Algorithms Lab

CSE Department, IIT Kharagpur

Autumn 2025

A4: Intersections of permutation lines — 12-Aug-2025

Consider two parallel lines: $L := y = 25$ and $L' := y = 125$.
Suppose there are n points uniformly placed on each of them such that:

1. The leftmost point on either line have $x = 25$.
2. The i-th point from the left on each line has the same x-coordinate: $x = 25i \ \forall\ i \in [1, n]$.
3. The points on L are labeled 1 through n from left to right.
4. The points on L' are also labeled by 1 to n, but in an arbitrary order.

For each label λ, consider the pair of points (one on L and one on L') having that label. We thus have n such pairs and n corresponding line segments, referred to as **permutation lines**. Determine the number of intersections among these permutation lines.

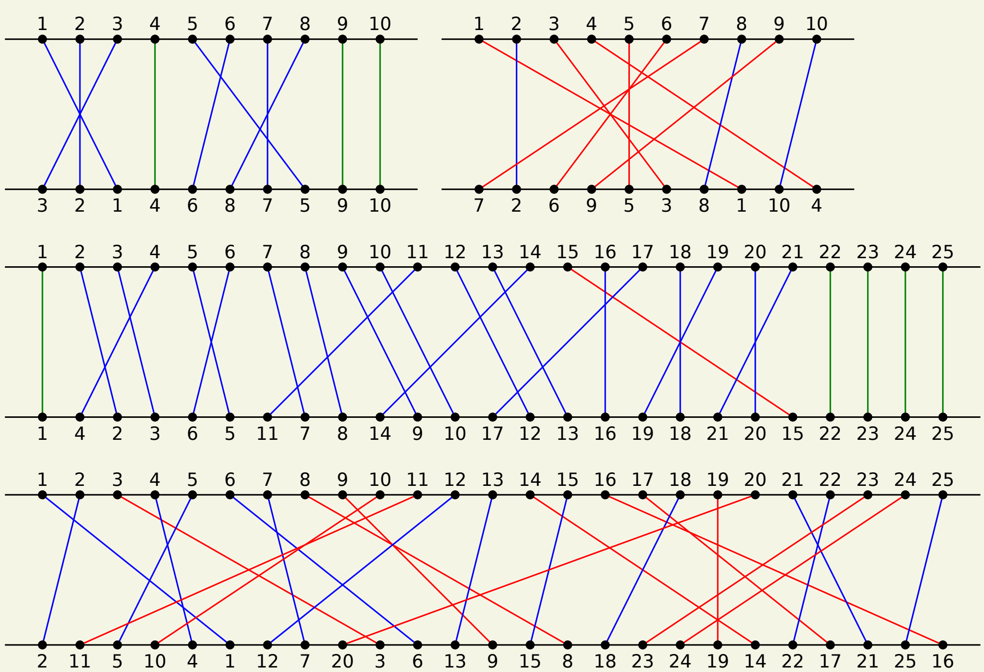
Part A: Design and implement an $\mathcal{O}(n^2)$-time algorithm. It should take n as user input, place the points uniformly on L and L', and print to the terminal all segment pairs that intersect, along with the total number of intersections. $\boxed{\text{10 marks}}$

As a visual output, it should also create an `SVG` file. Color each segment as follows: m5

- No intersections: `green`
- At most $\lceil \log_2 n \rceil$ intersections: `blue`
- More than $\lceil \log_2 n \rceil$ intersections: `red`

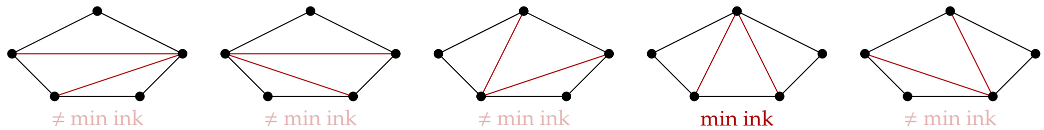
Part B: Design and implement an $\mathcal{O}(n \log n)$-time algorithm. Follow the same input/output conventions as in Part A. This algorithm is analogous to counting inversions. $\boxed{\text{15 + 5 marks}}$

Example 0.1 (Intersections of permutation lines)



A4*: Minimum-ink Triangulation (Take home) 16-Aug-2025

Triangulation of a convex polygon P with n vertices is a partition of the polygon into triangles, using non-crossing diagonals. Our goal is to triangulate P by a pen, spending minimum ink, which we refer to as **minimum-ink triangulation**.



For a convex pentagon, there are just five triangulations, as shown above, the 4th one taking minimum ink. As triangulation-count shoots up exponentially with n, minimum-ink triangulation looks difficult. For example, for a decagon ($n = 10$), it is 1430, while for a 20-gon, it leaps to 656,412,042.

Solution

Any triangulation splits the polygon into $n-2$ triangles using $n-3$ non-crossing diagonals. The number of triangulations being exponential in n, we use dynamic programming to find the solution in polynomial time. We assume that the vertices are given in counterclockwise order.

Definitions:

- $P(i,j) :=$ subpolygon with vertices $i, i+1, \ldots, j$.
- $\pi(\cdot) :=$ perimeter.
- $\pi(i,k,j) :=$ perimeter of $\triangle ikj$.
- $f(i,j) :=$ minimum ink needed to triangulate $P(i,j)$.

Observation 1: In any triangulation $\mathcal{S}$:

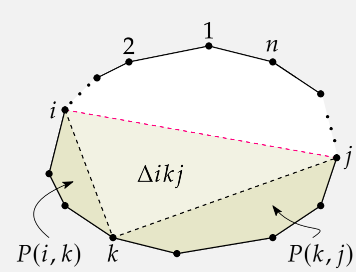
- Every edge of P appears in exactly one triangle $T \in \mathcal{S}$.
- Every diagonal $\mathbf{d} \in \mathcal{S}$ is shared by exactly two triangles.

Hence,

$$\sum_{T \in \mathcal{S}} \pi(T) = \pi(P) + 2 \cdot \sum_{\mathbf{d} \in \mathcal{S}} \mathbf{d}.$$

As $\pi(P)$ is fixed, we revise our goal to: Minimize: $\sum_{T \in \mathcal{S}} \pi(T)$.

Observation 2: For any subpolygon $P(i,j)$, where $j \geq i+2$ (indices taken modulo n), there always exists a minimum-cost triangulation that includes $\triangle ikj$ for some vertex k lying strictly between i and j in counterclockwise order. This triangle splits $P(i,j)$ into two smaller subpolygons, $P(i,k)$ and $P(k,j)$, which can then be recursively triangulated optimally—this leads naturally to a dynamic programming formulation.



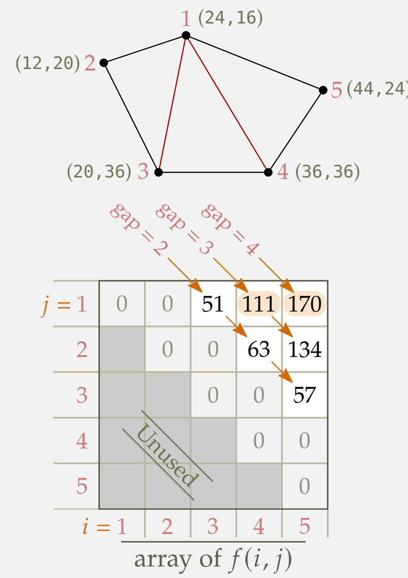
Optimal substructure:

- $j - i \geq 2 \;\Rightarrow\; f(i,j) = \min_{i<k<j} \{f(i,k) + f(k,j) + \pi(i,k,j)\}$
- $j - i < 2 \;\Rightarrow\; f(i,j) = 0$ (no triangle).

Overlapping subproblems: The same subpolygon $P(i,j)$ may arise in multiple recursive decompositions. Hence, memoization or bottom-up DP is used to avoid redundant recomputation.

Algorithm 1: Minimum-ink Triangulation

```
for i ← 1 to n do
    f(i,i) ← 0
    f(i,(i+1) mod n) ← 0
for gap ← 2 to n − 1 do
    for i ← 1 to n − gap do
        j ← i + gap
        f(i,j) ← ∞
        for k ← i + 1 to j − 1 do
            cost ← f(i,k) + f(k,j) + π(i,k,j)
            if cost < f(i,j) then
                f(i,j) ← cost
                trace(i,j) ← k
return f(1,n)
```



Computational steps to fill up the DP table

gap	i	j	k	$\pi(i,k,j)$	$f(i,j)$	trace(i,j)
2	1	3	2	$f(1,2)+f(2,3)+\pi(1,2,3) = 0+0+112 = 112$	51	2
	2	4	3	$f(2,3)+f(3,4)+\pi(2,3,4) = 0+0+128 = 128$	63	3
	3	5	4	$f(3,4)+f(4,5)+\pi(3,4,5) = 0+0+96 = 96$	57	4
3	1	4	2	$f(1,2)+f(2,4)+\pi(1,2,4) = 0+63+65 = 128$		
			3	$f(1,3)+f(3,4)+\pi(1,3,4) = 51+0+60 = 111$	111	3
	2	5	3	$f(2,3)+f(3,5)+\pi(2,3,5) = 0+57+77 = 134$	134	3
			4	$f(2,4)+f(4,5)+\pi(2,4,5) = 63+0+76 = 139$		
4	1	5	2	$f(1,2)+f(2,5)+\pi(1,2,5) = 0+134+66 = 200$		
			3	$f(1,3)+f(3,5)+\pi(1,3,5) = 51+57+69 = 177$		
			4	$f(1,4)+f(4,5)+\pi(1,4,5) = 111+0+59 = 170$	170	4

tracing back in linear time

to collect the diagonals $(1,4)$ and $(1,3)$, using trace(i,j)

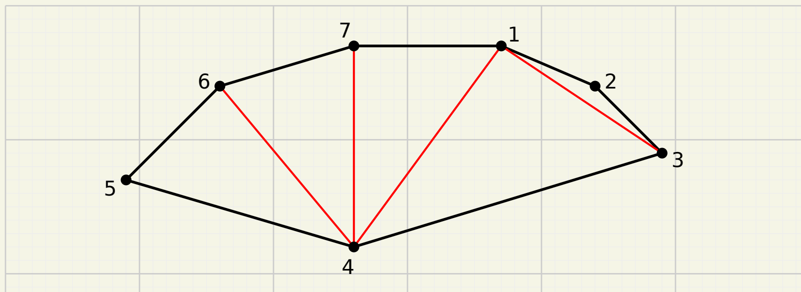
Time and space complexities: There are $\mathcal{O}(n^2)$ distinct subproblems $f(i,j)$ with $i < j$. For each such subproblem, we try all k with $i < k < j$, giving at most n choices. Hence, total time complexity is $\mathcal{O}(n^3)$. The space complexity is $\mathcal{O}(n^2)$ to store the DP table.

Example 0.2 (Minimum-ink Triangulation of convex polygons)

The top-left point has coordinates $(0, 0)$. The $+x$-axis is directed rightward, the $+y$-axis downward.

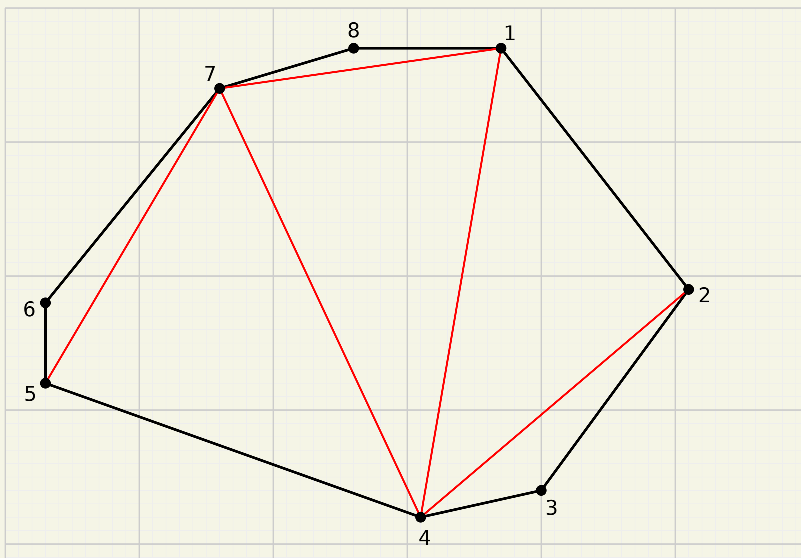
7 vertices:

```
370 30
440 60
490 110
260 180
90 130
160 60
260 30
```



8 vertices:

```
370 30
510 210
400 360
310 380
30 280
30 220
160 60
260 30
```



15 vertices:

```
370 30
440 60
490 110
510 210
450 320
400 360
310 380
200 380
130 365
70 320
30 280
30 220
90 130
160 60
260 30
```