33.2-5

left-to-right issue

\[ \begin{array}{c}
A \\
\hline
C \\
D \\
\hline
B \\
\end{array} \]

Finds the A-B intersection first, which is to the right of the C-D intersection.

All intersections issue

a) Counting: The for-loop in Any-Segments-Intersect executes \(2n\) times for \(n\) segments. Each iteration does either two intersection tests (left endpoint) or one such test (right endpoint). So \(3n\) tests in all, but at most \(3n\) intersections can be reported. But there can be up to \((n^2-n)/2\) intersections of \(n\) segments. So some must missed.

Particular example:

\[ \begin{array}{c}
A \\
\hline
B \\
C \\
\end{array} \]

B always remains below A on the sweep line, so A and C never "see" each other to get tested for intersection.

33-3 a) Start with a point \(p\) on the convex hull (such as min y point). Assume \(p\) is a ghost. (The ghostbuster case works the same.) Sort the other points by increasing polar angle with \(p\). Let the first point in this order be \(q\). If \(q\) is a ghostbuster, we have the property we want:

There are \(n-1\) ghosts and \(n-1\) ghostbusters to the left of the \(p-q\) line. (Same works if last point in order is GB.)

If \(q\) is a ghost, too, then consider the line from \(p\) to \(r\), where \(r\) is the first ghostbuster in order.

There are 0 ghostbusters and \(C > 0\) ghosts to the right of
the p-r line (since at least q is included). So more ghosts than ghostbusters to the right. The line from p to s, where s is the last ghostbuster in polar-angle order, has n-1 ghostbusters to the right and n-1-d ghosts to the right (since the last point in order is a ghost). So more ghostbusters than ghosts. So there is some point t in between r and s where there are the same # of ghosts and ghostbusters to the right. This process is O(n lg n), since it is a polar-angle sort, plus a linear scan.

3 ghosts
5 ghostbusters to right

\[ \hole = \text{ghost} \quad \& = \text{ghostbuster} \]

b) Use the previous result for divide-and-conquer. Find the partitioning line, and have the ghost shoot at the ghostbuster. Then solve left and right recursively. In the worst case, we have to repeat the process n times, because all the points land on one side of the line. So

\[ O(n \cdot n \lg n) = O(n^2 \lg n) \] worst case complexity.
5A. \text{RFFT}(0, 1, 2, 3)
\begin{itemize}
  \item \text{y[0]} = \text{RFFT}(0, 2)
    \begin{itemize}
      \item \text{y[0]} = \text{RFFT}(0) = 0
      \item \text{y[1]} = \text{RFFT}(2) = 2
    \end{itemize}
  \text{return } (0+2(i), 0-2(i)) = (2, -2)
  \item \text{y[1]} = \text{RFFT}(1, 3)
    \begin{itemize}
      \item \text{y[0]} = \text{RFFT}(1) = 1
      \item \text{y[1]} = \text{RFFT}(3) = 3
    \end{itemize}
  \text{return } (1+3(1)^2, 1-3(1)) = (4, -2)
\end{itemize}
\text{return } (2+4, -2+2i, 2-4, -2-2i) = (6, -2-2i, -2, -2+2i)

5B. We can merge the lists for different letters in a pattern. We need to offset the positions in the list for a letter by the position of the letter in the pattern.
Suppose the pattern is 'test'.

So we merge
\begin{align*}
  t - \text{list (no offset)} &= 1, 11, 14, 16, 19 \\
  e - \text{list (-1 offset)} &= 11, 16 \\
  s - \text{list (-2 offset)} &= 2, 5, 11 \\
  t - \text{list (-3 offset)} &= 8, 11, 13, 16
\end{align*}
\text{intersect} = \begin{array}{c}
  \text{merge} = 11 \\
  \text{pattern appears at position 11.}
\end{array}

5C. For each point, sort the other points by polar angle around that point. Go through the list, and use X-product to see if the point plus any two consecutive points on the list are in a row.
\text{O(n log n) for the polar-angle sorting. To run through n points as the sorting origin is } \text{O(n^2 log n).}
5D. The idea is to determine the Hamiltonian path by deleting edges from \( G \) as long as they don’t destroy the Hamiltonian path. (More accurately, we remove an edge if there is some Hamiltonian path that doesn’t use it.)

Let \( \text{HP} \) be a polynomial-time algorithm that decides Ham-Path.

1. If \( \text{HP}(G,v,w) = \text{false} \) return "nocycle"

2. For each edge \( e \) in \( G \)
   - let \( G' \) be \( G \) with \( e \) removed
   - if \( \text{HP}(G',v,w) = \text{true} \), then \( G \leftarrow G' \)

3. The edges in \( G \) form a path from \( v \) to \( w \). The order of the nodes can be found, for example, with DFS.