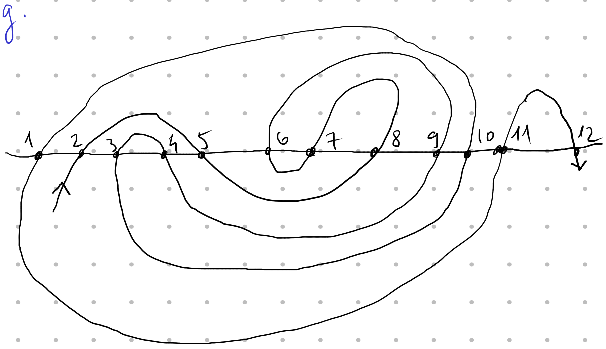
Identifying and sorting Jordan sequences

e.g.



- horizontal line
- Jordan curve
  - no self-intersections
- number line/curve intersections left-to-right
- read out numbers along the curve:
- assume curve starts and ends below line

2, 5, 8, 7, 6, 9, 4, 3, 10, 1, 11, 12

Jordan seq: result of this process.

Problem: Given sequence, verify if Jordan-seq, and sort.

Want to do it in $O(n)$ time.

[Sorting needs $\Omega(n \log n)$ comparisons

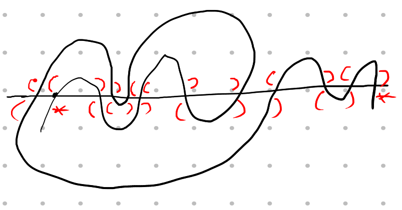
$n!$ permutations

$\log_2(n!) = \Omega(n \log n)$]

Claim. # permutations of size n that are Jordan-seq. is $< 5^n$.

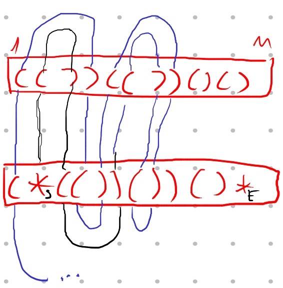
(No $n \log n$ lower bound does not apply, same argument would yield at best $\Omega(n)$.)

Proof of Claim:



Above

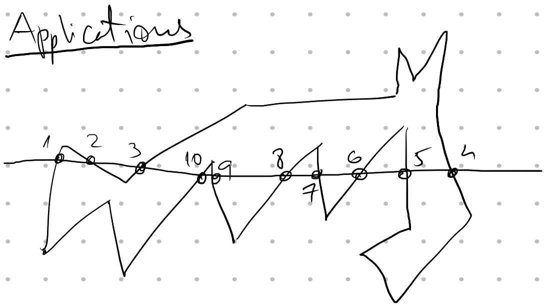
Below



- encoding of curve/line intersections
- How many possible valid encodings?

$\leq \underbrace{2^n}_{\text{2 characters, length } n} \cdot \underbrace{2^n}_{\text{2 chars, length } n} \cdot \underbrace{n^2}_{\text{two } *\text{s}} = 4^n \cdot n^2 \in O(5^n)$

Applications

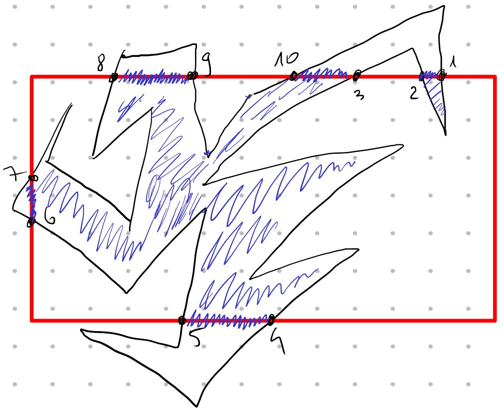


Find intersections between polygon and line

(typically easier in order along polygon)

Want to report along line.

→ Jordan sorting



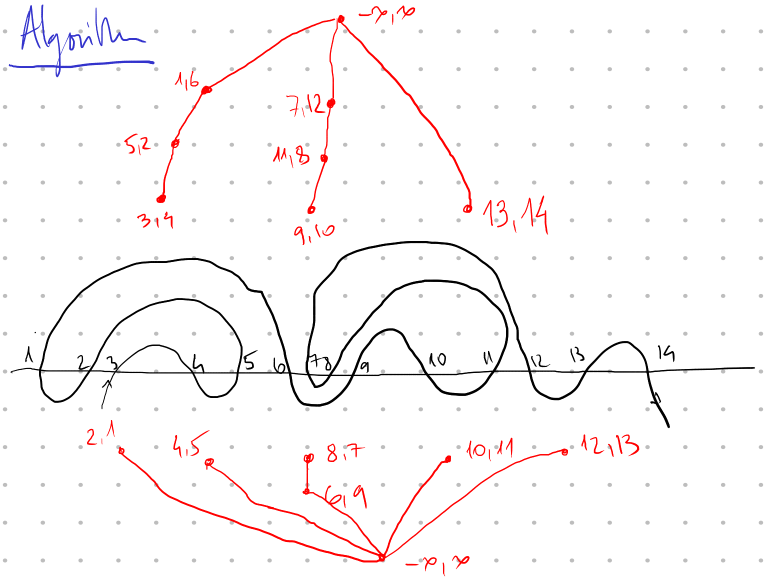
Application (Computer Graphics)

- find visible part of polygon inside window
  - find intersections along polygon
  - exit then along window

  → Jordan sorting

Algorithm

Idea: Represent Jordan seq. with two trees, one above one below



$X_1, X_2, \dots$ X_n

3, 4, 5, 2, 1, 6, 9, 10, 11, 8, 7, 12, 13, 14

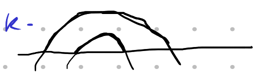
Obs. pair $\{X_{2i-1}, X_{2i}\}$ is a "bump"

Obs. bumps cannot intersect

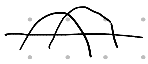
2 valid cases:

O.k.

O.k.



invalid case:



NO

Upper tree

Build tree with nodes $\{X_{2i-1}, X_{2i}\}$

for $i = 1, 2, \dots$

parent of $\{X_{2i-1}, X_{2i}\}$ is closest pair that contains it

(or $\{-\infty, \infty\}$)

Lower tree

nodes $\{X_{2i}, X_{2i+1}\}$

for $i = 1, 2, \dots$

Algorithm

Input: sequence $X_1, \dots, X_n$ (list of intersections along curve)

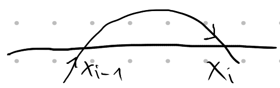
Process $X_1, \dots, X_n$

Build:
- upper tree
- lower tree
- sequence sorted along line

if get stuck, report "NOT Jordan seq."

Initialize:

- make $\{-\infty, \infty\}$ root of both trees
- Sorted list $(-\infty, X_1, \infty)$
- for $i = 2, \dots, n$ process X_i



if i even, ~~then~~ add $\{X_{i-1}, X_i\}$ to upper tree

(if i odd, then add $\{X_{i-1}, X_i\}$ to lower tree) – symmetric, details skipped

(assume $X_{i-1} < X_i$, other case symmetric)

$O(1)$ time

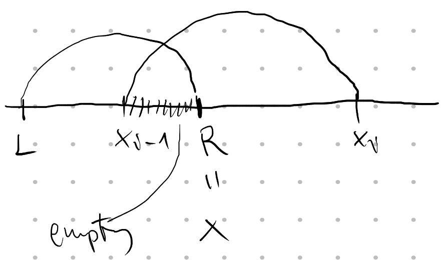
① find X in sorted list that is right neighbor of X_{i-1}

② find pair in upper tree containing X. → (assume elements in sorted list are linked to pair in trees that contain them)

$X \in \{L, R\}$

5 cases

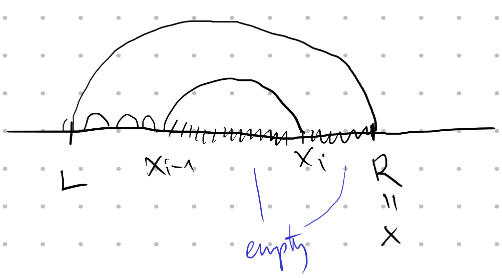
Ⓐ



$L < X_{i-1} < R < X_i$

NOT A Jordan-seq. STOP

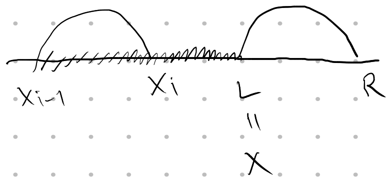
Ⓑ



$L < X_{i-1} < X_i < R$

- make $\{X_{i-1}, X_i\}$ rightmost child of $\{L, R\}$
- insert X_i after X_{i-1} in sorted list

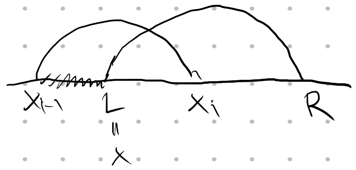
(C)



$x_{i-1} < x_i < L < R$

- make $\{x_{i-1}, x_i\}$ new left sibling of $\{L, R\}$
- insert x_i after x_{i-1} in sorted list

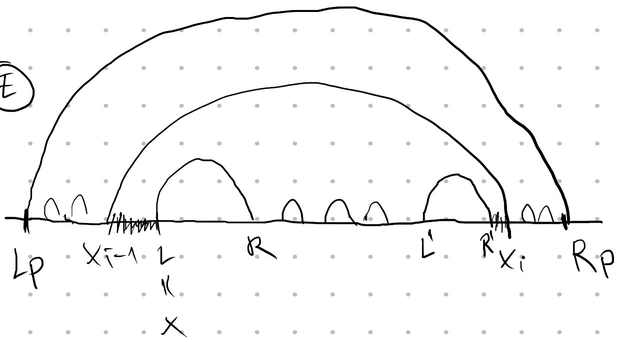
(D)

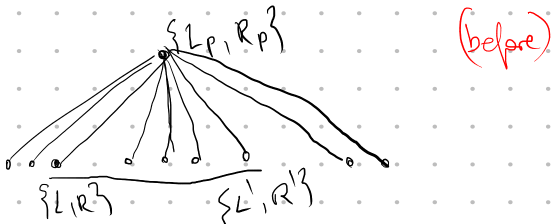


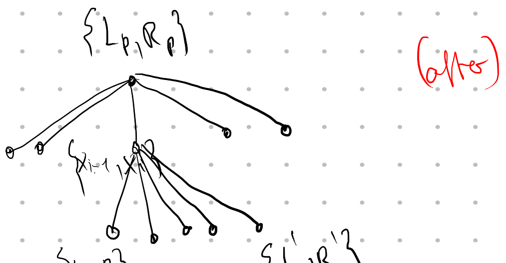
$x_{i-1} < L < x_i < R$

NOT a Jordan-seq. STOP

(E)







$x_{i-1} < L < R < x_i$

- Find rightmost sibling of $\{L, R\}$ s.t. $L' < x_i$
  (if $R' > x_i$ then NOT Jordan seq. STOP)
- Find parent of $\{L, R\}$, call it $\{L_p, R_p\}$
  (if $R_p < x_i$, then NOT Jordan seq. STOP)
- Remove sublist of children $\{L, R\}, \dots, \{L', R'\}$ of $\{L_p, R_p\}$, replace by new node $\{x_{i-1}, x_i\}$, make removed sublist child of $\{x_{i-1}, x_i\}$
- Insert x_i after R' in sorted list

<u>Data Structures</u>

- Sorted list as doubly linked list (insert after given element in $O(1)$ time)
- Lists of siblings (in upper and lower trees) as <u>finger search trees.</u> $\longrightarrow$ list of siblings supports insert, finger search, split w. 2 pivots

Running time

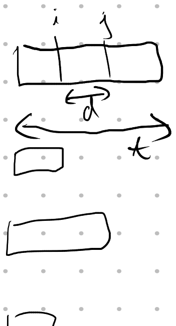
- insert rightmost child $O(1)$
- insert left neighbor $O(1)$
- 2-pivot split, remove sublist of length d from a list of length t
  (after searching for endpoint)

$$O\left(\log \min\{d, t-d\}\right)$$

(recall finger split of FST)

Overall running time

Total cost of splittings (reuse analysis of Application ② splittable lists)



- split(i,j) cost $\log \min\{d, t-d\}$
- insert cost $O(1)$

define potential $t - \log_2 t$ for list of length t

total potential $\phi = \sum_{\text{lists}}$

Amortized cost of split $O(1)$

Amortized cost of insert $O(1) + \Delta\phi = O(1)$

Processing x_i amortized $O(1)$

Total time for Jordan-Sorting $O(n)$.