The algorithm that we obtain is as follows:

**Closest-Pair (P)**

1. If \( |P| \leq 3 \), directly compute closest pair and return.
2. Sort \( P \) by \( x \)-coordinates. Let sorted order be \( P_1, P_2, \ldots, P_{\frac{n}{2}}, P_{\frac{n}{2}+1}, \ldots, P_n \).
3. Let \( P_1 = \{P_1, P_2, \ldots, P_{\frac{n}{2}}\} \)
   \( P_2 = \{P_{\frac{n}{2}+1}, \ldots, P_n\} \)

Recursively, \( (p_1, q_1) \leftarrow \text{Closest-Pair} (P_1) \)
\( (p_2, q_2) \leftarrow \text{Closest-Pair} (P_2) \)
\( s \leftarrow \min \{d(p_1, q_1), d(p_2, q_2)\} \)

Let \( a \) denote \( P_{\frac{n}{2}} \cdot x \).

- Compute \( P_1' \leftarrow \{ p \in P_1 \mid a - s \leq p \cdot x \leq a \} \)
- Compute \( P_2' \leftarrow \{ q \in P_2 \mid a \leq q \cdot x \leq a + s \} \)
- \( E \leftarrow P_1' \cup P_2' \)
Sort $E$ by y-coordinates.

$(P_3, q_3) \leftarrow$ Closest pair among all pairs in $E$ that are at most 8 locations apart.

Return closest among $(P_1, q_1)$, $(P_2, q_2)$, and $(P_3, q_3)$.

---

Besides the recursive calls, most expensive steps are the two sorting steps, which take $O(n \log n)$ time, where $n = |P|$. 

Let $T(n)$ denote worst-case running time of algorithm on $n$ points. We have

$$T(n) \leq T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + c_1 n \log_2 n \quad \text{if } n > 3$$

$$T(n) \leq c_2 \quad \text{if } n \leq 3.$$
As we'll see in class, this implies that \( T(n) = O(n \log^2 n) \). This is a substantial improvement over the \( O(n^2) \) algorithm.

The running time can be improved by eliminating the 2 sorting steps. For the first sorting step, we can pre-sort the input point set by \( x \)-coordinate. If \( P \) is sorted by \( x \)-coordinate, we can generate \( P_1 \) and \( P_2 \) in \( O(n) \) time so that they are sorted by \( x \)-coordinate.

We can also eliminate the second sorting step by pre-sorting the input along \( y \)-coordinates. But more care is needed here, read the algorithm in the textbook.
Another Illustration of Divide-and-Conquer

Given an array \( A[1...n] \) consisting of integers (positive or negative), find the maximum sum of any contiguous subarray.

Example:

\[
81 \ -41 \ 59 \ 26 \ -53 \ 58 \ 97 \ -93 \ -23 \ 84
\]

Solution is

\[
59 + 26 + (-53) + 58 + 97 = 187
\]

Formally, define

\[
\text{Sum} [i,j] \quad \text{to be} \quad A[i] + \ldots + A[j]
\]

for \( 1 \leq i \leq j \leq n \).

We want maximum value of \( \text{Sum} [i,j] \)

over all \( i,j \) such that \( 1 \leq i \leq j \leq n \).
There is an easy \( O(n^2) \) algorithm:

\[
\text{best} \leftarrow -\infty
\]

For \( i \leftarrow 1 \) to \( n \) do

\[
\text{Sum}[i,i] \leftarrow A[i]
\]

if \( \text{Sum}[i,i] > \text{best} \) then \( \text{best} \leftarrow \text{Sum}[i,i] \)

For \( j \leftarrow i+1 \) to \( n \) do

\[
\text{Sum}[i,j] \leftarrow \text{Sum}[i,j+1] + A[j]
\]

if \( \text{Sum}[i,j] > \text{best} \) then

\[
\text{best} \leftarrow \text{Sum}[i,j]
\]

end for

end for

Return best.

Can we do better?

Let's try divide-and-conquer.
Suppose that
\[ A[i_1] \ldots A[i_2] \] is best solution.

\[ \text{if } j_1 \leq \left\lfloor \frac{n}{2} \right\rfloor , \]
\[ A[i_1] \ldots A[i_2] \] is best solution for input \( A[1 \ldots \left\lfloor \frac{n}{2} \right\rfloor] \)

\[ \text{if } i_1 > \left\lfloor \frac{n}{2} \right\rfloor , \]
\[ A[i_1] \ldots A[i_2] \] is best solution for input \( A[\left\lceil \frac{n}{2} \right\rceil + 1, \ldots, n] \)

The third case is \( \frac{n}{2} \leq J_1 < J_1 \).

In this case
\[ A[i_1] \ldots A[\left\lfloor \frac{n}{2} \right\rfloor] \] is best subarray ending at \( \left\lfloor \frac{n}{2} \right\rfloor \), and
\[ A[\left\lfloor \frac{n}{2} \right\rfloor + 1] \ldots A[3_2] \] is best subarray that starts at \( \left\lfloor \frac{n}{2} \right\rfloor + 1 \).
The first two cases are handled by recursion, and the third in $O(n)$ time.

$\text{Best } (A, n)$

if $n \leq 3$, compute directly and return.

Otherwise, let $A_1$ be a way consisting of first $\lceil \frac{n}{2} \rceil$ elements of $A$.
let $A_2$ be a way consisting of remaining $\lfloor \frac{n}{2} \rfloor$ elements of $A$.

$(i_1, j_1) \leftarrow \text{Best } (A_1, \lceil \frac{n}{2} \rceil)$
$(i_2, j_2) \leftarrow \text{Best } (A_2, \lfloor \frac{n}{2} \rfloor)$

let $(i, j \lceil \frac{n}{2} \rceil)$ be best subarray ending at $A[\lceil \frac{n}{2} \rceil]$. Can be computed in $O(n)$ time.

let $(\lceil \frac{n}{2} \rceil + 1, j_3)$ be best subarray starting at $A[\lceil \frac{n}{2} \rceil + 1]$. 
Return best among $(i_1, j_1), (i_2, j_2), (i_3, j_3)$

Recurrence for worst-case running time:

\[ T(n) \leq T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + c_1n \]
if \( n > 3 \)

\[ T(n) \leq c_2 \]
if \( n \leq 3 \),

where \( c_1, c_2 > 0 \) are constants.

This implies \( T(n) \) is \( O(n \log n) \).