Contents Lecture 5

- The divide and conquer algorithm design technique
- Analysing a divide and conquer algorithm: Mergesort
- Counting inversions
- Closest pair of points

Refresher on proof by induction

Lemma

$S(n) = 1 + 2 + \ldots + n = n(n+1)/2$

Proof.

- Induction on n
- Base case $n = 1$: $S(1) = 1(1+1)/2 = 1$
- Induction hypothesis: $S(n)$ is true for $i = 1, 2, \ldots, n$
- Show that $S(n+1)$ is true using the induction hypothesis

$$\begin{aligned}
S(n+1) &= S(n) + n + 1 \\
&= n(n+1)/2 + n + 1 \\
&= n(n+1)/2 + 2(n+1)/2 \\
&= \frac{n(n+1)+2(n+1)}{2} \\
&= \frac{n^2+n+2n+2)}{2} \\
&= \frac{(n+1)(n+2)}{2} = S(n+1)
\end{aligned}$$

The divide and conquer algorithm design technique

- Suppose you have n items of input and the simplest technique to process it would be two nested for loops with a $\Theta(n^2)$ running time
- If n is small then this is fine
- With divide and conquer we instead aim at:
  - Divide in linear time the problem into two subproblems with $n/2$ items
  - Solve each subproblem
  - Combine the solutions to the subproblems in linear time into a solution for the n item problem
- The resulting running time becomes $\Theta(n \log n)$
- We will next study Mergesort

4 GHz modern CPU

n	n	$n \log n$	n^2	n^3	1.5^n	2^n	$n!$
10	2.5 ns	8.3 ns	25.0 ns	250.0 ns	14.4 ns	256.0 ns	907.2 μs
11	2.8 ns	9.5 ns	30.2 ns	332.8 ns	21.6 ns	512.0 ns	10.0 ms
12	3.0 ns	10.8 ns	36.0 ns	432.0 ns	32.4 ns	1.0 μs	119.8 ms
13	3.2 ns	12.0 ns	42.2 ns	549.2 ns	48.7 ns	2.0 μs	1.6 s
14	3.5 ns	13.3 ns	49.0 ns	686.0 ns	73.0 ns	4.1 μs	21.8 s
15	3.8 ns	14.7 ns	56.2 ns	843.8 ns	109.5 ns	8.2 μs	5 min
16	4.0 ns	16.0 ns	64.0 ns	1.0 μs	164.2 ns	16.4 μs	1 hour
17	4.2 ns	17.4 ns	72.2 ns	1.2 μs	246.3 ns	32.8 μs	1.0 days
18	4.5 ns	18.8 ns	81.0 ns	1.5 μs	369.5 ns	65.5 μs	18.5 days
19	4.8 ns	20.2 ns	90.2 ns	1.7 μs	554.2 ns	131.1 μs	352.0 days
20	5.0 ns	21.6 ns	100.0 ns	2.0 μs	831.3 ns	262.1 μs	19 years
30	7.5 ns	36.8 ns	225.0 ns	6.8 μs	47.9 μs	268.4 ms	10^{15} years
40	10.0 ns	53.2 ns	400.0 ns	16.0 μs	2.8 ms	5 min	10^{31} years
50	12.5 ns	70.5 ns	625.0 ns	31.2 μs	159.4 ms	3.3 days	10^{47} years
100	25.0 ns	166.1 ns	2.5 μs	250.0 μs	3 years	10^{13} years	10^{141} years
1000	250.0 ns	2.5 μs	250.0 μs	250.0 ms	10^{159} years	10^{284} years	huge
10^4	2.5 μs	33.2 μs	25.0 ms	4 min	huge	huge	huge
10^5	25.0 μs	415.2 μs	2.5 s	2.9 days	huge	huge	huge
10^6	250.0 μs	5.0 ms	4 min	8 years	huge	huge	huge
10^7	2.5 ms	58.1 ms	7 hour	10^4 years	huge	huge	huge
10^8	25.0 ms	664.4 ms	28.9 days	10^7 years	huge	huge	huge
10^9	250.0 ms	7.5 s	8 years	10^{10} years	huge	huge	huge

Mergesort

- Mergesort is a stable sort algorithm
- Running time $\Theta(n \log n)$
- See `mergesort.c` e.g. in the book

Recurrence relation

- Swedish differensekvation or rekursionsekvation
- A **recurrence relation** or just **recurrence** is a set of equalities or inequalities such as

$$T(n) = \begin{cases} 0, & n = 1 \\ 2T(n/2) + n, & n > 1 \end{cases}$$

- The value of $T(n)$ is expressed using smaller instances of itself and a boundary value.
- To analyze the running time of a divide and conquer algorithm, recurrences are very natural
- But we want to have an expression for $T(n)$ in **closed form**
- Closed form means an expression only involving functions and operations from a generally accepted set — i.e. ”common knowledge”.
- Closed form can also be called **explicit form**
- So our next goal is to rewrite $T(n)$ into closed form

Mergesort recurrence

- $T(n)$ = max comparisons to mergesort n items
- Mergesort recurrence:

$$T(n) \leq \begin{cases} 0, & n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n, & n > 1 \end{cases}$$

- This is a simplification as can be seen if compared with the source code, but it is sufficiently accurate.
- We initially ignore ceil and floor:

$$T(n) \leq \begin{cases} 0, & n = 1 \\ 2T(n/2) + n, & n > 1 \end{cases}$$

- We also assume n is a power of 2
- We will show that these simplifications do not affect our running time analysis, i.e. they are valid

Rewriting a recurrence to closed form

- The easiest way to understand what the closed form is, may be to ”expand” or ”unroll” the recurrence and simply ”see” what is happening
- For Mergesort the closed form will be easy to find this way
- Another way is to look at small inputs and try to guess the closed form
- When we have a guess which works for the small inputs, we then prove by induction that our guess is correct
- In both cases we prove our closed form by induction
- We will start with expanding $T(n)$

$$T(n) \le \begin{cases} 0, & n = 1 \\ 2T(n/2) + n, & n > 1 \end{cases}$$

$T(n)$	n
$T(n/2)$ $T(n/2)$	$2(n/2)$
$T(n/4)$ $T(n/4)$ $T(n/4)$ $T(n/4)$	$4(n/4)$
$T(n/8)$ $T(n/8)$ $T(n/8)$ $T(n/8)$ $T(n/8)$ $T(n/8)$ $T(n/8)$ $T(n/8)$	$8(n/8)$
$\vdots$	$\vdots$

- Assume n is power of 2
- $\log_2 n$ levels
- n comparisons per level
- In total $n \log n$ comparisons
- $T(n) = n \log n$

Proof by induction

Lemma

The recurrence

$$T(n) = \begin{cases} 0, & n = 1 \\ 2T(n/2) + n, & n > 1 \end{cases}$$

has the closed form $T(n) = n \log_2 n$.

Proof.

- Recall $\log ab = \log a + \log b$, so $\log_2 2n = \log_2 n + \log_2 2 = \log_2 n + 1$, and $\log_2 n = \log_2 2n - 1$
- Induction on n.
- Base case: $n = 1$: $T(1) = 1 \log_2 1 = 0$
- Induction hypothesis: assume $T(n) = n \log_2 n$
- $T(2n) = 2T(n) + 2n = 2n \log_2 n + 2n = 2n(\log_2 n + 1) = 2n(\log_2 2n - 1 + 1) = 2n \log 2n$

$\square$

Remark about previous proof

- Normally we assume $S(i)$ is true and prove $S(i+1)$
- On previous slide we did not increment by one but rather doubled our variable
- We could have stated the lemma in terms of $S(i)$ and let $n = 2^i$
- Then we use induction on i and assume $S(i)$ and prove $S(i+1)$

Proof by induction, removing assumption $n = 2^k$

Lemma

The recurrence

$$T(n) \le \begin{cases} 0, & n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n, & n > 1 \end{cases}$$

has the closed form $T(n) \le n \lceil \log_2 n \rceil$.

Proof.

- Induction on n
- Base case: $n = 1$: $T(1) = 1 \log_2 1 = 0$
- Let $n_1 = \lfloor n/2 \rfloor$ and $n_2 = \lceil n/2 \rceil$
- Induction hypothesis: assume true for $n = 1, 2, \ldots, n-1$
- $T(n) \le T(n_1) + T(n_2) + n \le n_1 \log_2 n_1 + n_2 \log_2 n_2 + n$
- Previous line follows from induction hypothesis

Proof by induction, removing assumption $n = 2^k$

Lemma

The recurrence

$$T(n) \leq \begin{cases} 0, & n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n, & n > 1 \end{cases}$$

has the closed form $T(n) \leq n \lceil \log_2 n \rceil$.

Proof.

$$\begin{aligned}
T(n) &\leq T(n_1) + T(n_2) + n \\
&\leq n_1 \log_2 n_1 + n_2 \log_2 n_2 + n \\
&\leq n_1 \log_2 n_2 + n_2 \log_2 n_2 + n \\
&\leq (n_1 + n_2) \log_2 n_2 + n \\
&= n \log_2 n_2 + n \\
&\leq n(\lceil \log_2 n \rceil - 1) + n \\
&= n \lceil \log_2 n \rceil
\end{aligned}$$

Looking at small inputs

$$T(n) \leq \begin{cases} 0, & n = 1 \\ 2T(n/2) + n, & n > 1 \end{cases}$$

- Let us try out some small values:

n	1	2	4	8	16	32	64
$T(n)$	0	2	8	24	64	160	384

- Can we identify a pattern?

n	1	2	4	8	16	32	64
$T(n)$	0	2	8	24	64	160	384
$T(n)/n$	0	1	2	3	4	5	6

- $\log_2 n$ is incremented by one when n is doubled: $\log_2 2n = 1 + \log_2 n$
- So $T(n) = n \log_2 n$ is tempting to try to prove by induction, which we already know is true

The master theorem (MSc thesis by Dorothea Haken)

- There is a nice formula for finding $T(n)$ for many recursive algorithms:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= aT(n/b) + n^s. \end{aligned}$$

- There are three closed form solutions (for details, see the book):

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

- $T(n) = 2T(n/2) + n$. With $a = b = 2$ and $s = 1$, we have $\log_b a = \log_2 2 = 1 = s$, so $T(n) = O(n \log n)$.
- $T(n) = 2T(n/2) + \sqrt{n}$. With $a = b = 2$ and $s = 0.5$, we have $\log_b a = \log_2 2 = 1 > s$, so $T(n) = O(n)$.
- $T(n) = 4T(n/3) + n^2$. We have $\log_b a = \log_3 4 = \frac{\log_{10} 4}{\log_{10} 3} = 1.26 < s = 2$, so $T(n) = O(n^2)$.

Finding people with similar tastes

- Consider a category such as text editor, programming language, preferred tab width, or the 22 Mozart operas
- To compare how similar tastes within a category three people have, they can rank a list of say 5 operas A-E

Tintin:	A	D	C	E	B
Captain Haddock:	A	C	B	D	E
Bianca Castafiore:	A	B	D	C	E

- All agree opera A is best
- Who have most similar tastes?

Inversions

Tintin:	A	D	C	E	B
Captain Haddock:	A	C	B	D	E
Bianca Castafiore:	A	B	D	C	E
- We have 5 positions in each list
- Start with Tintin's list and label each item $1, 2, \ldots, 5$:

Tintin:	A	D	C	E	B
Tintin:	1	2	3	4	5
- Then we put these labels according to Captain Haddock's ranking:

Captain Haddock:	1	3	5	2	4
	a_1	a_2	a_3	a_4	a_5
- i and j are **inverted** if $i < j$ and $a_i > a_j$
- Inversions: (3,2), (5,2), and (5,4)
- The fewer inversions, the more similar tastes (obviously)

Counting inversions

```
for (c = i = 0; i < n; i += 1)
        for (j = i+1; j < n; j += 1)
                if (a[i] > a[j])
                        c += 1;

printf("%d inversions\n", c);
```

- Running time is $O(n^2)$
- How can we use divide and conquer to achieve $O(n \log n)$?
  1 3 | 5 2 4
- Count inversions in left part
- Count inversions in right part
- Somehow combine these parts and add number of inversions...???

What can we do to simplify the problem?

- 1 3 | 5 2 4
- Assume you know there are no inversions in the left part and two in the right part
- It is OK to "destroy" the array, such as sorting it, if that helps...
- If modifying the array is forbidden, we can always make a copy and work with the copy instead
- Copying the array is fine since that is faster than $O(n \log n)$
- Copying the array is $O(n)$ but memory allocation can be costly so don't do it too much
- For Mergesort, it is non-trivial to not use a second array

Sorting the array

- By subarray is meant the part our recursive subproblem is going to work with
- Sorting the subarray **after** counting the inversions may help
- 1 3 | 5 2 4
- After having counted in the subarrays we have: 1 3 | 2 4 5
- Combining two sorted parts can be done in linear time as in Mergesort

  3 | 2 4 5 ‖ 1
  3 | 4 5 ‖ 1 2

  The 2 was inverted with each remaining in left part — only the 3 in this example so one inversion is counted when the parts are combined

  | 4 5 ‖ 1 2 3
  | 5 ‖ 1 2 3 4
  | ‖ 1 2 3 4 5

- In total 3 inversions

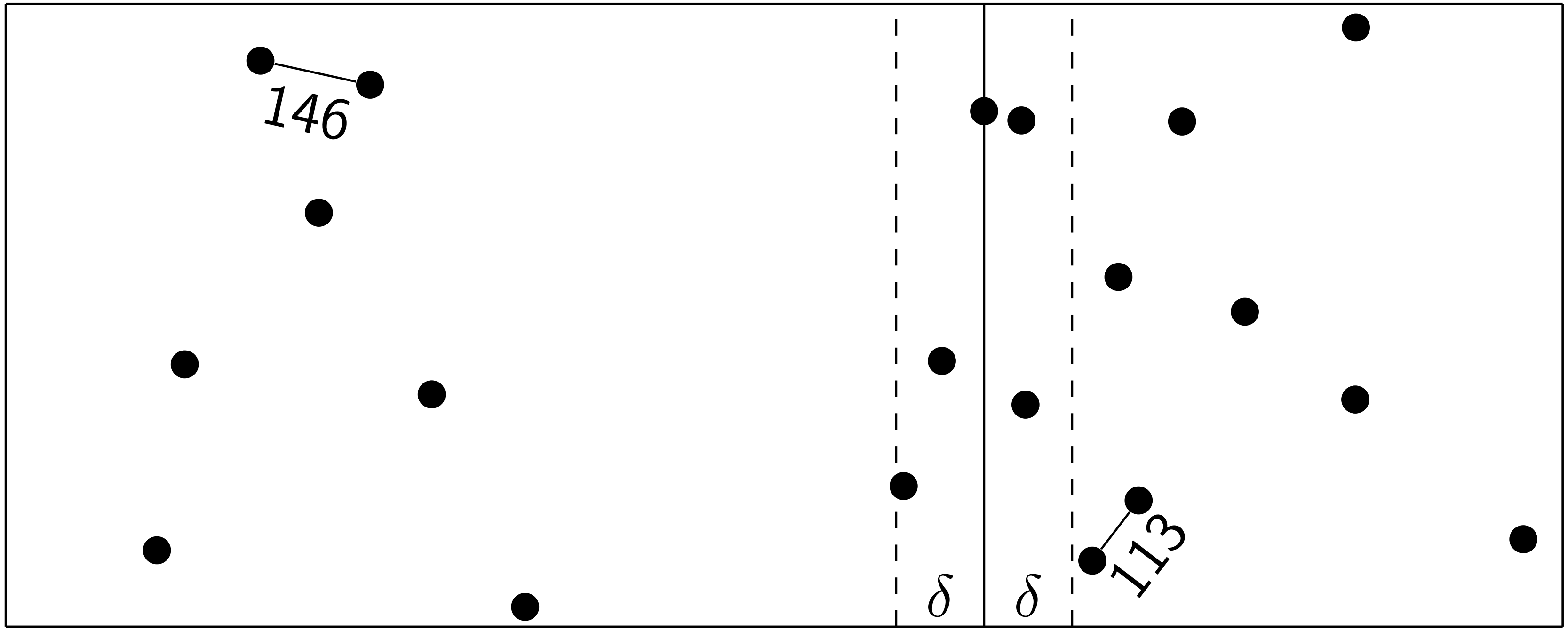
Implementing the $n \log n$ algorithm

- As always: first make a simple reference implementation that can be used to verify the correctness of a faster implementation
- In this case the n^2 algorithm is ideal if used with small inputs

Lab 4: Closest points in a plane

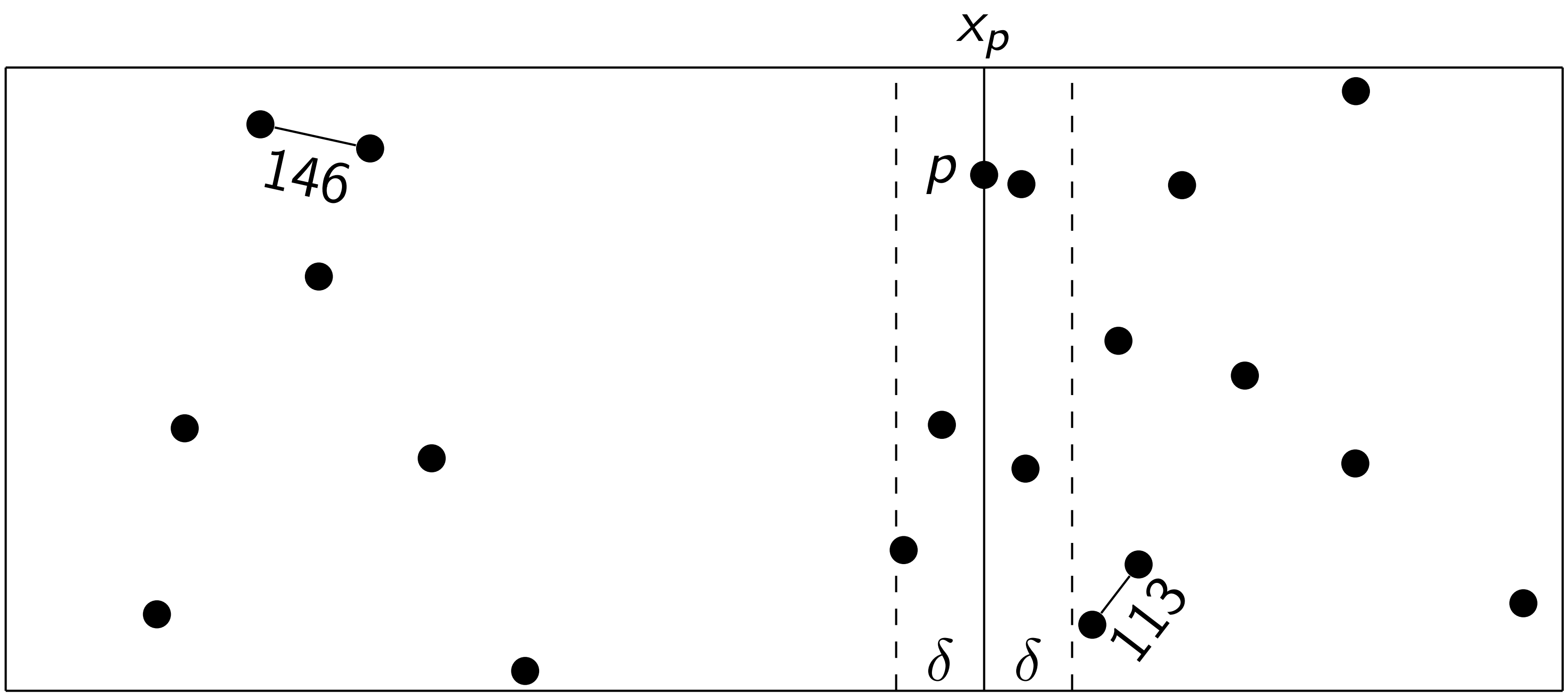
- Lab 4 is about the field of **computational geometry**
- Consider n points (x_i, y_i) in a plane
- We want to find which points are closest
- Comparing all points with each other in an n^2 algorithm is simple
- But comparing points ”obviously” far from each other is a waste
- How can divide and conquer be used to find an $n \log n$ algorithm?
- We cut the plane in two halves and find closest points in each half
- We have then three categories of point pairs which can be closest:
  1. Point pairs in the left half
  2. Point pairs in the right half
  3. Point pairs with one point in the left and the other in the right half
- Can we find close points from the last category in linear time???

An example



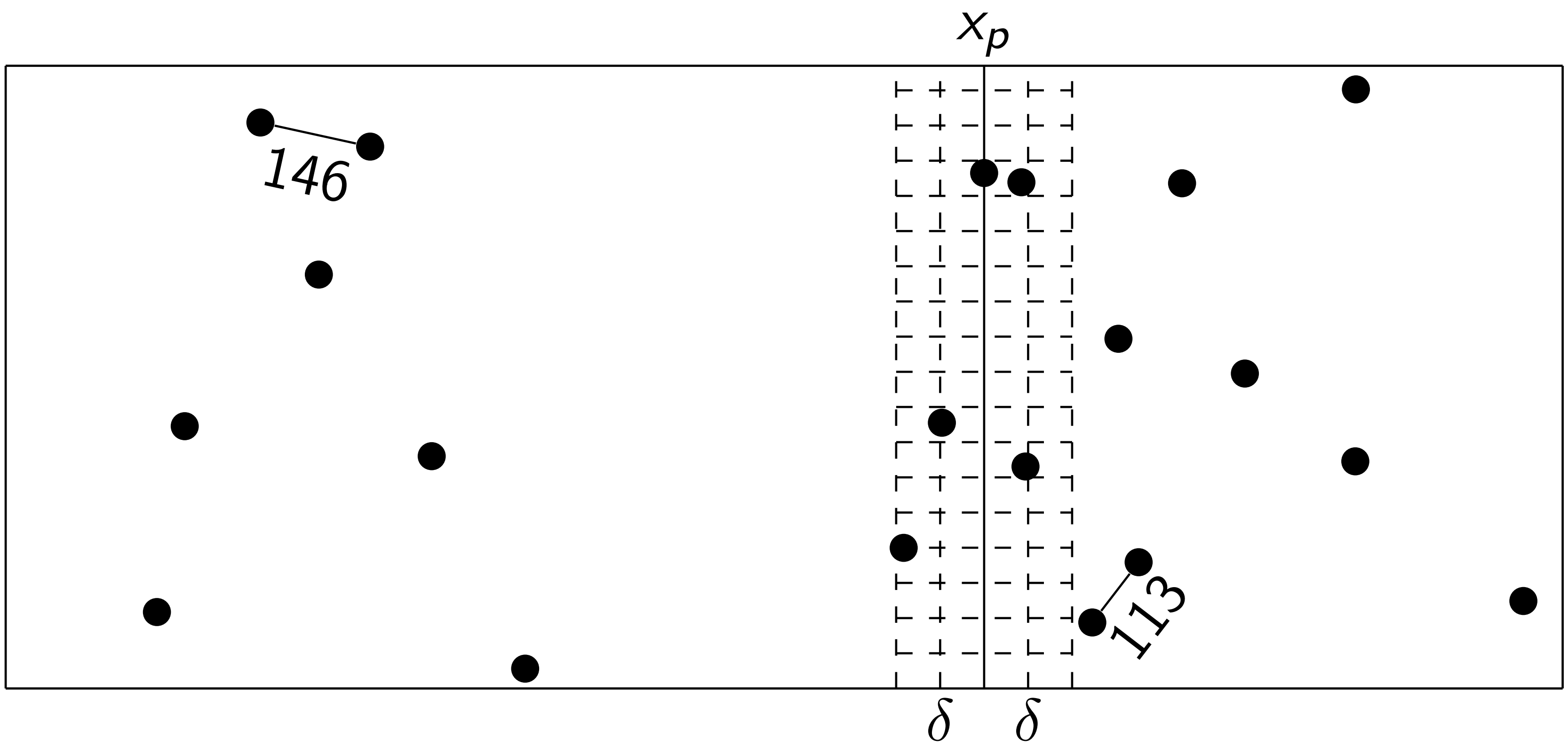
- We cut the plane in two halves with 10 points in each half
- We compute the nearest points in each half
- $\delta = \min(146, 113)$
- We only have to consider points within δ from the vertical line
- If there are none, then δ is the answer
- If there are, then they must be checked with points from the other side which also must be within δ from the vertical line, of course

Combining



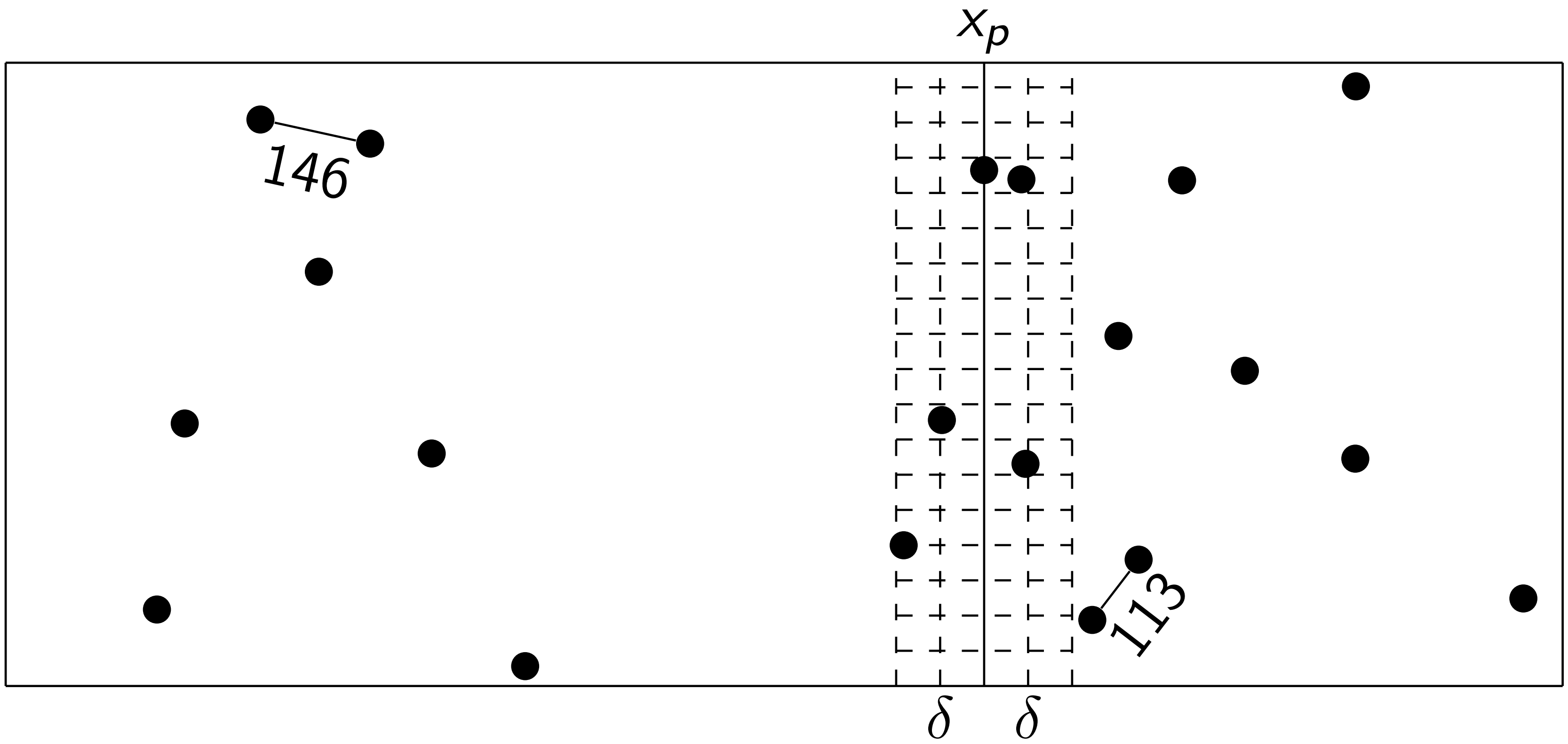
- The point p on the vertical line x_p belongs to the left half but there could also be points in the right half with the same x-coordinate
- Let the set S consist of all points with a distance within δ from the line x_p, (5 points here)
- Clearly it is sufficient to compare only points q and r from S such that p comes from the left half and q from the right part

Combining



- Each dashed box has a side of $\delta/2$
- How many points can each such box contain at most?
- The diagonal of a dashed box is $\sqrt{2} \times \delta/2 < \delta$
- With two points in a dashed box, their distance would be less than δ so at most one point

Combining



- With at most one point per dashed box, we can do as follows.
- Let S be sorted on y-coordinates
- Each point $p \in S$ is inspected at a time.
- The distances from p to each of the next 6 points on the other side in S (according to y-coordinates) are checked to see if it less than the shortest distance found so far

Algorithm outline

- What do we need for this?
- Input is a set of n points P
- We produce two sorted arrays P_x and P_y before starting our recursion
- We divide P_x into two arrays L_x and R_x (left and right)
- We divide P_y into two arrays L_y and R_y
- We solve the two subproblems $(L_x, L_y, n/2)$ and $(R_x, R_y, n/2)$
- Then we compute δ as the minimum from these subproblems
- Then we create the set S_y from P_y
- All dividing and combining can be done in linear time, so we solve this in $\Theta(n \log n)$ time