Lower bound on $\text{opt}$. 

Search sequence $R = r_1, \ldots, r_n$.

$\text{OPT}(R)$ is smallest possible # rotations and pointers needed for every $R$.

"offline" optimum

Recall that we would like to prove:

Cost of Splay or Greedy is $O(\text{OPT}(R))$ for all $R$.

In geometric view

\[ \text{OPT}(P) : \text{smallest } S \ni P \text{ s.t.} \]

\[ S \text{ has no empty rectangles} \]

For some Algorithm denote $\text{Alg}(P)$: cost of algorithm on $P$.

\[ \frac{\text{Alg}(P)}{\text{OPT}(P)} \leq (\text{something}) \]

Comparing directly to $\text{OPT}(P)$ is difficult, i.e. $\text{OPT}$ is not well-understood.

("recall in last update MTF vs OPT"

= # inverted pairs)

Idea: compare algorithm against a lower bound on $\text{OPT}$.

A quantity $\text{LB}(P)$ s.t.

\[ \text{LB}(P) \leq \text{OPT}(P) \leq \text{Alg}(P) \]

"lower bound" abstract quantity easy to compute

\[ \frac{\text{Alg}(P)}{\text{OPT}(P)} \leq (\text{something}) \]

Need a good $\text{LB}$ for $\text{OPT}$. 
Interleave lower bound (and Poen 1980)

$L(L(P))$
$L(L(R))$

$R = \{r_1, \ldots, r_m\}$

- Split with vertical line through the middle point by x-coordinate
- Visit all points in order bottom to top
- Enumerate L/R crossing
- Do the same recursively on L/R side

$L(L(P)) = |P| + \#\text{crossings at all levels}$

eg: $L(L(P)) = 11 + 5 + 1 + 2 = 19$

Divide and conquer algorithm captured cut of search tree e.g. in a balanced tree

$DC(P)$

obs: $L(L(P)) \leq DC(P)$

Thus: $L(L(P)) \leq OPT(P) \leq DC(P)$

eg: $L(L(P)) = n$
Geometric view of BST (recap)

Given $P$, find $S \supseteq P$, a superset of $P$, as well as possible, such that:

1. $x, y \in S$
2. Rectangle with corner $x, y$
3. Contains some point in $S \setminus \{x, y\}$
4. (unless $x, y$ in same row/column)

$OPT(P) = \text{size of smallest such set } S$

Captures BST optimum of server search sequence $P$
Interleave lower bound

$\text{IL}(P)$

$\text{IL}(R)$

$R = (r_1, ..., r_m)$

\[ \text{IL}(P) : \]
- split with vertical line through the middle point by x-coordinate
- visit all points in order bottom-to-top
- enumerate L/R crossing
- do the same recursively on L/R side

\[ \text{IL}(P) = |P| + \# crossings at all levels \]

e.g. $\text{IL}(P) = 11 + 5 + 1 + 2 = 19$

Divide and conquer algorithm: captured cut of recutity search set in a fixed balanced tree.

$\text{DC}(P)$

$\text{IL}(P) \leq \text{DC}(P)$

Thus: $\text{IL}(P) \leq \text{OPT}(P) \leq \text{DC}(P)$

e.g. $\text{IL}(P) = m$
Theorem

\[ IL(P) \leq \text{OPT}(P) \]

Proof

(2) \[ IL(P) \leq MN(P) \leq \text{OPT}(P) \]

Optimum Manhattan Network of \( P \)

Manhattan path from \( x \) to \( y \)

Sequence

\( x_0, \ldots, x_k \)

\( x_i \neq x_j \) for all \( i \neq j \)

\( (\text{no step in "wrong direction"}) \)

(1) \( x_{i+1}, x_k \) are in the same row or column

\( \forall i \in [0, k-1] \)

(2) \( d(x_{i+1}, x_k) \leq d(x_i, x_k) \)

\( \forall i \in [0, k-1] \)

only allow steps \( \rightarrow \) or \( \downarrow \)

(and symmetrical 2 cases)
**Obs**

$S$ has no empty rectangles

$\forall x, y \in S$, there is a Manhattan path $x \rightarrow y$ in $S$

**Proof**

1. **Rects not empty**

2. **Must contain some point $z$**

   $x \rightarrow z$ Manhattan path

   $z \rightarrow y$

   Merger two paths by

   $x \rightarrow y$ Manhattan path

   (Note: this is in general not true, but here.)
Given \( P \), find smallest S2P s.t.
\[ \forall x, y \in S \ \text{there is a Manhattan path } x \rightarrow y \ \text{in } S \]
\[ \text{cost: } \text{OPT}(P) \]

Alternative problem
Given \( P \), find smallest S2P s.t.
\[ \forall x, y \in P \ \text{there is a Manhattan path } x \rightarrow y \ \text{in } S \]
\[ \Rightarrow \]
Input points
This is an easier problem
\[ \text{cost: } \text{MN}(P) \]

\[ \xrightarrow{\text{Manhattan Network,}} \]
\[ \text{dec. tree is path } \ a \rightarrow b \]
\[ b \rightarrow c \]
\[ a \rightarrow c \]
\[ \Rightarrow \text{but not a valid saturated expert (BST)} \]
\[ b, c \text{ form empty rectangle} \]

Consequence:
\[ (\text{MN}(P) \leq \text{OPT}(P)) \]

Remark
\[ \Rightarrow \text{it is conjectured that } \text{MN}(P) = \Theta(\text{OPT}(P)) \]

\[ \Rightarrow \text{optimum Manhattan network can be efficiently 2-approximated} \]
\[ (\text{MN}(P)) \]
Remains to show
\\(2) \\
IL(P) \leq MN(P)

Optimal Manhattan Net:
Size \(MN(P)\)

\[#\text{segments in MN} \leq MN(P) - |P|\]

\(IL\) lower bound

\(\overline{\text{Obs: each horizontal segment of MN is charged at most once.}}\)

\[\Rightarrow \#\text{crossings} \leq \#\text{segments} \text{ in } IL\]

\[IL(P) = \#\text{crossings} + |P| \leq \#\text{segments} \text{ in } MN + |P| = MN(P) + |P| - |P|\]
Next: Tryo tree

Cost of Tryo tree $T \leq O(\log \log n) \cdot IL(R)$

for $R \leq O(\log \log n) \cdot OPT(R)$

Start with IL lower bound

Point set $P$ obtained from search sequence $R$

$R = r_1, \ldots, r_m$

View recursive separately like as nodes of a balanced BST

Each node has a "preferred child"

on which side we searched previously (most recently)

Initialize arrows (preferred children) arbitrarily

Update arrows after each search.
Obs. Changes in preferred child = crossing in IL

lower bound tree

- create arrows into paths
- "preferred path decomposition" of tree

Obs. each path is of length at most log n.

Idea: maintain preferred-path decomposition of lower bound tree as a data structure.

- each preferred path is maintained as an auxiliary balanced BST (e.g. AVL, red-black tree, etc.)

→ Tapyo tree DS.

Search in Tapyo tree?
- as in a normal BST

Obs. cost of search:
- as long as we stay inside an auxiliary tree (black)
- cost is only $O(\log \log n)$

Problem: search may pass from one auxiliary tree to another,
- take a non-preferred child in LB tree → crossing in IL
In summary: whenever we cross from one aux-tree to another, we charge it to a cross in IL

⇒ for every cross in IL we spend \( O(\log n) \) cost

\[ \text{cost} \leq O(\log n) \cdot |L| \]

A detail still missing:

preferred child decomposition of LS tree must be updated after each search.

main search is a right subtree of root

uplink arrow: we cut out part of B and join it with C

for every change of arrow 1 split + 1 join

- split subtree contains keys in some interval \([a,b]\)

- join in between two trees, one joins a continuous subinterval \([a,b]\), the other has no key in \([a,b]\)

Lemma: We can implement split and join of trees of size \( \leq k \) in time \( O(\log k) \).
Split splay tree at $x$:
- Splay($x$)
- Join
- Total splay of $T_2$ is not attached to $T_1$ as left child

Split and join of auxiliary trees take $O(\log \log n)$ time.

We also charge this to $\mathcal{L}$ lower bound.

\[ \Rightarrow \text{total cost} = O(\log \log n) \cdot \mathcal{L}(R) \leq O(\log \log n) \cdot \text{OPT}(R) \]

Note: What bookkeeping is needed to implement split/join of aux-trees?

We notice we ended a different aux-tree, so we need to cut/join.
We augment aux-tree to store in each node its original depth in pref. path.

So we know $b$ and depth $d$.

We need to cut out all nodes with depth $> d$.

These nodes form an interval in aux-tree $[a,b]$.

How do we find $a$?

Need an extra augmentation: each node stores max-depth of nodes in its subtree.

(this needs to be maintained under rotations in red-black/splay, see lecture on aux. tree)

Let $a$ be the root. Go down from root repeatedly, as long as max-depth $> d$

If possible, go to left child, otherwise to the right.

Summary
- we need to cut out nodes with depth $> d$
- aux-tree is not sorted by depth, but by key
- nodes with depth $> d$ form interval $[a,b]$ by key
- we can find $a$/$b$ using augmentation