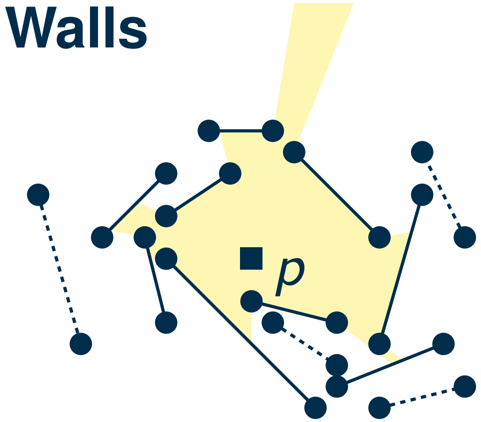
Computational Geometry

Exercise 3 *Assignment 2, 3 and kd-Trees*

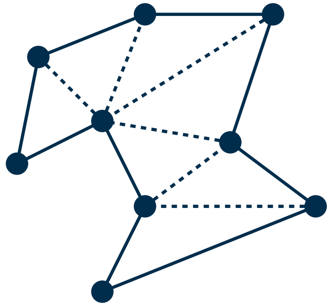
Jean-Pierre, Marcus, Wendy

Assignment 2

Dangerous Walls



Triangulation

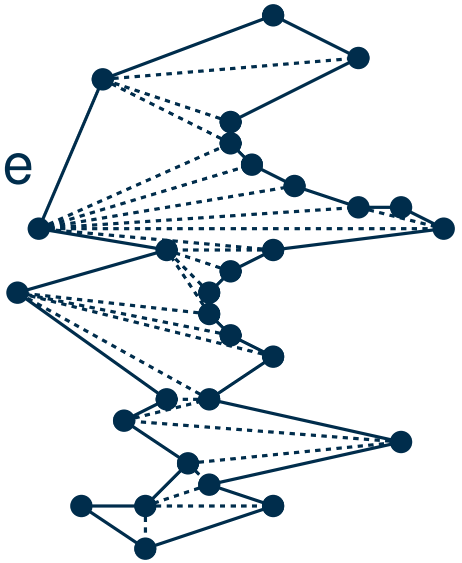


Q: how many triangles?

what if P contains holes?

y-Monotone Triangulation

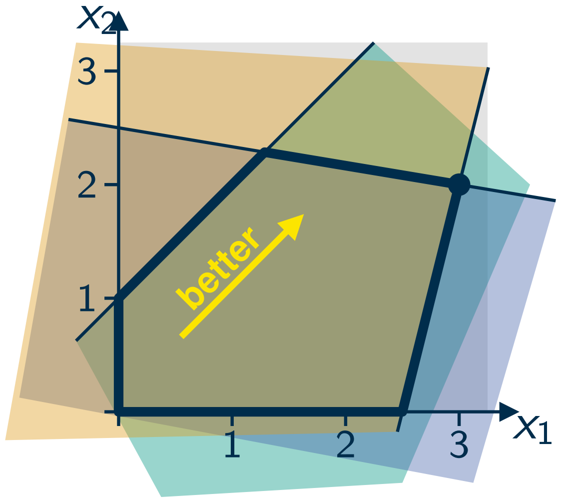
How to triangulate y-monotone Polygon in $O(n)$?



Bounded LP-solution

Given: 2D LP

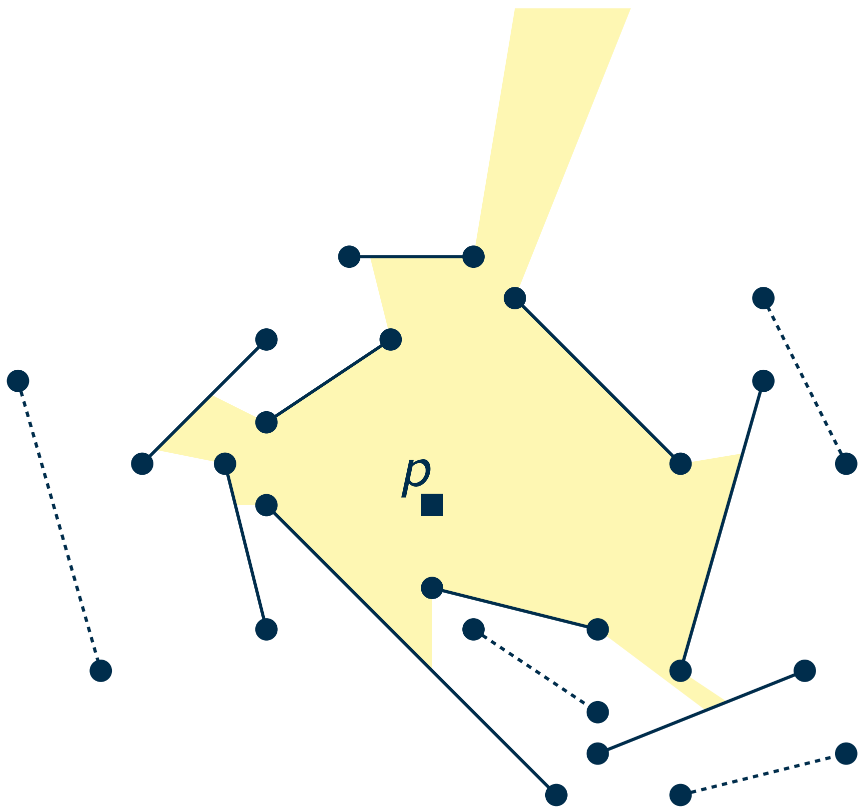
Decide: are there solutions with arbitrarily large objective?



Dangerous Walls

Sweepline:

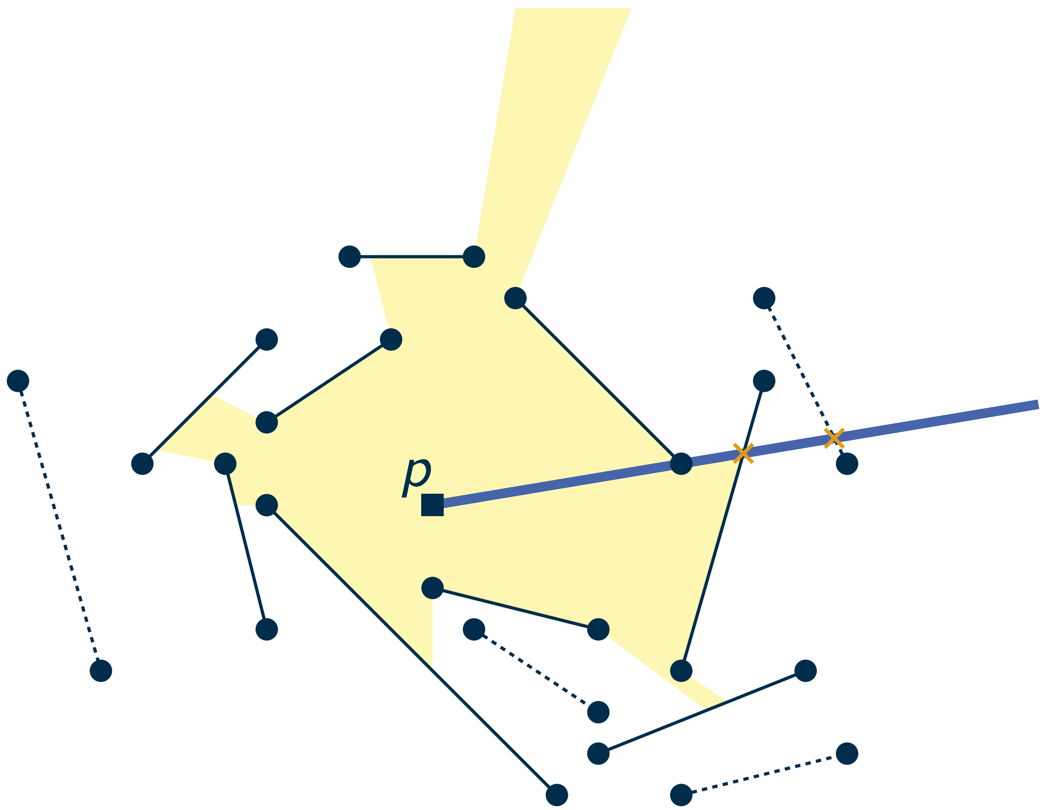
- Events: start and endpoints of segments
- State: Sorted intersections with walls



Dangerous Walls

Sweepline:

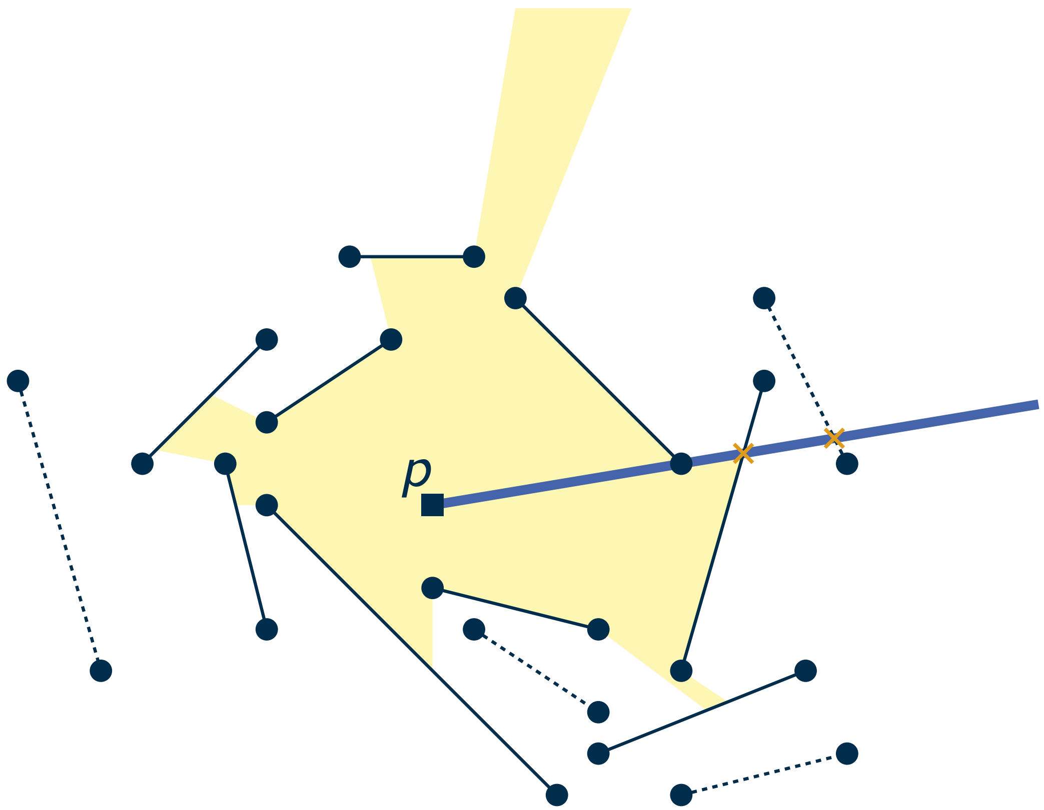
- Events: start and endpoints of segments
- State: Sorted intersections with walls



Dangerous Walls

Sweepline:
- Events: start and endpoints of segments
- State: Sorted intersections with walls

At each event:
- Insert / remove wall from sweepline state
- Mark closest wall



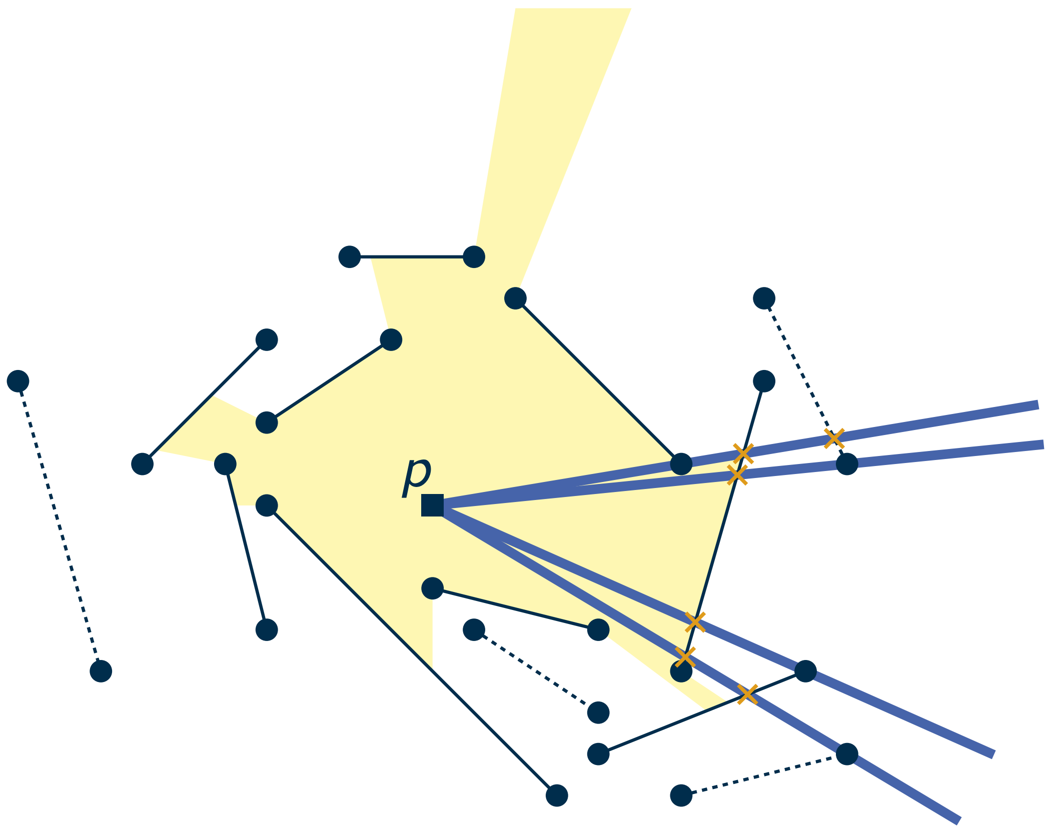
Dangerous Walls

Sweepline:

- Events: start and endpoints of segments
- State: Sorted intersections with walls

At each event:

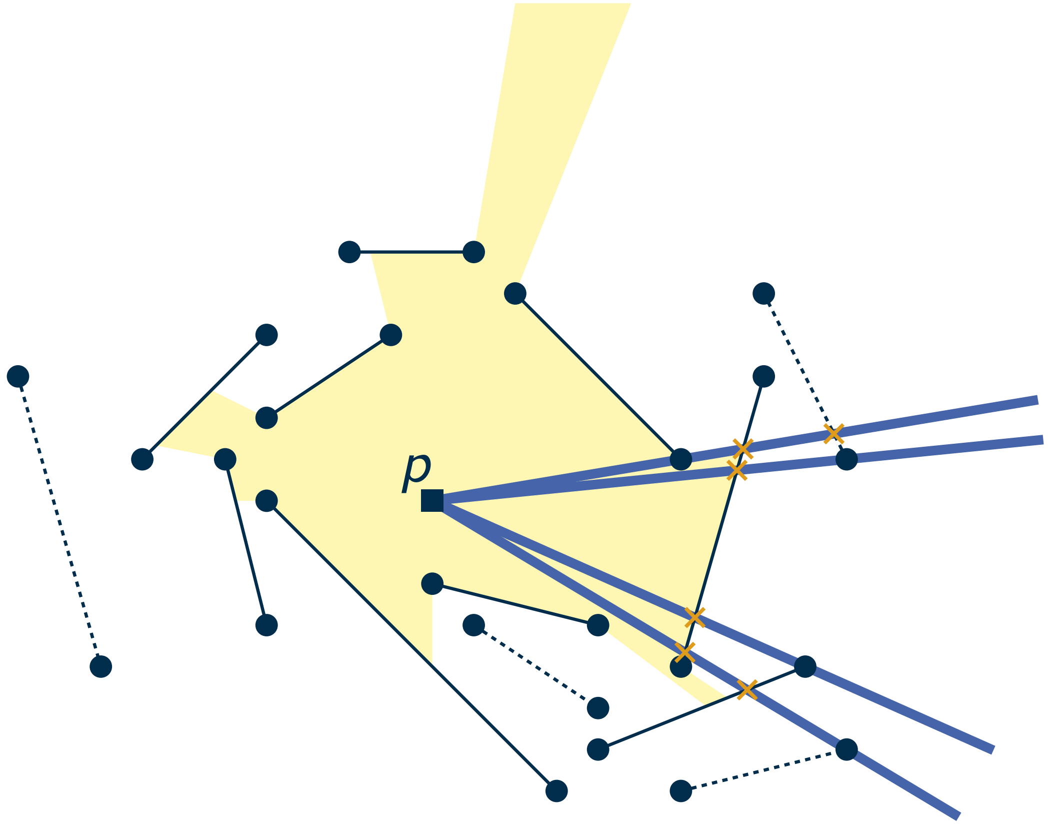
- Insert / remove wall from sweepline state
- Mark closest wall



Dangerous Walls

Sweepline:

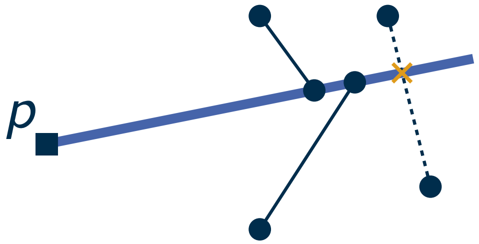
- Events: start and endpoints of segments
- State: Sorted intersections with walls



At each event:

- Insert / remove wall from sweepline state
- Mark closest wall

Special Case:

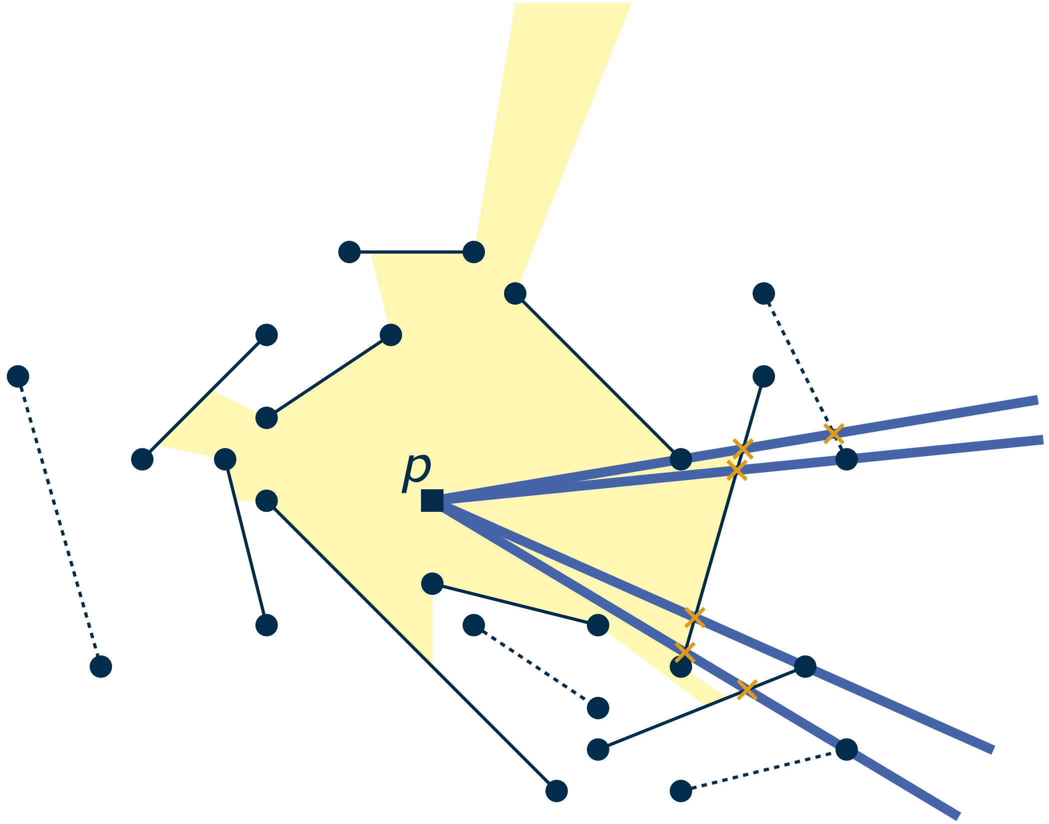


Dangerous Walls

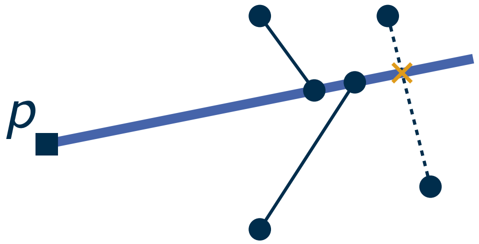
Sweepline:
- Events: start and endpoints of segments
- State: Sorted intersections with walls

Are we storing real numbers in the state
or something else?

At each event:
- Insert / remove wall from sweepline state
- Mark closest wall



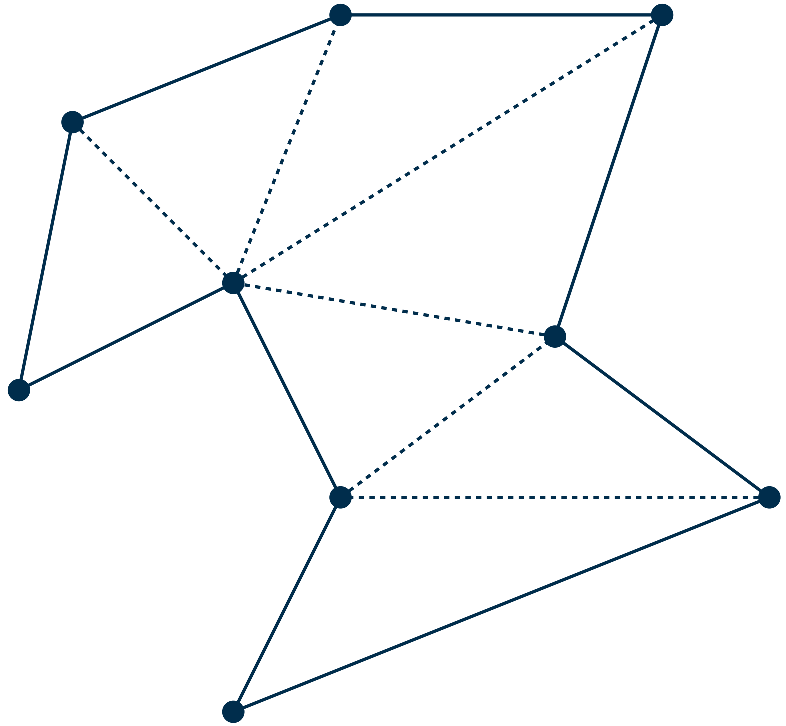
Special Case:



Triangulation

Given Polygon P with n vertices, how many triangles does a triangulation contain?

- $n - 2$
- *Proof*: by induction



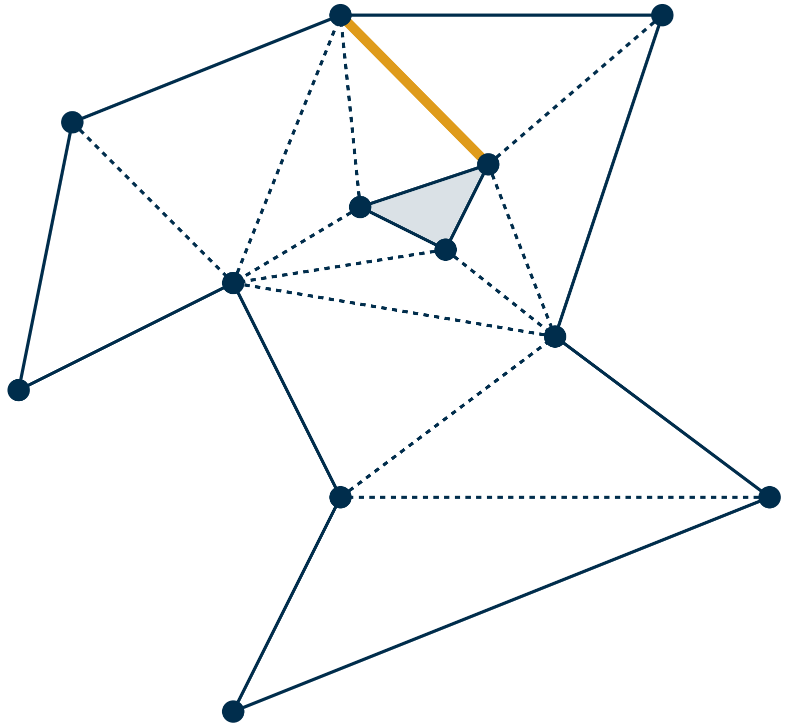
Triangulation

Given Polygon P with n vertices, how many triangles does a triangulation contain?

- $n - 2$
- *Proof*: by induction

Can P be triangulated, if it contains a hole?

- Find edge that connects the hole to P
- Use Lemma from lecture



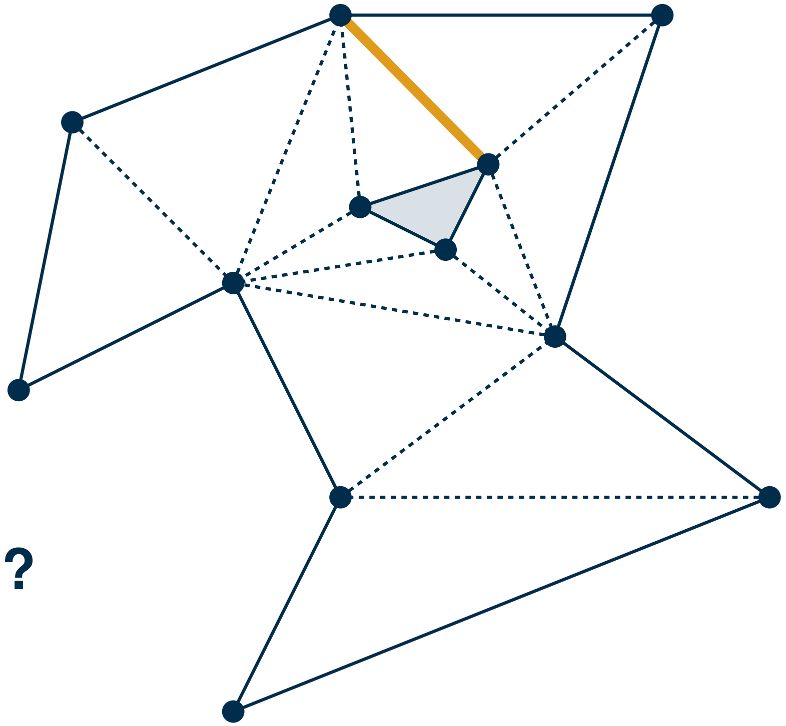
Triangulation

Given Polygon P with n vertices, how many triangles does a triangulation contain?

- $n - 2$
- *Proof*: by induction

Can P be triangulated, if it contains a hole?

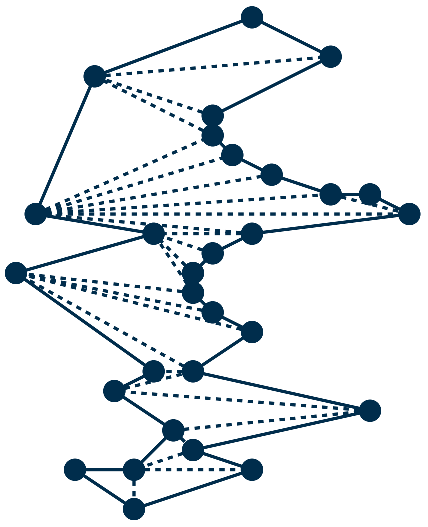
- Find edge that connects the hole to P
- Use Lemma from lecture



How many triangles are there, when P contains k holes? (n includes vertices of holes)

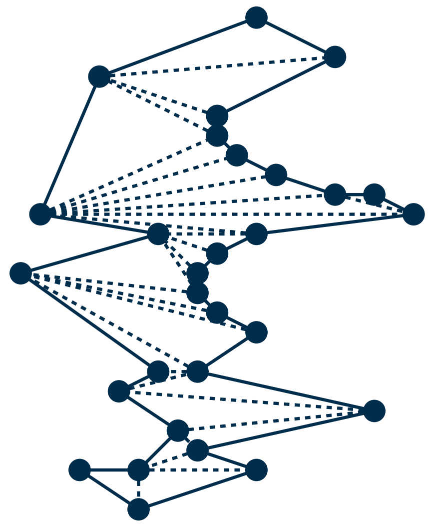
- $n + 2(k - 1)$
- *Proof*: Triangulation is planar graph,
  Euler's formula to count vertices, edges and faces

y-monotone Triangulation



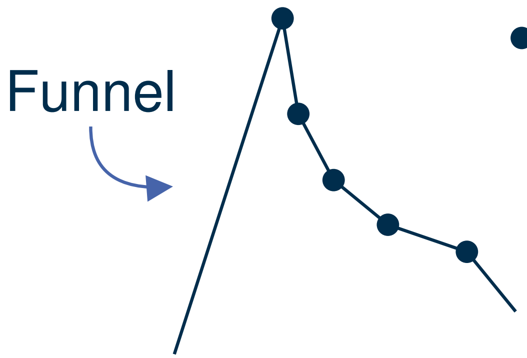
Idea: Iterate over the vertices from top to bottom and greedily add as many edges as possible to vertices above

y-monotone Triangulation

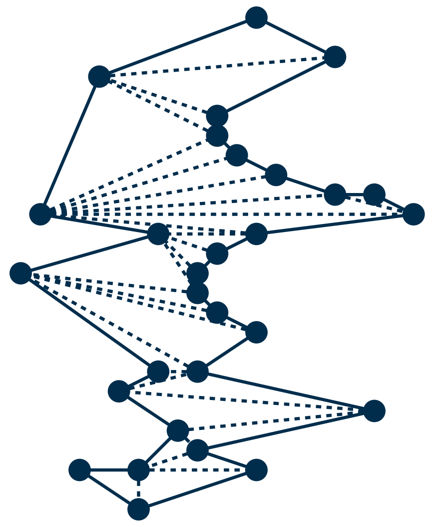


Idea: Iterate over the vertices from top to bottom and greedily add as many edges as possible to vertices above

Invariant: Untriangulated part above last seen vertex forms a “funnel” (ger. Trichter). One side consists of only one edge.

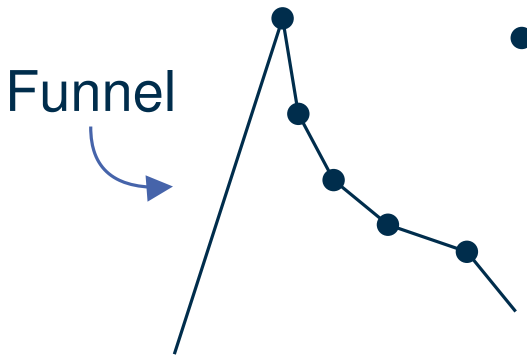


y-monotone Triangulation



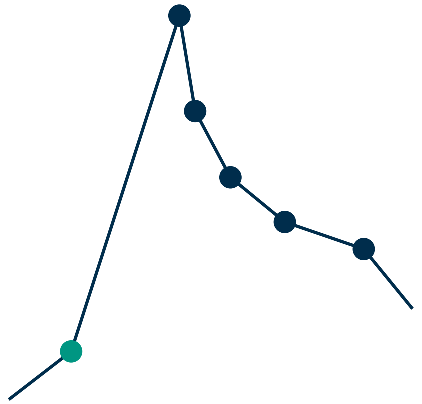
Idea: Iterate over the vertices from top to bottom and greedily add as many edges as possible to vertices above

Invariant: Untriangulated part above last seen vertex forms a “funnel” (ger. Trichter). One side consists of only one edge.

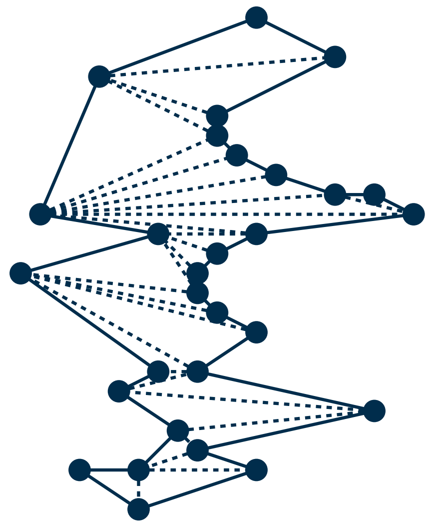


Case 1

Vertex on single edge

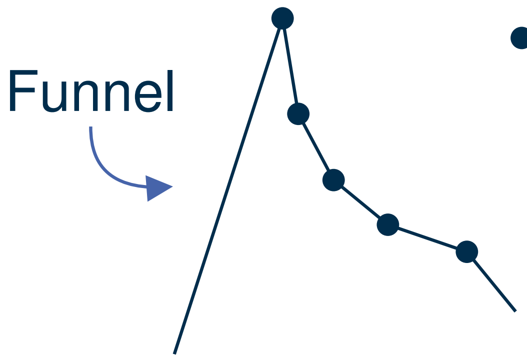


y-monotone Triangulation



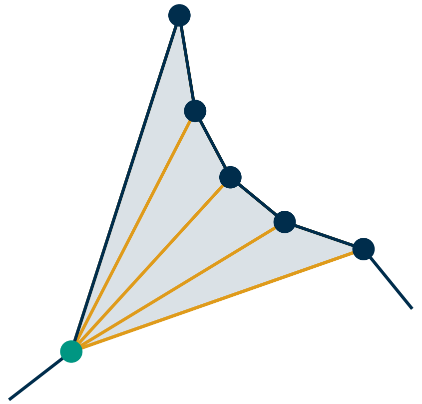
Idea: Iterate over the vertices from top to bottom and greedily add as many edges as possible to vertices above

Invariant: Untriangulated part above last seen vertex forms a “funnel” (ger. Trichter). One side consists of only one edge.



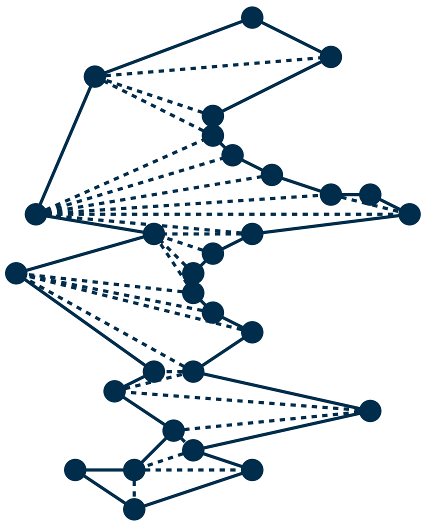
Case 1

Vertex on single edge



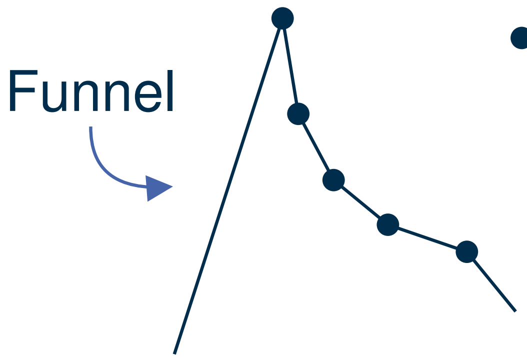
- Connect new vertex with whole funnle

y-monotone Triangulation

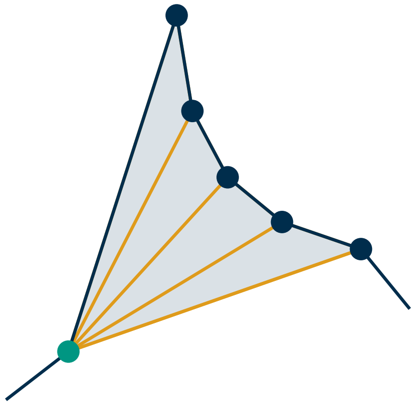


Idea: Iterate over the vertices from top to bottom and greedily add as many edges as possible to vertices above

Invariant: Untriangulated part above last seen vertex forms a “funnel” (ger. Trichter). One side consists of only one edge.

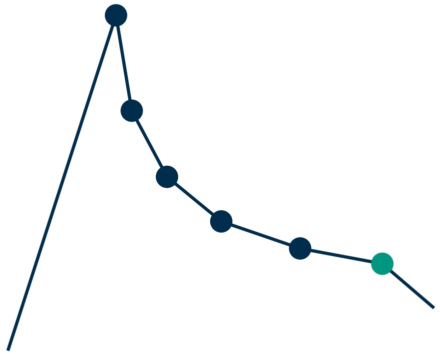


Case 1
Vertex on single edge



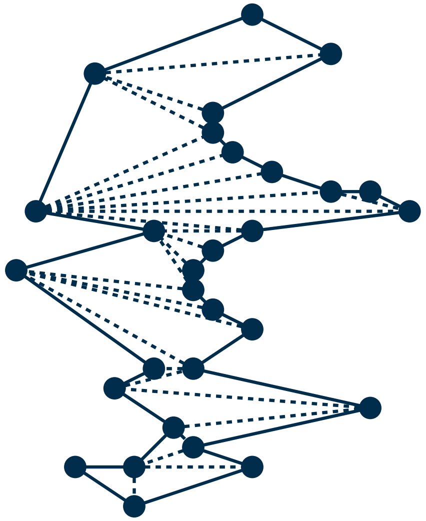
- Connect new vertex with whole funnle

Case 2.1
Vertex on chain side



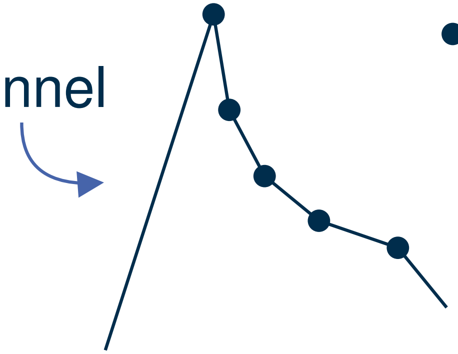
- Nothing to do

y-monotone Triangulation

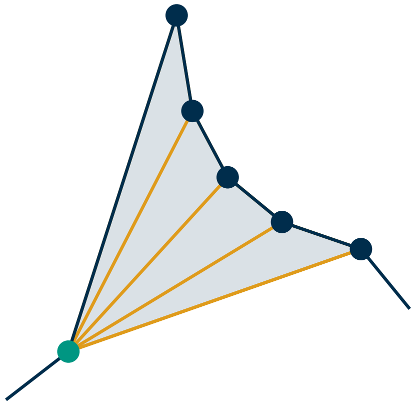


Idea: Iterate over the vertices from top to bottom and greedily add as many edges as possible to vertices above

Invariant: Untriangulated part above last seen vertex forms a “funnel” (ger. Trichter). One side consists of only one edge.

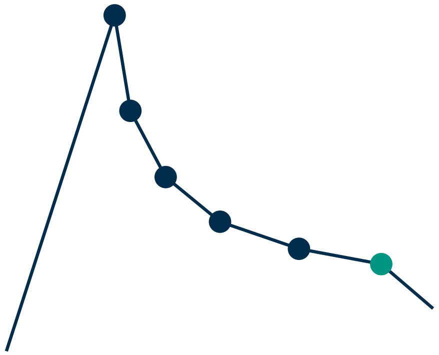


Case 1
Vertex on single edge



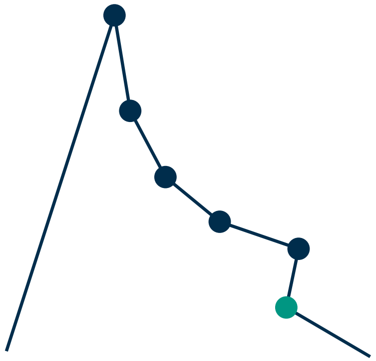
- Connect new vertex with whole funnle

Case 2.1
Vertex on chain side

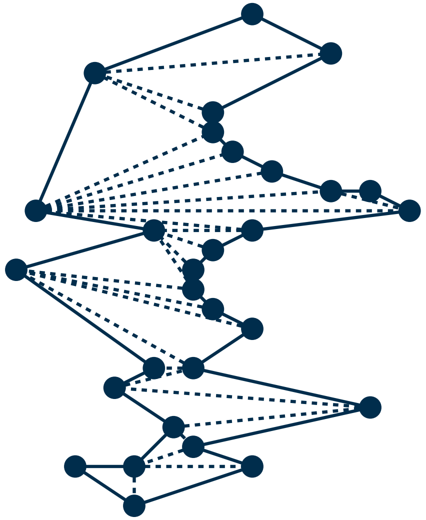


- Nothing to do

Case 2.2
Vertex on chain side



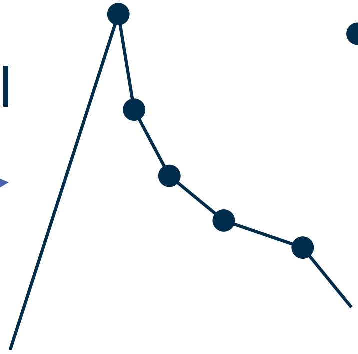
y-monotone Triangulation



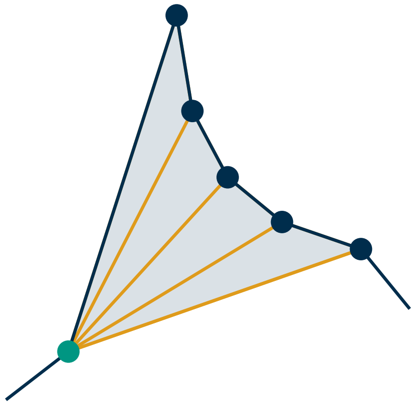
Idea: Iterate over the vertices from top to bottom and greedily add as many edges as possible to vertices above

Invariant: Untriangulated part above last seen vertex forms a “funnel” (ger. Trichter). One side consists of only one edge.

Funnel

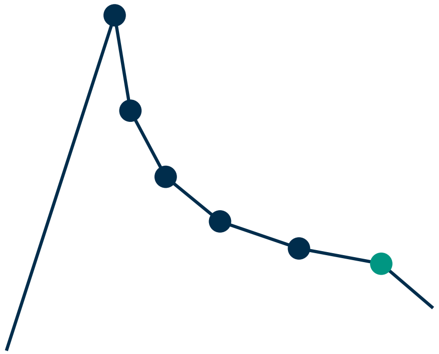


Case 1
Vertex on single edge



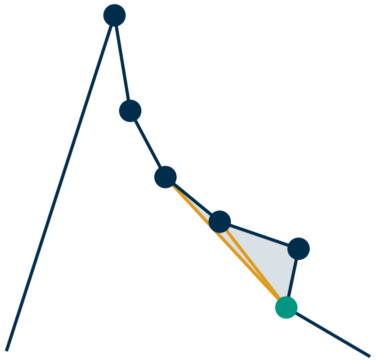
- Connect new vertex with whole funnle

Case 2.1
Vertex on chain side



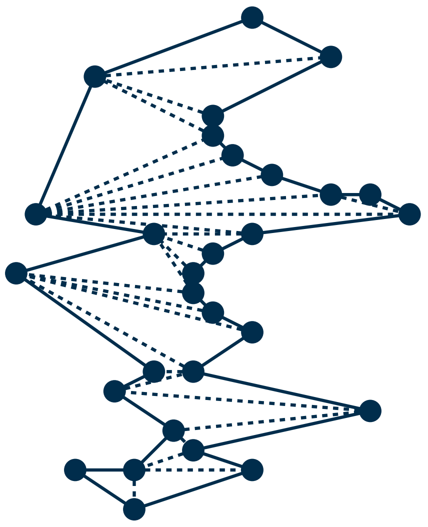
- Nothing to do

Case 2.2
Vertex on chain side



- Connect with as much of chain as possible

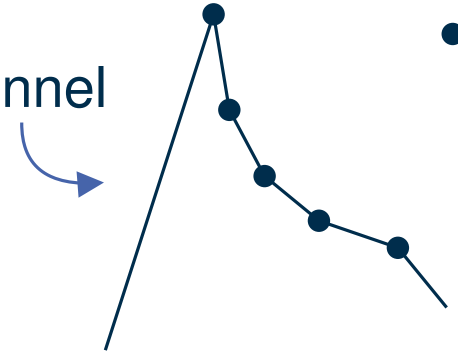
y-monotone Triangulation



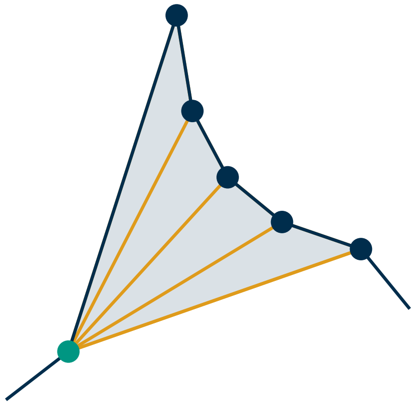
Idea: Iterate over the vertices from top to bottom and greedily add as many edges as possible to vertices above

Invariant: Untriangulated part above last seen vertex forms a “funnel” (ger. Trichter). One side consists of only one edge.

Funnel

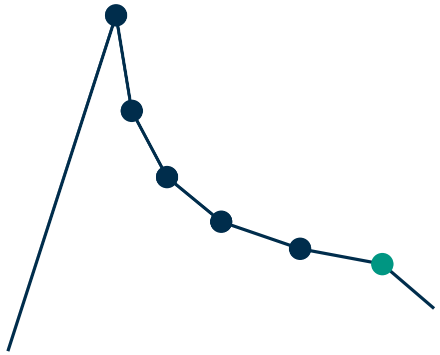


Case 1
Vertex on single edge



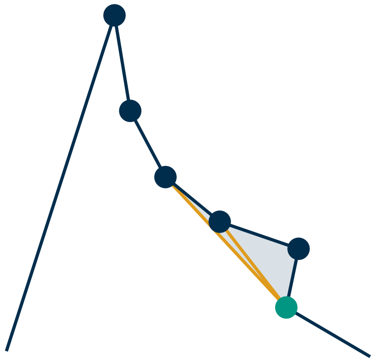
- Connect new vertex with whole funnle

Case 2.1
Vertex on chain side



- Nothing to do

Case 2.2
Vertex on chain side



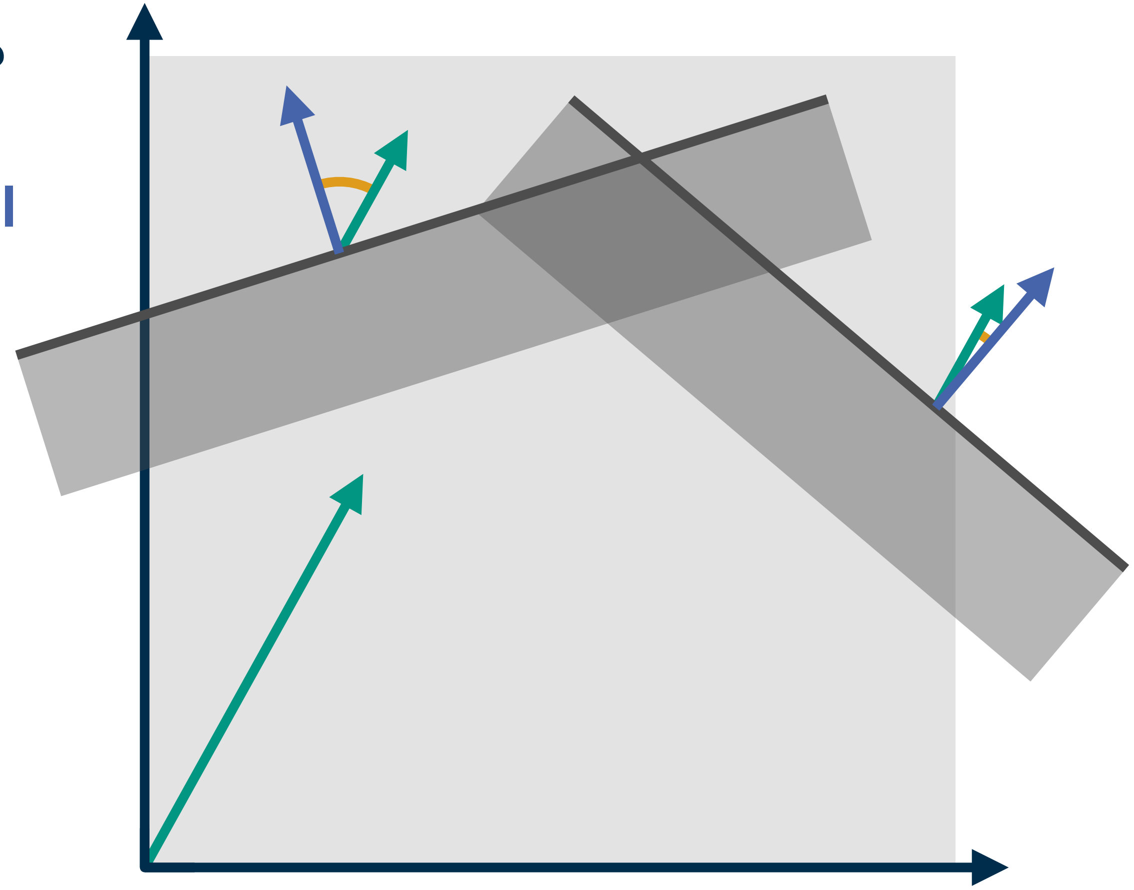
- Connect with as much of chain as possible

Implementation
- Store vertices on stack
- Stack is sorted by y
- Add edges by popping vertices from stack

Bounded LP-solution

Given $2d$ LP: is an optimal solution bounded?

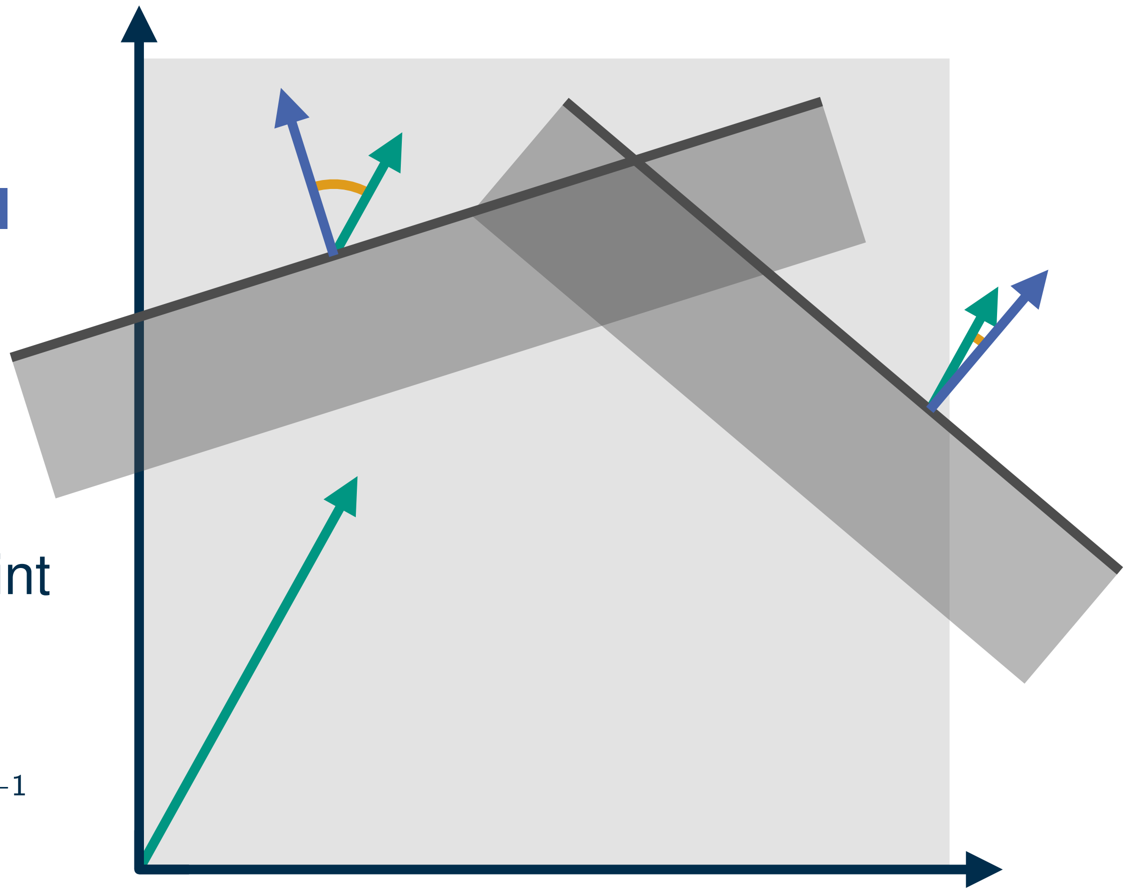
- It suffices to find at most two half planes
- Calculate angle between **half plane normal** and **maximization vector**



Bounded LP-solution

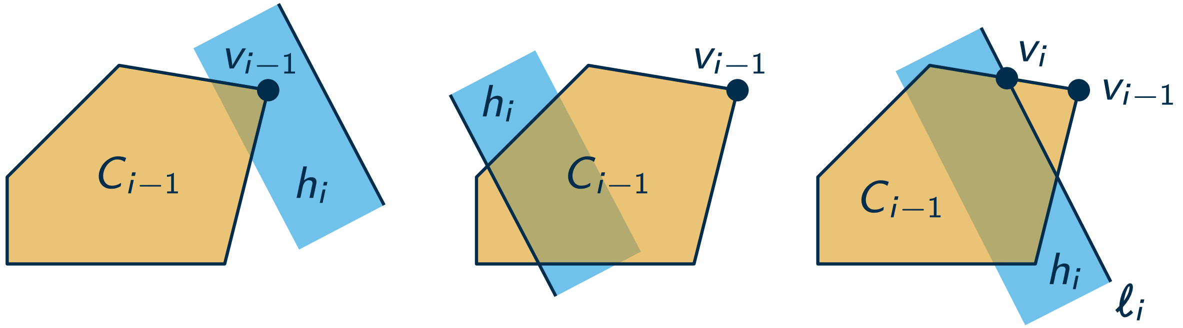
Given 2d LP: is an optimal solution bounded?

- It suffices to find at most two half planes
- Calculate angle between **half plane normal** and **maximization vector**



LP Algorithm from Lecture:

- Iterate over halfplanes (in some order); update best point



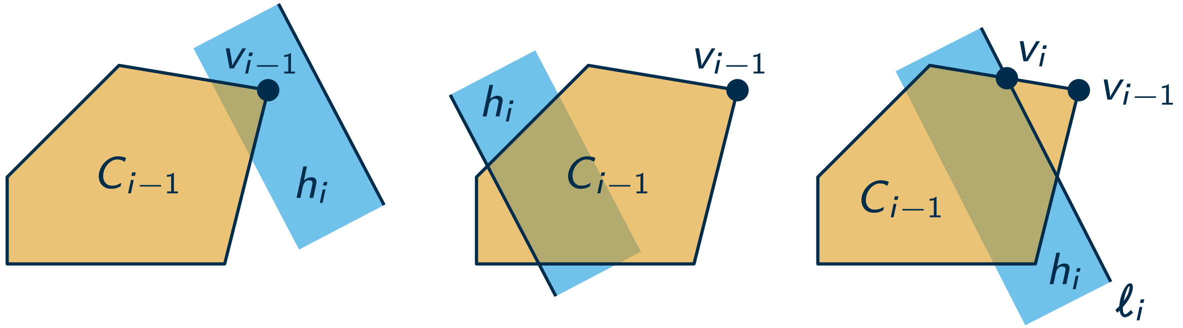
Bounded LP-solution

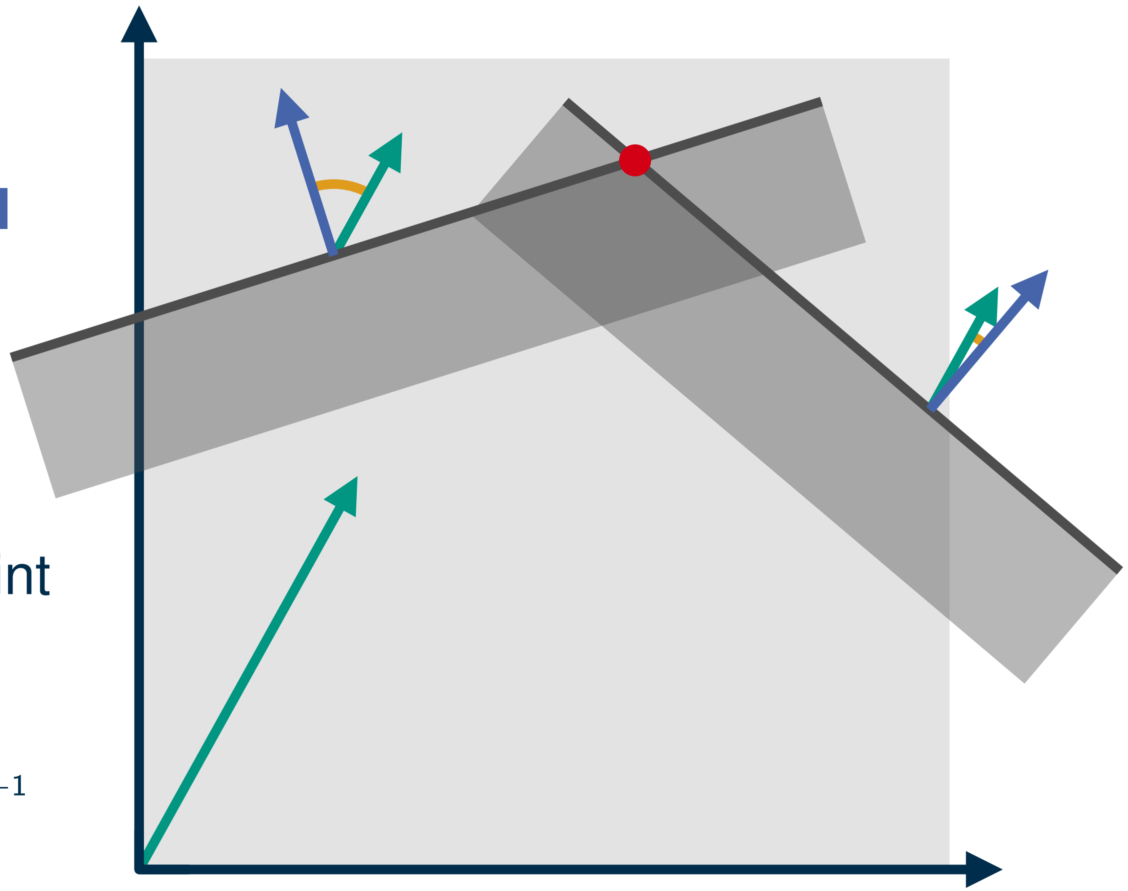
Given $2d$ LP: is an optimal solution bounded?

- It suffices to find at most two half planes
- Calculate angle between **half plane normal** and **maximization vector**

LP Algorithm from Lecture:

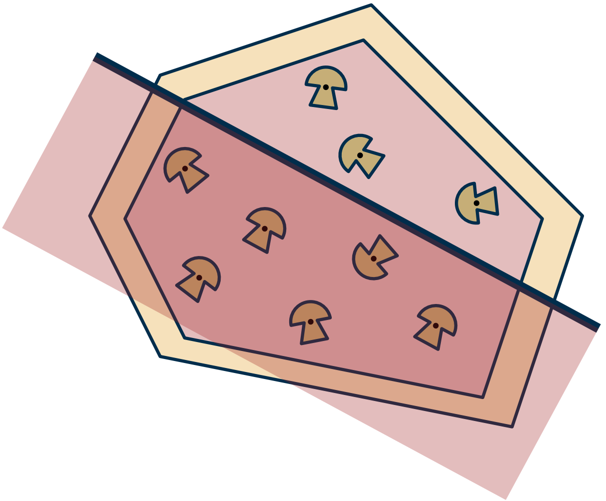
- Iterate over halfplanes (in some order); update best point
- need **initial solution**





Assignment 3

Pizza-Guillotine

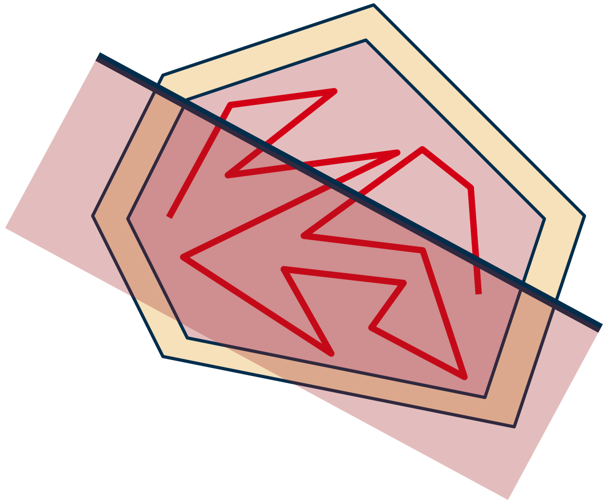


- At which points does the blade cut the pizza polygon?
- Which ingredients are cooked?

- $\mathcal{O}(n \log n)$ precompute, $\mathcal{O}(\log n)$ query

Assignment 3

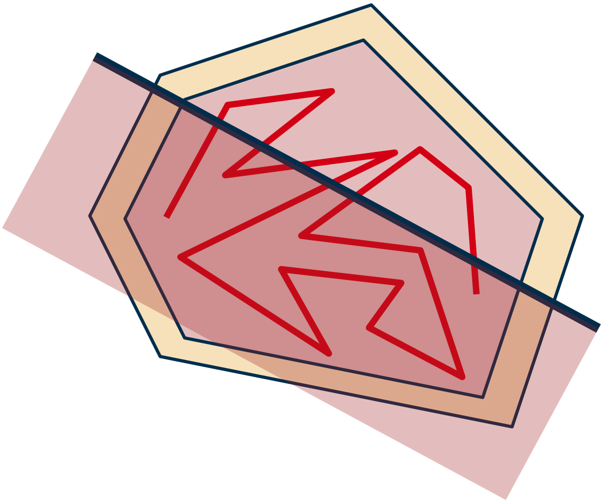
Pizza-Guillotine



- At which points does the blade cut the pizza polygon?
- Which ingredients are cooked?
- how many times is the tomato sauce hit?
- $\mathcal{O}(n \log n)$ precompute, $\mathcal{O}((k+1)\log(n/(k+1)))$ query

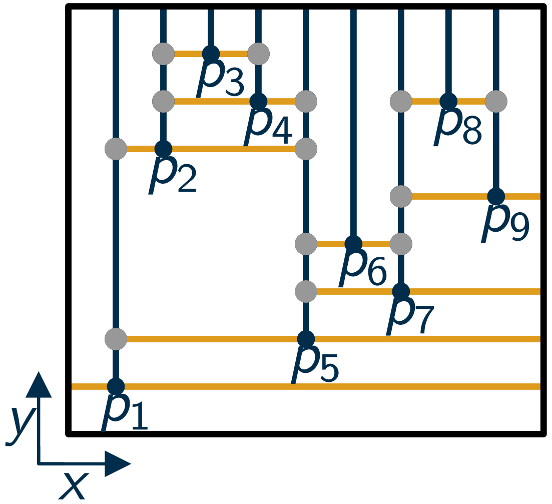
Assignment 3

Pizza-Guillotine



- At which points does the blade cut the pizza polygon?
- Which ingredients are cooked?
- how many times is the tomato sauce hit?
- $\mathcal{O}(n \log n)$ precompute, $\mathcal{O}((k+1)\log(n/(k+1)))$ query

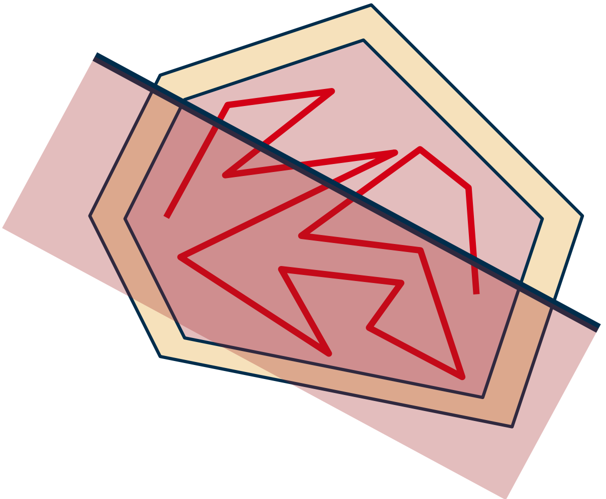
$2d$ Range Queries



- Given the points sorted by x, calculate the geometric graph in $\mathcal{O}(n)$

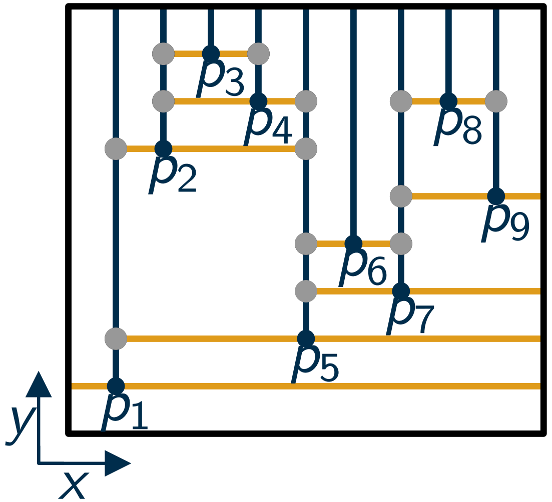
Assignment 3

Pizza-Guillotine



- At which points does the blade cut the pizza polygon?
- Which ingredients are cooked?
- how many times is the tomato sauce hit?
- $\mathcal{O}(n \log n)$ precompute, $\mathcal{O}((k+1)\log(n/(k+1)))$ query

$2d$ Range Queries



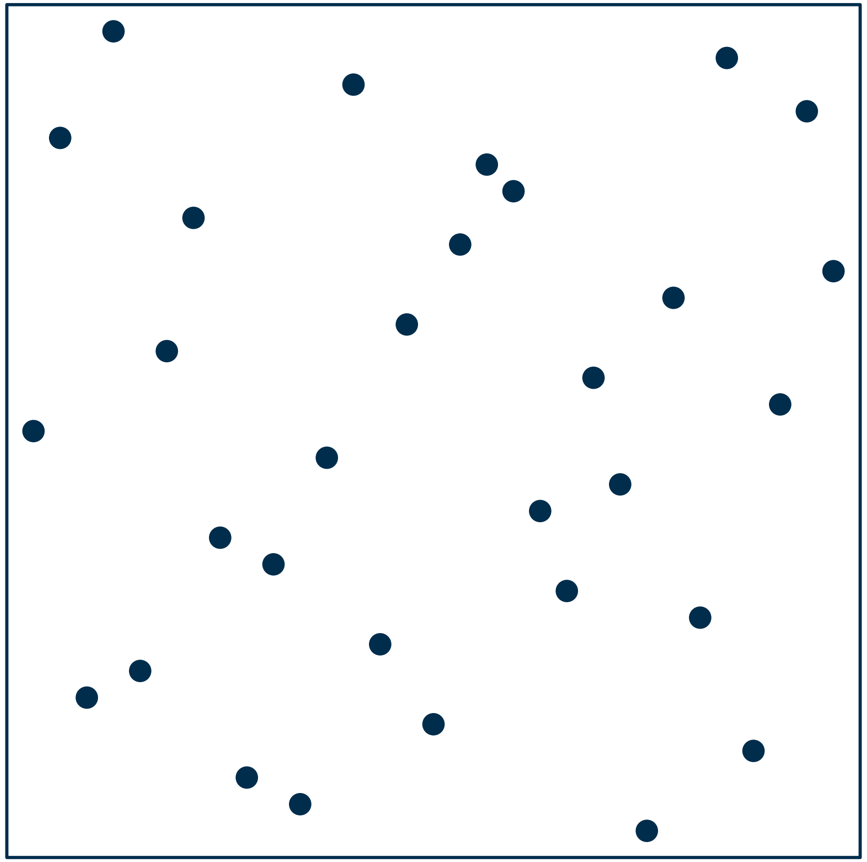
- Given the points sorted by x, calculate the geometric graph in $\mathcal{O}(n)$

Bonus:
Range-Tree implementation. Testcases are on website.

kd-Trees

Binary space partition (in 2d)

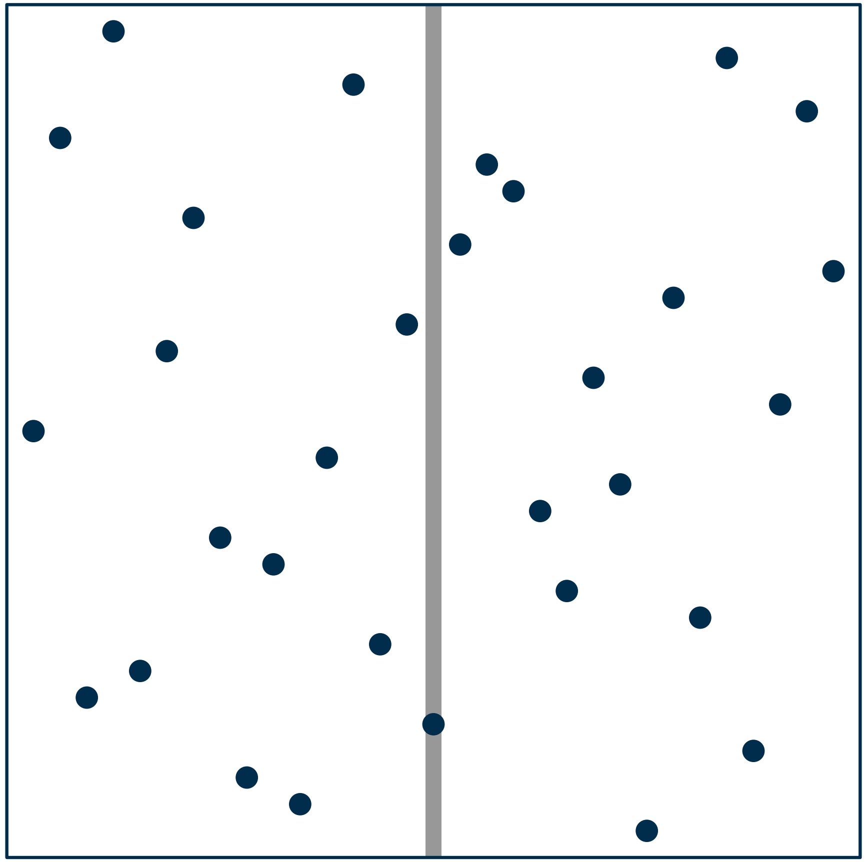
- Given: Set of points



kd-Trees

Binary space partition (in 2d)

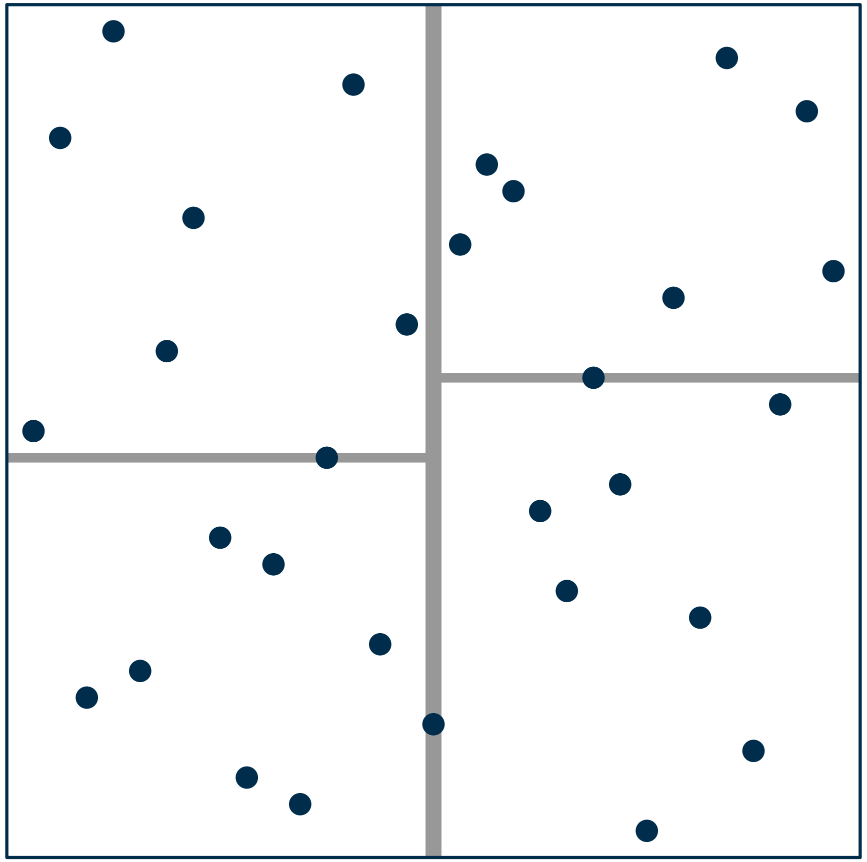
- Given: Set of points
- divide with respect to *x*-coordinate



kd-Trees

Binary space partition (in 2d)

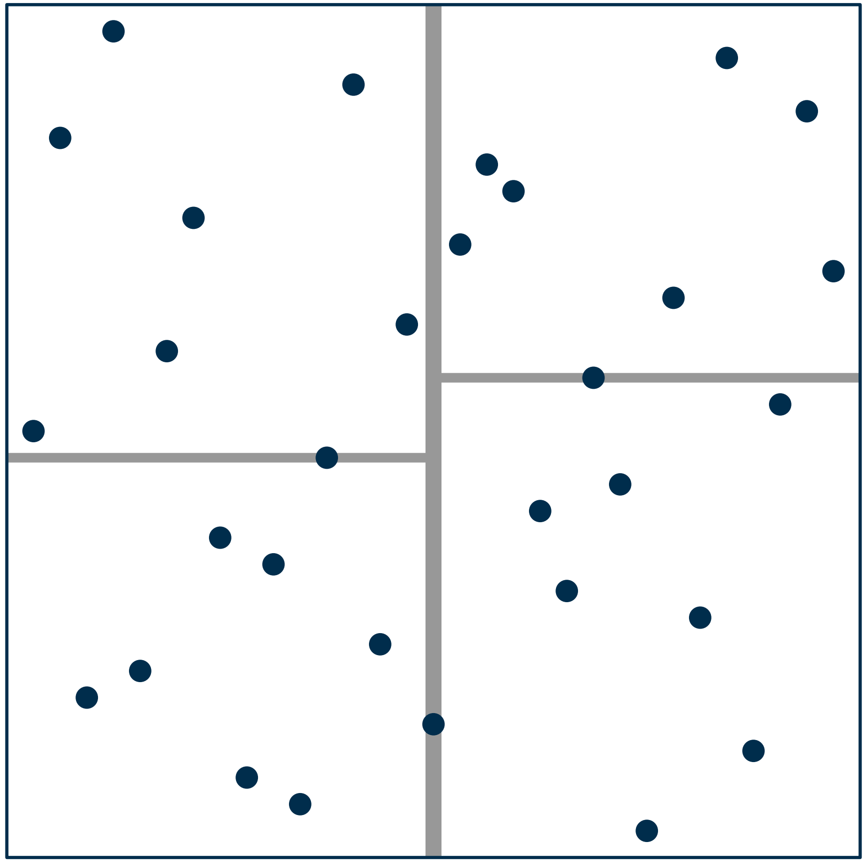
- Given: Set of points
- divide with respect to x-coordinate
- divide each side with respect to y-coordinate



kd-Trees

Binary space partition (in 2d)

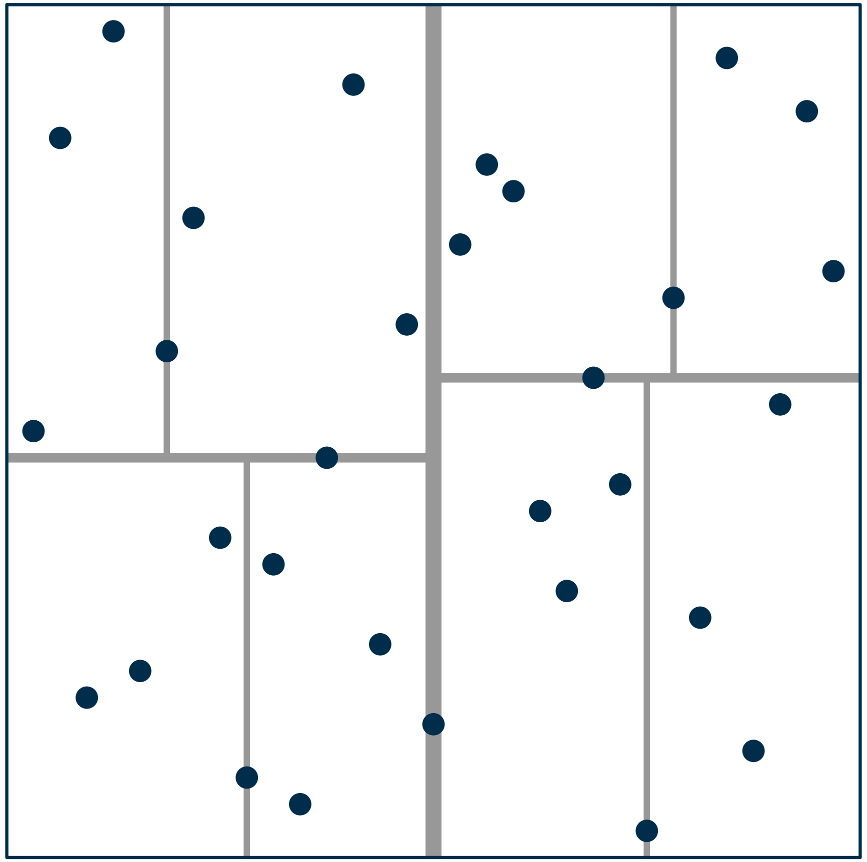
- Given: Set of points
- divide with respect to x-coordinate
- divide each side with respect to y-coordinate
- iterate, until each region contains only $\Theta(1)$ points



kd-Trees

Binary space partition (in 2d)

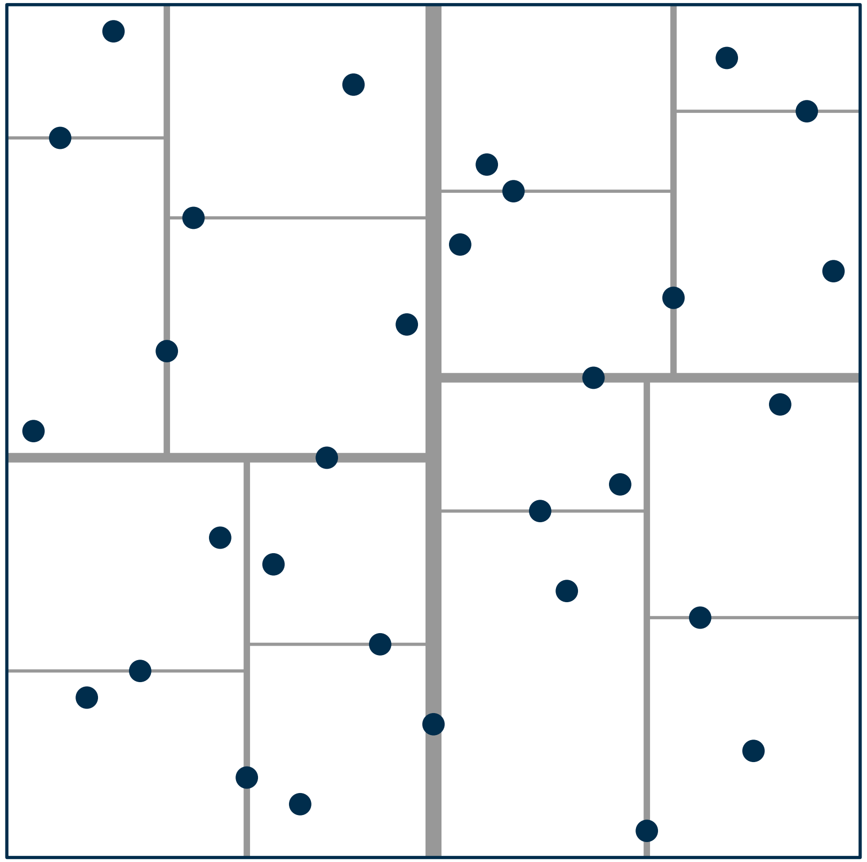
- Given: Set of points
- divide with respect to x-coordinate
- divide each side with respect to y-coordinate
- iterate, until each region contains only $\Theta(1)$ points



kd-Trees

Binary space partition (in 2d)

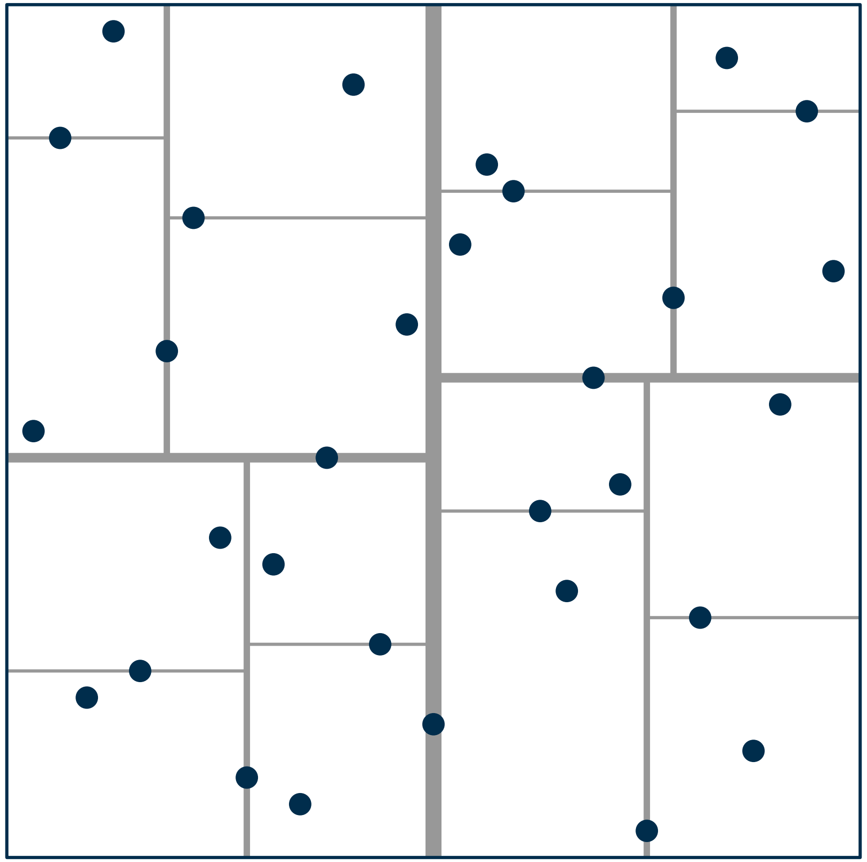
- Given: Set of points
- divide with respect to x-coordinate
- divide each side with respect to y-coordinate
- iterate, until each region contains only $\Theta(1)$ points



kd-Trees

Binary space partition (in 2d)

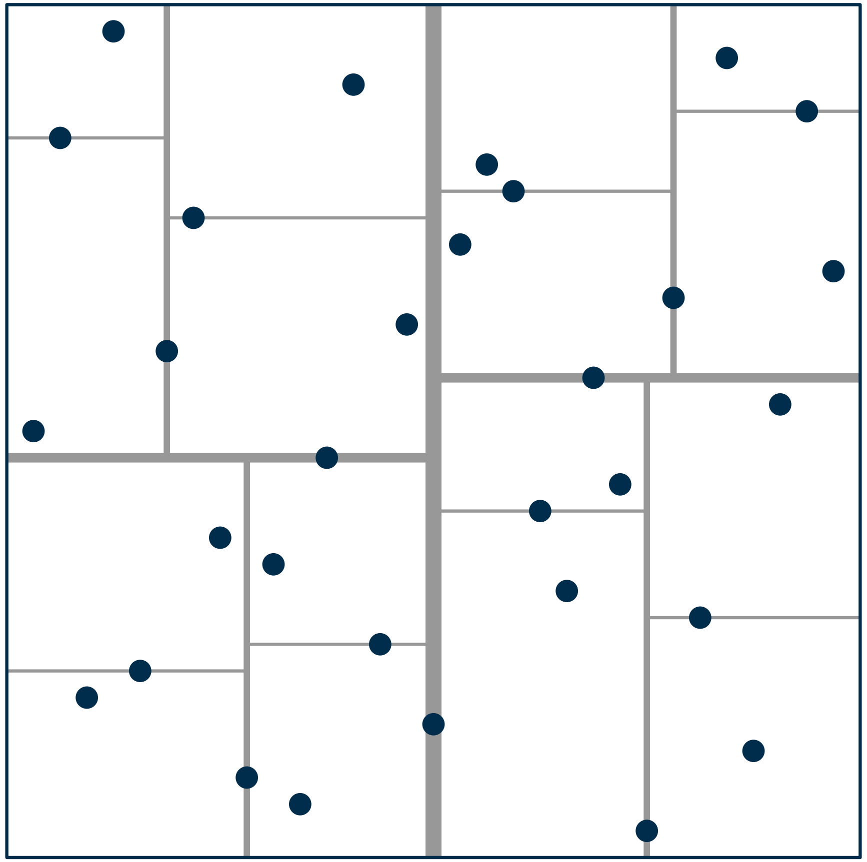
- Given: Set of points
- divide with respect to x-coordinate
- divide each side with respect to y-coordinate
- iterate, until each region contains only $\Theta(1)$ points



kd-Trees

Binary space partition (in 2d)

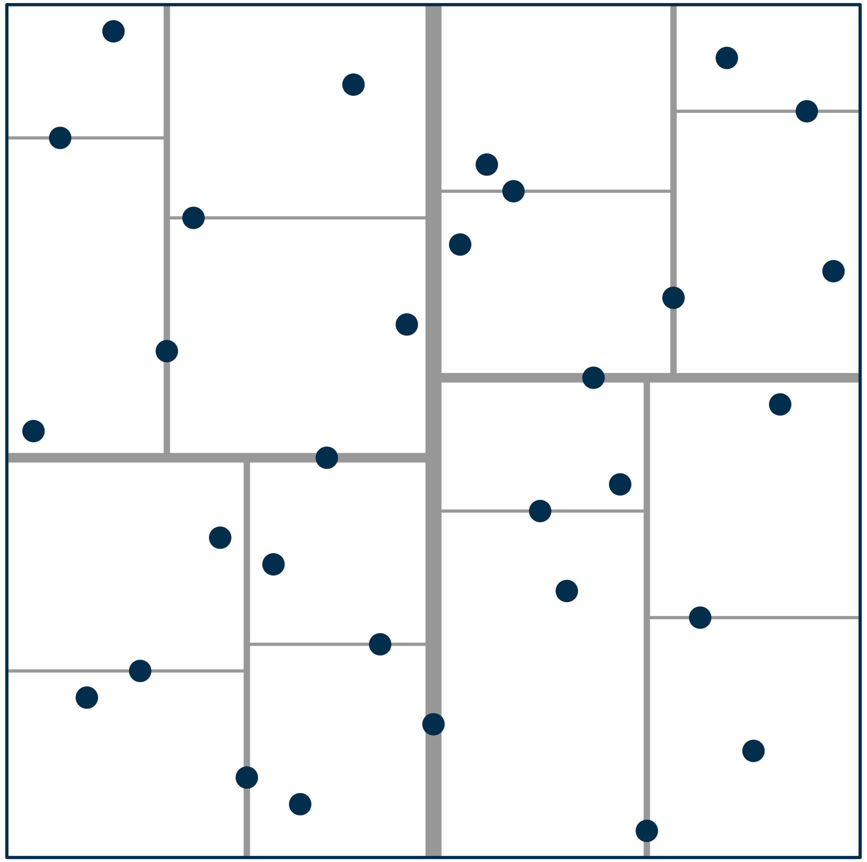
- Given: Set of points
- divide with respect to x-coordinate
- divide each side with respect to y-coordinate
- iterate, until each region contains only $\Theta(1)$ points



kd-Trees

Binary space partition (in 2d)

- Given: Set of points
- divide with respect to x-coordinate
- divide each side with respect to y-coordinate
- iterate, until each region contains only $\Theta(1)$ points



How fast is construction?
How much space do we need?

How can we answer range queries?
(e.g.: find all points in rectangle)

How expensive is an orthogonal range query in the worst case?