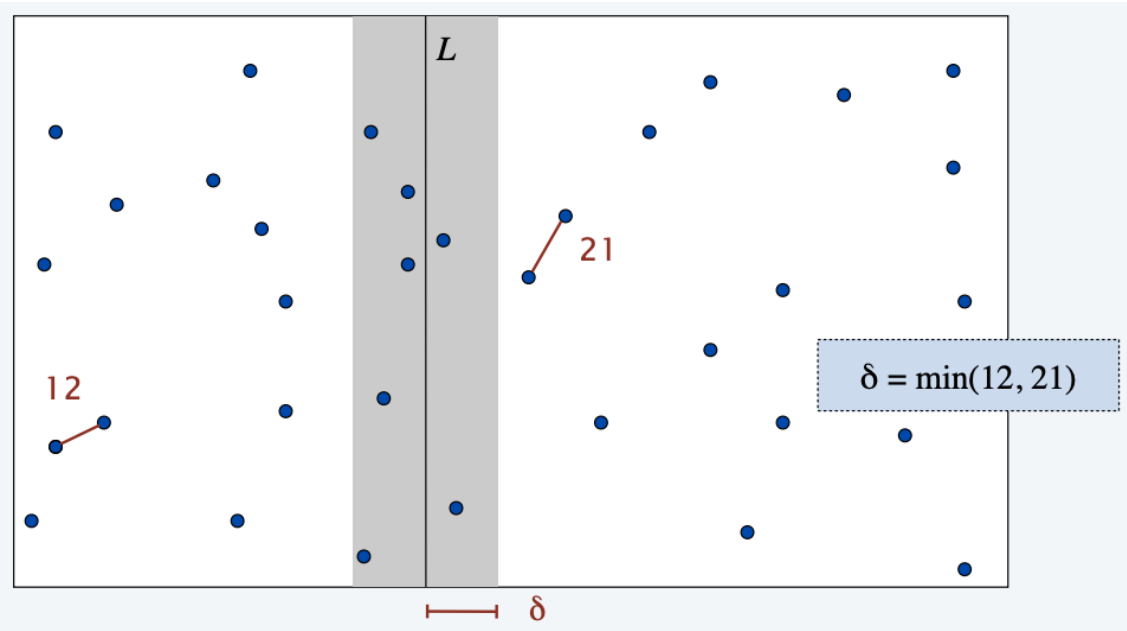
sounds like $O(n^2)$

$\sim O(n^2)$



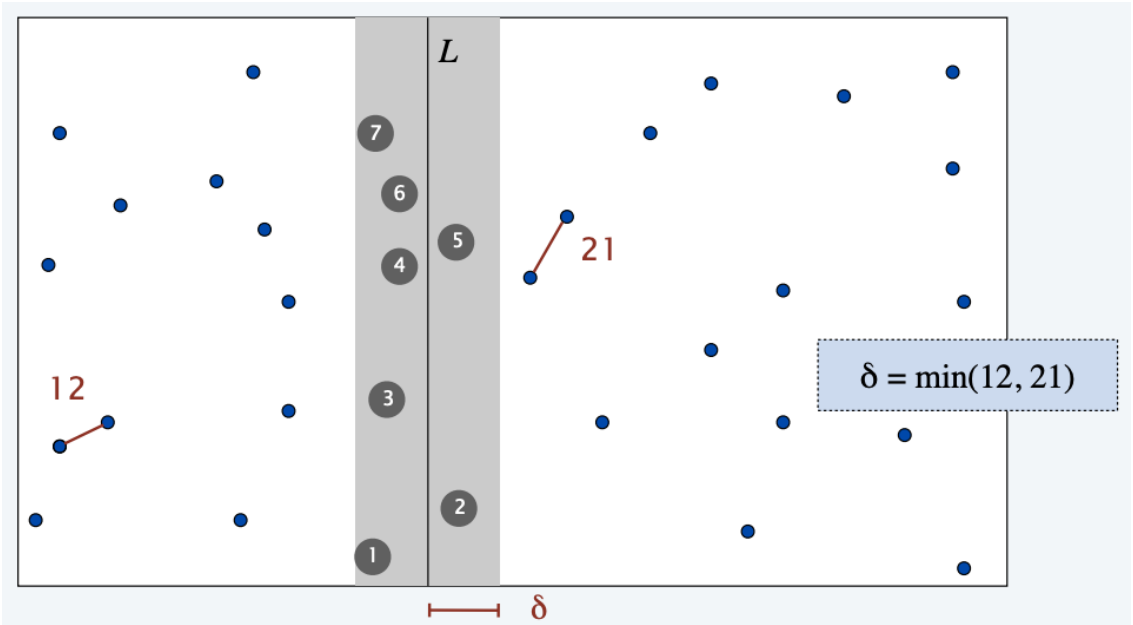
How to find closest pair with one point in each side?

- Find closest pair with one point in each side, assuming that distance $< \delta$.
- Observation: suffices to consider only those points within δ of line L.
- Why?



How to find closest pair with one point in each side?

- Idea:
  - Sort points in the 2δ-strip by their y-coordinate
  - Check distances of only those points within 7 positions in the sorted list.



How to find closest pair with one point in each side?

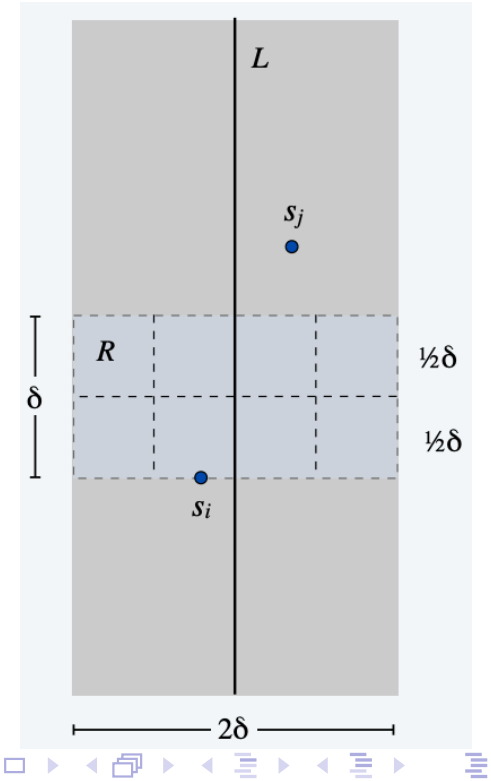
Let s_i be the point in the 2δ-strip, with the i-th smallest y-coordinate.

Claim

If $|j - i| > 7$, then the distance between s_i and s_j is at least δ

Proof.

Consider the 2δ-by-δ rectangle R in the strip whose min y-coordinate is y-coordinate of s_i. Distance between s_i and any point s_j above R $\geqslant \delta$. Subdivide R into 8 $\delta/2$-by-$\delta/2$ squares. Diagonal of each square is $\delta/\sqrt{2}$. There cannot be two points in the same squares as it would contradict our assumption. So, there can be at most 1 point per square $\Rightarrow$ there can be at most 7 other points in R. ■



Closest-Pair Algorithm

$T(n)$

Algorithm: Closest-Pair($p_1, p_2, .., p_n$)

- Compute separation line L such that half the points are on each side.
- $l \leftarrow$ Closest Pair in the left half $T(n/2)$
- $r \leftarrow$ Closest Pair in the right half $T(n/2)$
- $\delta \leftarrow \min(l, r)$
- Delete all points further than δ from L $O(n)$
- Sort points in y-order $O(n \log n)$
- Scan points in y-order and compute distance between each point and next 7 neighbors, and update δ accordingly $O(n)$
- return δ

Closest-Pair Algorithm

- The recurrence is:

$$T(n) \leqslant 2T(\lceil n/2 \rceil) + O(n \log n)$$

Closest-Pair Algorithm

- The recurrence is:

$$T(n) \leqslant 2T(\lceil n/2 \rceil) + O(n \log n)$$

- Using recursion tree argument, the amount of work per level is $n \log n$ and there are $\log n$ levels. Hence, the total running time is is $O(n(\log n)^2)$

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

Further Improvement

- Look for redundancy!

Further Improvement

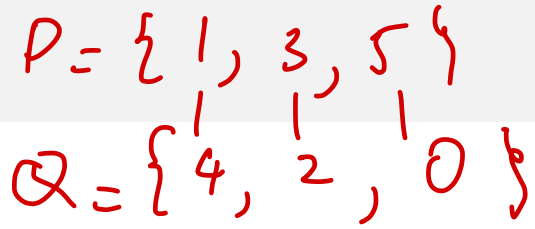
- Look for redundancy!
- We sort the points from scratch each time.
- Avoid by returning two sorted lists: by x-coordinate and by y-coordinates.
- Merge sorted sub-lists similar to Sort-and-Count.

The running time becomes:

$$T(n) \leqslant 2T(\lceil n/2 \rceil) + O(n)$$

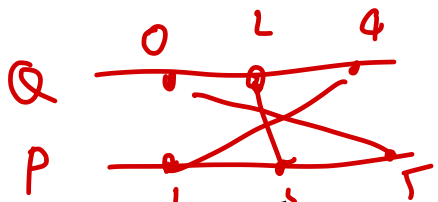
which is $O(n \log n)$

Discussion: Intersections



Suppose you are given two sets of n points.

- $\{p_1, p_2, \ldots, p_n\}$ on the line $y = 0$
- $\{q_1, q_2, \ldots, q_n\}$ on the line $y = 1$



Then, create a set of n line segments by connecting each point p_i to the corresponding point q_i.

Question: Describe and analyze a divide-and-conquer algorithm to determine how many pairs of these line segments intersect, in $O(n \log n)$ time.