we have seen basic examples:

- is a given point inside a given polygon
- is a given point inside a given convex polygon
- is a given point inside a given star-shaped polygon

$\Theta(n)$

$\Omega(\log n)$
we have seen basic examples:

- is a given point inside a given polygon
- convex polygon
- star-shaped polygon

\[ \Theta(n) \]
\[ \Theta(\log n) \]

we can solve those problems faster if the polygon is already known

\[ \leq \text{(pre-processing)} \]
POINT LOCATION

- we have seen basic examples:
  - is a given point inside a given polygon
  - is a given point inside a given convex polygon
  - is a given point inside a given star-shaped polygon

- we can solve those problems faster if the polygon is already known

2 things to minimize:
  - pre-processing time
  - space (data structure)
Pre-processing  
1) store all vertices by x-coord.

Query •
Pre-processing:
1) store all vertices by x-coord.
2) For every vertical slab:
   store a sorted list of segments that cross the slab (ordered & not intersecting)

Time: ?
Space: ?

Query
Pre-processing
1) store all vertices by x-coord.
2) For every vertical slab:
   store a sorted list of segments that cross the slab
   (ordered & not intersecting)

Time: sort + space
Space: $O(n^2)$

Query •
Pre-processing
1) store all vertices by x-coord.
2) For every vertical slab:
   store a sorted list of segments that cross the slab
   (ordered & not intersecting)

Time: sort + space
Space: $O(n^2)$

Query • in $O(\log n)$
1) Binary search on x
2) Binary search in 1 slab
Notice that from one slab to the next, the vertically sorted polygonal segments are almost identical. 
(±2)

So it's a huge waste of space to store a list of size \(O(n)\) in each slab.

.....
I also mentioned two advanced things briefly:
1) A "persistent data structure" can handle sorting all edges in each slab, much faster.
2) There is a logarithmic lower bound for point location

Each of these topics are project material.
our concern is to keep $O(\log n)$ query time but reduce pre-processing time/space to ... $O(n)$

In fact we will do this for arbitrary planar straight-line graphs

more general more interesting
First pre-processing step: triangulate (each region)

$O(n)$ time & space

we will actually determine which triangle contains any given query point

even more general
Even more general: create a triangular outer face

All of this is still $O(n)$ time & space
some useful properties of triangulations
What is the average degree of a triangulation?

\[ \frac{1}{n} \cdot \sum_{i=1}^{n} d(v_i) \]
What is the average degree of a triangulation?

\[
\frac{1}{n} \cdot \sum_{i=1}^{n} d(v_i) = \frac{1}{n} \cdot 2e = \frac{6n - 12}{n} \leq 6
\]
\[ e = 3n - 6 \]

What is the average degree of a triangulation?

\[ \frac{1}{n} \cdot \sum_{i=1}^{n} d(v_i) = \frac{1}{n} \cdot 2e = \frac{6n - 12}{n} \leq 6 \]

\[ \Rightarrow \] Every triangulation has a vertex w/ degree \( \leq 5 \)

(but might only have 12)

Can we find many low-degree vertices? \( \rightarrow \) not if "low" = 5.
what if "low" = 8?
\[ e = 3n - 6 \]

What is the average degree of a triangulation?

\[
\frac{1}{n} \sum_{i=1}^{n} d(v_i) = \frac{1}{n} \cdot 2e = \frac{6n - 12}{n} \leq 6
\]

\[ \Rightarrow \text{Every triangulation has a vertex w/ degree } \leq 5 \]

(but might only have 12)

Can we find many low-degree vertices? \[ \rightarrow \text{not if "low" = 5. what if "low" = 8?} \]

Say you had \( \geq \frac{n}{2} \) vertices w/ degree \( \gg 9 \)

\[
\sum_{d \gg 9} d(v_i) \gg 9 \cdot \frac{n}{2}
\]

sum degrees of \( \frac{n}{2} \) of them
What is the average degree of a triangulation?

\[ \frac{1}{n} \cdot \sum_{i=1}^{n} d(v_i) = \frac{1}{n} \cdot 2e = \frac{6n-12}{n} \leq 6 \]

\[ \Rightarrow \text{Every triangulation has a vertex w/ degree} \leq 5 \]

(but might only have 12)

Can we find many low-degree vertices? \[ \rightarrow \text{not if "low"} = 5. \]

what if "low" = 8?

Say you had \( \gg \frac{n}{2} \) vertices w/ degree \( \gg 9 \)

\[ \sum_{d \gg 9} d(v_i) \gg 9 \cdot \frac{n}{2} \]

sum degrees of \( \frac{n}{2} \) of them \[ \Rightarrow \sum_{d \gg 9} d(v_i) \gg 3 \cdot \frac{n}{2} \]

\( \Rightarrow \) all other vertices have degree \( \gg 3 \)

\[ \Rightarrow \sum_{d \gg 9} d(v_i) \gg 3 \cdot \frac{n}{2} \]

\( \sum + \sum \gg 6n \)

contradiction

always have \( \gg \frac{n}{2} \) w/ deg. \( \leq 8 \)
Always has $\geq \frac{n}{2}$ vertices w/ degree $\leq 8$

In fact many of them must be nicely separated independent set $\Leftarrow$
no 2 are neighbors

Why?
Always has \( \geq \frac{n}{2} \) vertices w/ degree \( \leq 8 \)

In fact many of them must be nicely separated independent set \( \Leftarrow \) no 2 are neighbors

Why? → start w/ \( \frac{n}{2} \) of them; pick one, and mark it.
Mark all of its neighbors too (don’t care about their deg.) \( \Rightarrow \) now we’ve marked \( \leq 9 \) vertices
Always has \( \geq \frac{n}{2} \) vertices w/ degree \( \leq 8 \)

In fact many of them must be nicely separated

\[ \text{independent set} \quad \leftarrow \]

\[ \text{no 2 are neighbors} \]

Why? \( \rightarrow \)

start w/ \( \frac{n}{2} \) of them; pick one, and mark it.

Mark all of its neighbors too (don’t care about their deg.) \( \Rightarrow \) now we’ve marked \( \leq 9 \) vertices

Repeat: pick any unmarked vertex w/ deg. \( \leq 8 \), & mark \( \bigcirc \)

We can do this \( \frac{n/2}{9} \) times

\( \Rightarrow \) get \( \frac{n}{18} \) independent vertices w/ degree \( \leq 8 \)
we find $\circ$ w/ deg. $\leq 8$

degree $< 8$ \{ we don't care ...

degree $\geq 8$
we find \( \bigcirc \) w/ deg. \( \leq 8 \)

and then \( \bigcirc \) also w/ deg. \( \leq 8 \) and not a neighbor
Time to find independent set of degree \( \leq 8 \) ?
Time to find independent set of degree ≤ 8?

- First mark all degree > 9: $O(n)$ ($\exists d(v) = O(n)$)
- Place all unmarked vertices in a list
- Each time scan from start of list for first unmarked vertex (& delete marked)
- When marking neighbors can also delete from list

Requires simple graph structure (access to neighbors) & links to the list.

$O(n)$ overall
Point Location Data Structure
SURROUND GRAPH WITH 3 VERTICES
Find low-degree vertex.
Removed independent set
RE-TRIANGULATION IS ARBITRARY
STORE THIS
STORE THIS
ANOTHER EXAMPLE
LOW DEGREE VERTEX
& NEIGHBORS
NEW L-D & NEIGHBORS
End of Round 1:

Independent Set
RETRIANGULATE
ARBITRARILY
LOW DEGREE VERTEX
& NEIGHBORS
END OF ROUND 2:
INDEPENDENT SET
When we delete a point, we create a hole: remove $O(1)$ triangles.
When we delete a point, we create a hole: remove $O(1)$ triangles

Retriangulate: create $O(1)$ triangles in the same region
When we delete a point, we create a hole: remove $O(1)$ triangles.

Retriangulate: create $O(1)$ triangles in the same region.

Between successive stored triangulations, keep links between overlapping triangles: $O(1)$ links per triangle.
Every iteration: 1) get rid of constant fraction of points
\[ \geq \frac{1}{18} \quad \text{low-degree} \]
Every iteration:

1) get rid of constant fraction of points
   $\geq \frac{1}{18}$  low-degree

2) re-triangulate: $O(1)$ per hole  low-degree
Every iteration:

1) get rid of constant fraction of points

\[ > \frac{1}{18} \quad \text{low-degree} \]

2) re-triangulate: \( O(1) \) per hole

For \( k \) remaining points,

Time:

[1] \( O(k) \) \quad \text{scan all points}

[2] \( O(k) \) \quad \# \text{holes}
Every iteration:
1) get rid of constant fraction of points
\[ \geq \frac{1}{18} \text{ low-degree} \]
2) re-triangulate: \( O(1) \) per hole

For \( k \) remaining points,

Time:
[1] \( O(k) \) scan all points
[2] \( O(k) \) \# holes

\[ O(n) \text{ overall} \]
\[ O(n) \text{ size of structure} \]
How to use this structure to locate a query point Q
How to use this structure to locate a query point \( Q \)

1) Look at top-level structure: \( \triangle \) outer triangle: triangulation \( T_1 \)
How to use this structure to locate a query point $Q$

1) Look at top-level structure: $\triangle$ outer triangle: triangulation $T_1$

   $Q$ outside? Done. Inside? Dig deeper: look at $T_2$
How to use this structure to locate a query point \( Q \)

1) Look at top-level structure: \( \triangle \) outer triangle: triangulation \( T_1 \)
   \( Q \) outside? Done. Inside? Dig deeper: look at \( T_2 \)

2) \( \triangle \) \( T_2 \)
   Repeat question: which of these triangles contains \( Q \)?
How to use this structure to locate a query point $Q$

1) Look at top-level structure: $\triangle$ outer triangle: triangulation $T_1$
   $Q$ outside? Done. Inside? Dig deeper: look at $T_2$

2) $T_2$
   Repeat question: which of these triangles contains $Q$?

3) Dig deeper:
   find what triangles of $T_3$ overlap $\triangle$ in $T_2$
How to use this structure to locate a query point $Q$

1) Look at top-level structure: $\triangle$ outer triangle: triangulation $T_1$
   $Q$ outside? DONE. Inside? Dig deeper: look at $T_2$

2) $\triangle T_2$
   Repeat question: which of these triangles contains $Q$?

3) Dig deeper:
   find what triangles of $T_3$ overlap $\triangle$ in $T_2$
How to use this structure to locate a query point $Q$

1) Look at top-level structure: $\triangle$ outer triangle: triangulation $T_1$
   $Q$ outside? done. Inside? Dig deeper: look at $T_2$

2) \[ T_2 \] Repeat question: which of these triangles contains $Q$?

3) Dig deeper:
   find what triangles of $T_3$ overlap $\triangle$ in $T_2$

\[ T_2 \]

Repeat: which of these new triangles in $T_3$ contains $Q$?
   - get 1 triangle & dig deeper
# triangles → $T_1$

$T_2$

$T_3$

Per internal node:

$1 \leq \# \text{outgoing edges} \leq 8$
Per internal node:
\[1 \leq \text{#outgoing edges} \leq 8\]

\[O(\log n)\] because we remove constant fraction between levels
Per internal node:
1 ≤ #outgoing edges ≤ 8

Search involves traversal: root → leaf:

\[O(\log n)\] because we remove constant fraction between levels

at every node do triangle test on all children