TODAY: Dynamic Optimality I (of 2)
- binary search trees
- analytic bounds
- splay trees
- geometric view
- greedy algorithm
Q: is there one best binary search tree (BST)?

**BST:** comparison data structure
  supporting search
  (& predecessor/successor, insert/delete)

Also a model of computation (for DSs)
  - data must be stored in a BST
  - unit-cost operations:
    - walk left, right, or up (parent)
    - rotate this node & its parent

\[
\begin{align*}
&\text{rotate } x \\
&\text{rotate } y
\end{align*}
\]

(- create/destroy leaf)

\[\Rightarrow \text{search cost} = \text{length of root-to-node path}\]

**DSs in this model:**
  - vanilla BST (no rotations)
  - AVL trees
  - red-black trees (B-trees)
  - BB[\alpha] trees
  - splay trees
  - Tango trees
  - Greedy

\[\mathcal{O}(\log n) \quad /\text{op.}\]

\[\text{focus here}\]
Is $O(\lg n)$ search optimal?
- depends on sequence of searches
- say we're storing keys $1, 2, \ldots, n$
& search for $x_1, x_2, \ldots, x_m$

Sequential access property:
$1, 2, \ldots, n \Rightarrow O(1)$ amortized/operation
[in-order traversal in any BST]

Dynamic finger property:
$|x_i - x_{i-1}| = k \Rightarrow O(\lg k)$/operation, possible
[think level-linked B-trees ~ but BST]

Entropy bound/static optimality:
k appears $p_k$ fraction of the time
$\Rightarrow O\left(\sum_{k=1}^{n} p_k \lg \frac{1}{p_k}\right)$/operation.
[store $x_i$ at height $\leq \lg \frac{1}{p_k} + 1$]

Working-set property:
if $t_i$ distinct keys accessed since last
access to $x_i$, then $O(\lg t_i)$ possible
[intuition: store most recent higher up]
$\Rightarrow$ if all $x_i \in S$ then $O(\lg |S|)$/operation, possible
[form BST on $S$, put rest below]

\* = hard to do with BST, but possible!
Unified property: \[ \text{if } t_{ij} \text{ distinct keys accessed in } x_i \ldots x_j \text{ then } x_j \text{ costs } O \left( \lg \min_i \left[ \frac{1}{t_i} \right] + t_{ij} + \alpha \right) \]

"fast if close to something recent"

- e.g. \( 1, \frac{n}{2}, n, \frac{n}{2} + 1, 3, \frac{n}{2} + 3, \ldots \) \( \Rightarrow O(1) / \text{op} \)
- implies both working set & dynamic finger
- possible on pointer machine \([\text{Iacono, Badiu, Cole, Demaine, Iacono} - \text{Algorithmica 2007}]\)
- possible on BST up to additive \( O(\lg \lg n) \) \([\text{Bose, Douieb, Dujmović, Howat - Algorithmica 2012}]\)

**OPEN**: possible on a BST?

Dynamic optimality / \( O(1) \)-competitive:

\[ \text{total cost} = O(OPT) \]

\( \text{min. cost of any BST on this access sequence} \)

- **OPEN**: possible for any (online) BST?
- **OPEN**: possible for any pointer-machine DS?
- **OPEN**: is any pointer-machine DS \( = O(OPT \text{ offline pointer-machine DS}) \)?

- balanced BST is \( O(\lg n) \)-competitive
- Tango trees are \( O(\lg \lg n) \)-competitive \([L6]\)
Splay trees: [Sleator & Tarjan – JACM 1985]
- binary search for \( x \)
- modify the path:
  - zig-zig:
  - zig-zag:
  - at the end, possible single rotation to put \( x \) at root
  - key feature: at most half the nodes on the path go down in the tree

Performance: (amortized)
- has working-set property [Sleator & Tarjan]
- has dynamic-finger property [Cole – SICOMP 2005]

- CONJECTURE: has unified property [Iacono]
- CONJECTURE: dynamically optimal [Sleator & Tarjan]
Geometric view:

access sequence
\rightarrow point set \{ (x_i, i) \}

BST execution
\rightarrow point set: which nodes touched during search(\(x_i\))?  

Theorem: point set is a valid BST execution \iff Arborally Satisfied Set (ASS)

- rectangle spanned by two points in set, not on horizontal/vertical line, contains another point
- in fact must have another point on a rectangle side incident to either corner:

Corollary: \( \text{OPT} = \text{smallest ASS containing input} \)

OPEN: complexity? \(O(1)\)-approximation?
Proof of Theorem:

\( \Rightarrow \) consider rectangle spanned by \((i, x) \Rightarrow (j, y)\)
- let \(a_t = \text{lca of } x \& y\) just before time \(t\)
- for all \(t\): \(x \leq a_t \leq y\) & \(a_t\) is an ancestor of \(x \& y\)
\(\Rightarrow (a_i, i) \& (a_j, j) \in \text{execution}\)
  (need to touch all ancestors of touched nodes)
- want a third point in the rectangle
- if \(a_i \neq x\) then use \((a_i, i)\)
- if \(a_j \neq y\) then use \((a_j, j)\)
- else: \(a\) changes from \(x\) to \(y\) between times \(i \& j\)
\(\Rightarrow y\) rotated before time \(j\)
\(\Rightarrow (y, t) \in \text{execution for some } i \leq t < j\)
\( \Leftarrow \) define tree at all times to be treap: 
- BST & heap ordered by next-touch-time  
  - note: next-touch-time has some ties, 
    so this is not uniquely defined  
  - when we reach time \( i \), nodes to touch 
    form a connected subtree at the top 
    (by heap-order property)  
  - these nodes get new next-touch-time  
  - re-arrange into local treap  
    (this still may be ambiguous — break ties 
    arbitrarily — but still restricts global choice)  
- claim: global treap

\[
\text{touched} 
\xrightarrow{\text{heapify}} 
\text{heaped}
\]

if \( y \) to be touched sooner than \( x \) 
then \( (x, \text{now}) \rightarrow (y, \text{next-touch}(y)) \) 
is an unsatisfied rectangle: 
(according to 2\textsuperscript{nd} definition of ASS)

\[
\text{next-touch}(x) \rightarrow \circ \quad \text{empty by "if"} 
\rightarrow \circ \\
\]

leftmost such point would be right child 
of \( x \) after search(\( x_i \)), not \( y \)
Simple example:
Greedy algorithm: [Lucas 1988; Munro 2000]
- consider point set one row at a time
- add the necessary points on that row
- in tree view: re-arrange root-to-x path optimally for future searches

**CONJECTURE**: Greedy = \( \mathcal{O}(\text{OPT}) \)  
or even: \( \text{OPT} + \mathcal{O}(m) \)
- seems obvious... "just" need to show you needn't stray from the access path

So what?

**Theorem**: online ASS algorithm  
\( \Rightarrow \) online BST (with \( \mathcal{O}(1) \) slowdown)

**Corollary**: Greedy is actually an online BST!  
- Conjecture \( \Rightarrow \) dynamically optimal
Proof sketch of theorem:

- Store touched nodes from access in a split tree: split(x) moves x to root & deletes x, leaving 2 split trees in $O(1)$ amortized time — if fully split:
  - really: all n splits in $O(n)$ time (& make split tree on n items in $O(n)$)
  - 2-3-4 tree with min & max pointers can split into $n'$ & $n''$ in $O(\log \min\{n', n''\}) + O(n)$ total merges
  - Use potential $\Phi = \sum_{\text{split tree } T} (|T| - \log |T|)$

$\Rightarrow O(1)$ amortized search cost for split
- Simulate with BST:
  - interleaved min/max search

$\Rightarrow$ BST is "treap of split trees", where heap order is by previous touch & ties mean in split tree (⇒ optimal order)
- Use proof similar to (⇒) above
- By ASS, when touching node in split tree, also touch predecessor & successor in parent split tree ⇒ cheap to reach