How long to process a sequence of searches?

If access frequencies are known in advance and initial tree is arbitrary but fixed, an optimum binary search tree (Knuth-style) minimizes the total search time.

What if access frequencies are not known in advance?

What if tree is allowed to change during the sequence?

Total time for a sequence of accesses

= total search time

(sum of 1+ depth of accessed item, when accessed)

\+ total number of rotations

(between searches arbitrary rotations can be done)

Goal: Compare the minimum-cost off-line strategy with (simple) on-line strategies.

Can an on-line strategy
(no future knowledge)
achieve a performance within a
constant factor of that of the
optimum off-line strategy
(access requests known in advance)?

Splaying: Sleator and Tarjan (1985)

Rotate each edge along an access path.

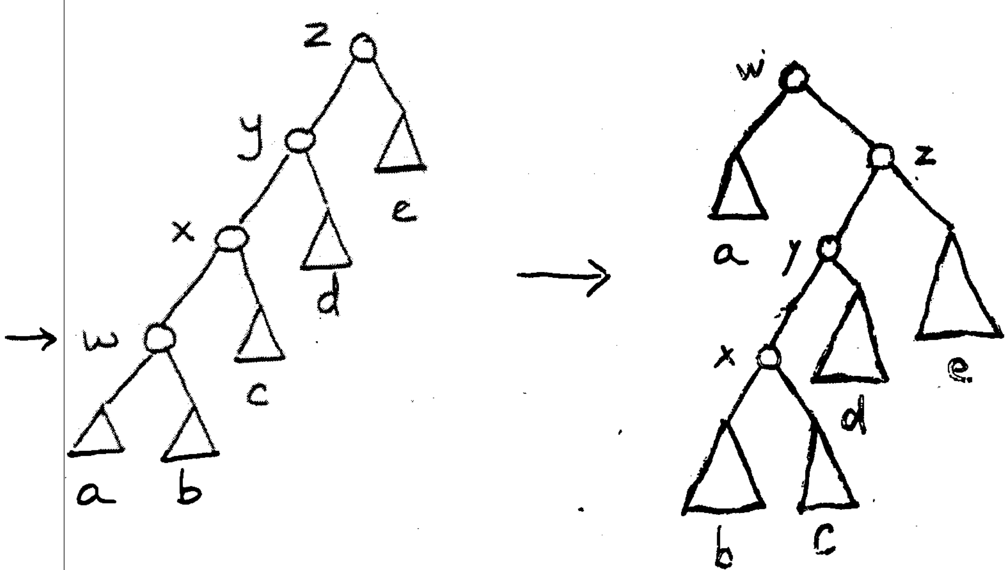
Perform rotations in pairs, roughly bottom-up.

Access path is (roughly) halved, other nodes can move down, but only by a few steps.

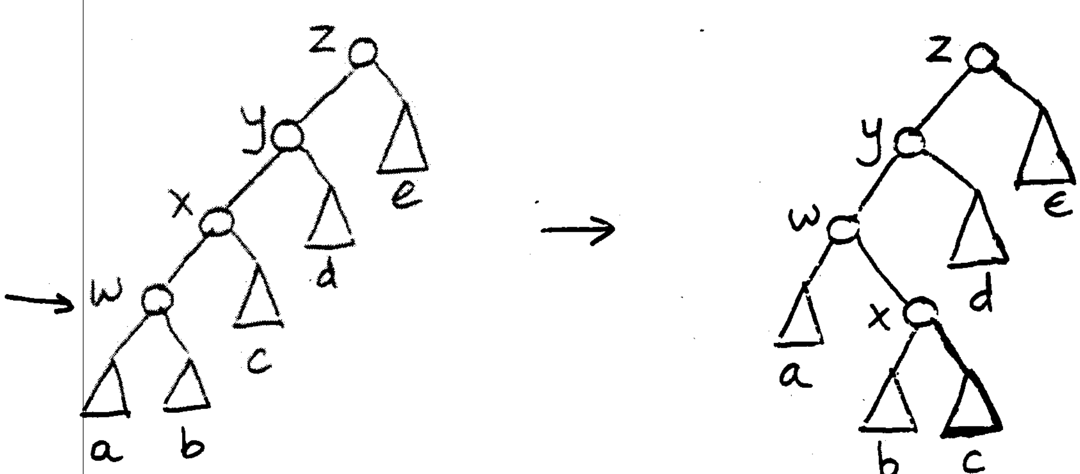
Previous Self-Adjusting Heuristics

(Allen and Munro, Bitner)

1. Move to root: do single rotations all along access path.



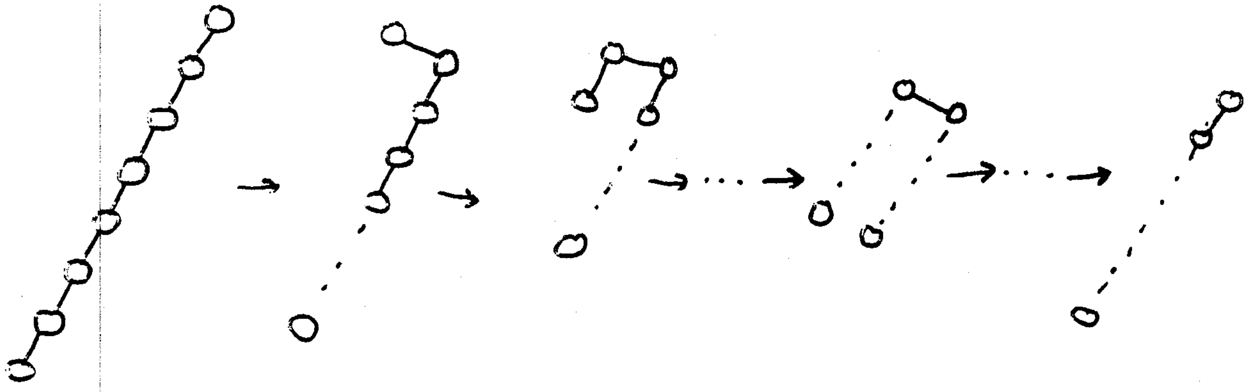
2. Single exchange: do one rotation at parent of accessed node.



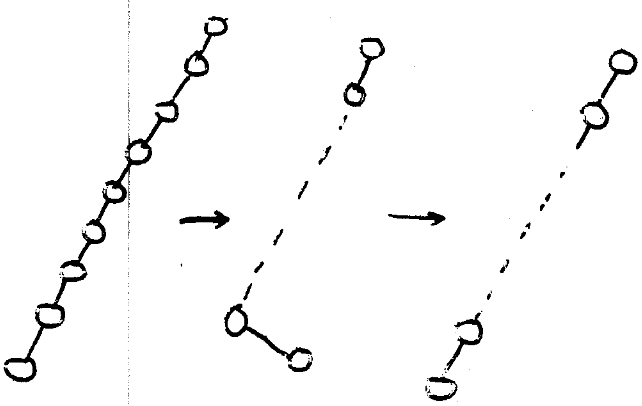
Both are o(n) per operation, even amortized.

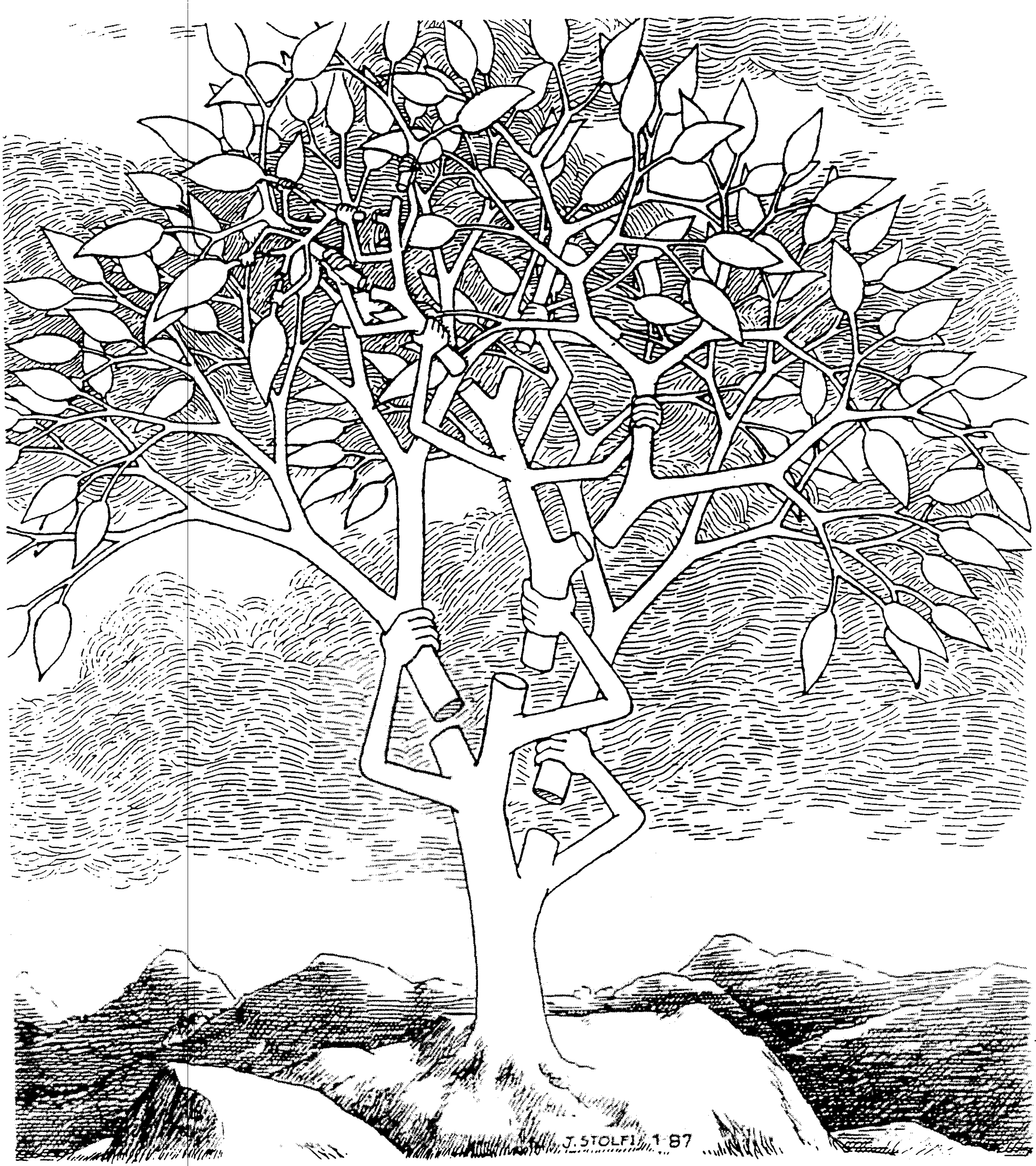
Bad Examples

MTR



SE

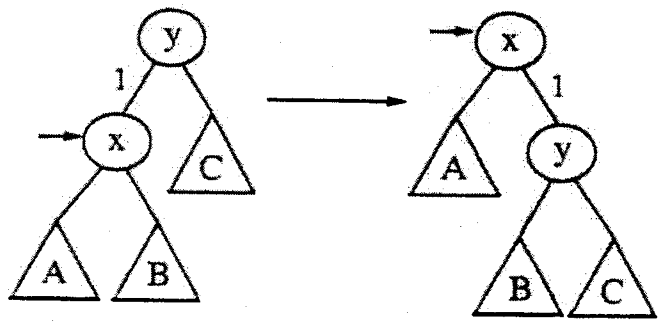




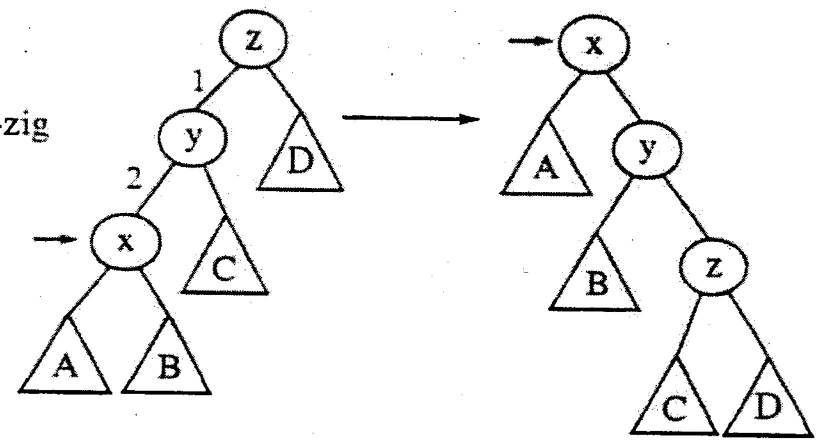
A Self-Adjusting Search Tree

Cases of Splaying

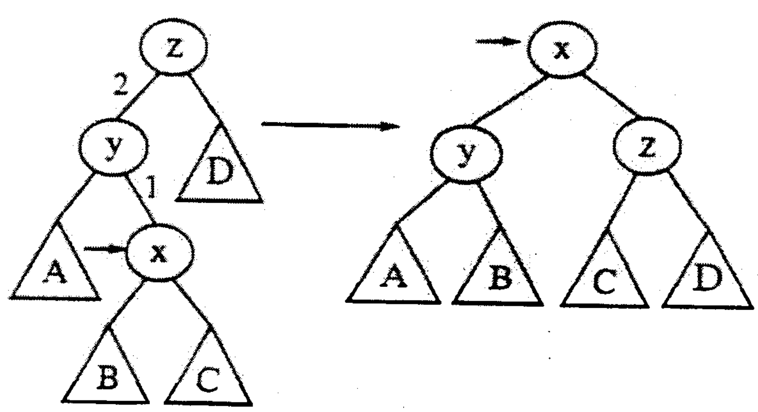
zig



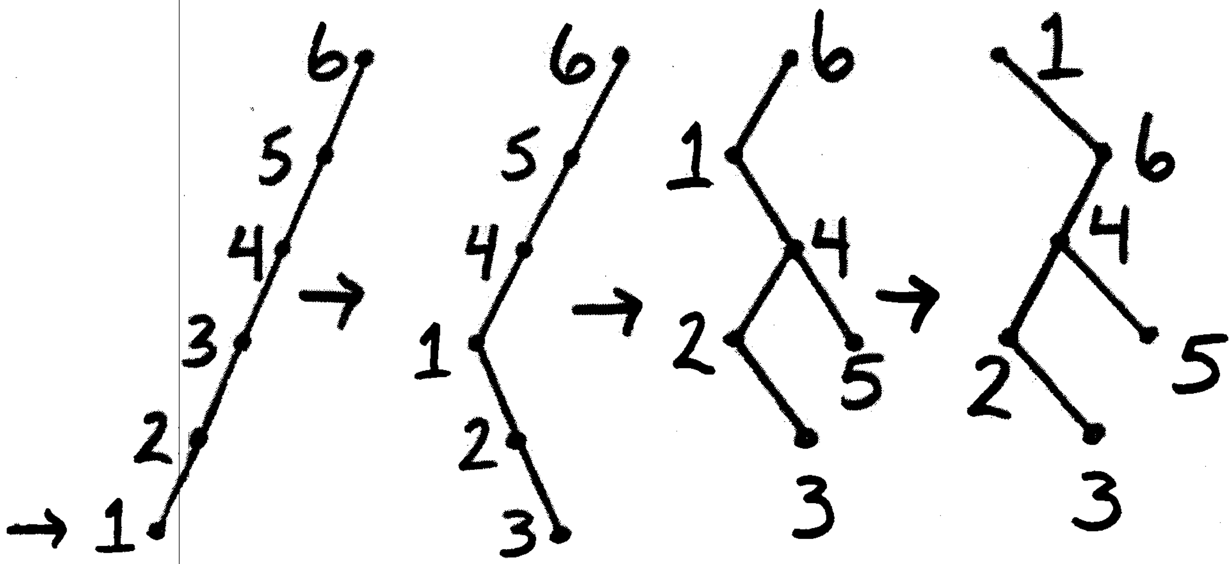
zig-zig

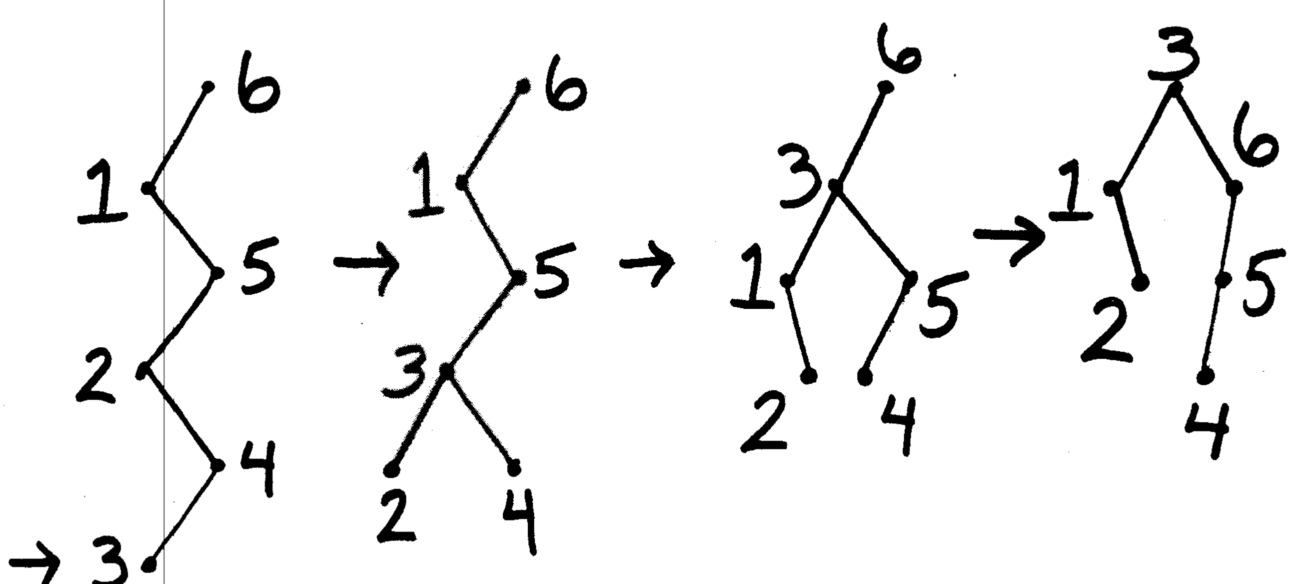


zig-zag

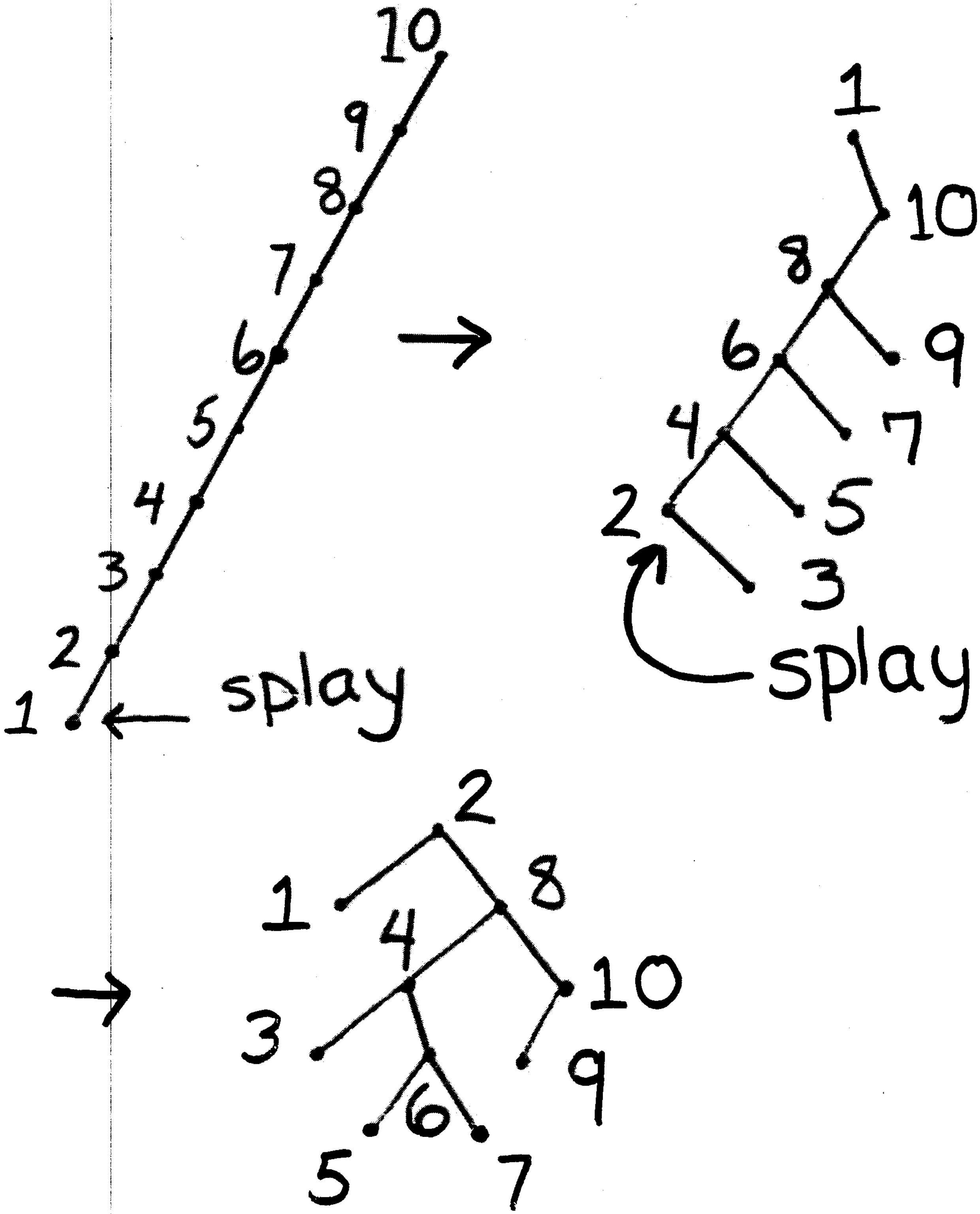


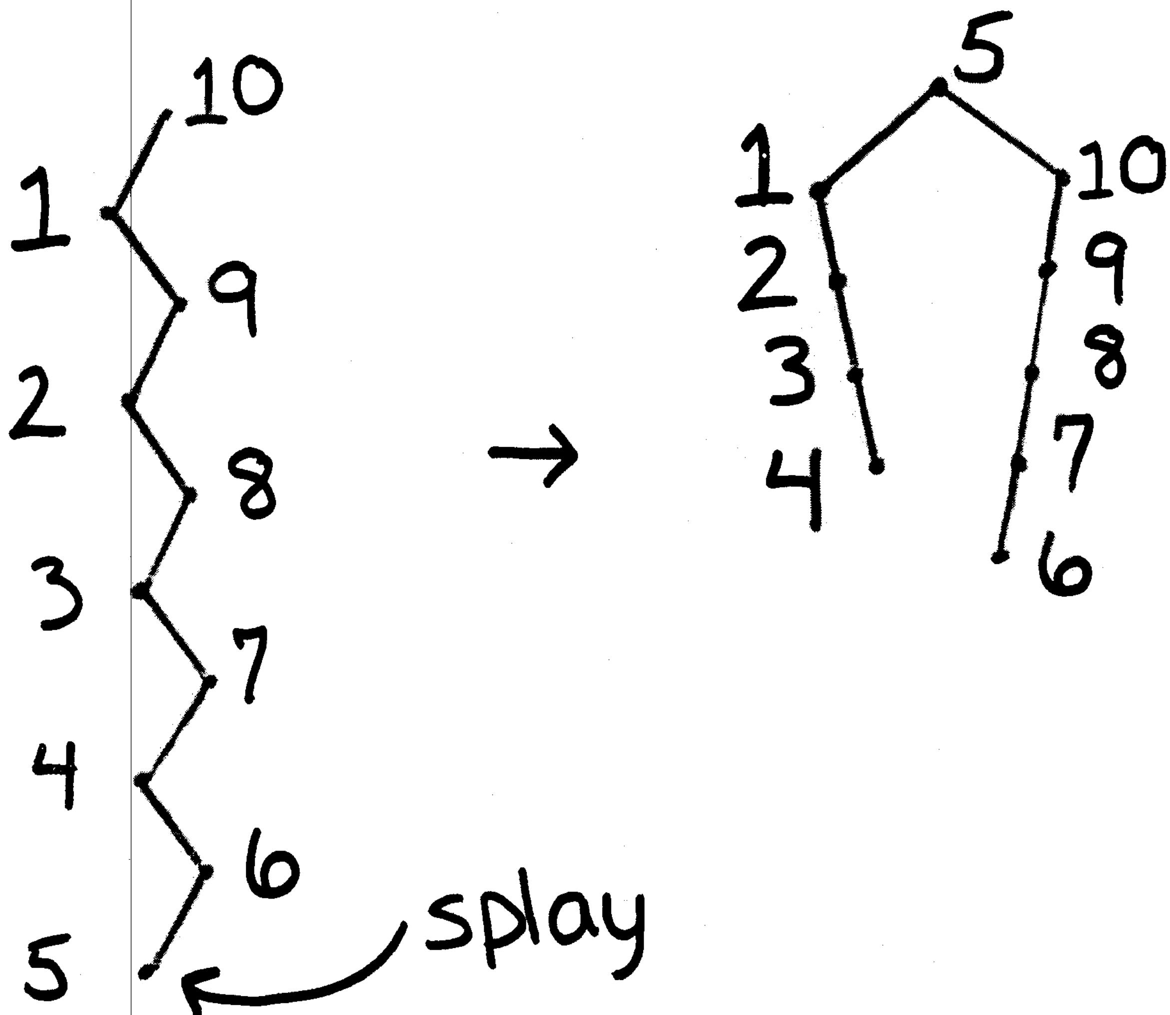
Step by Step Examples





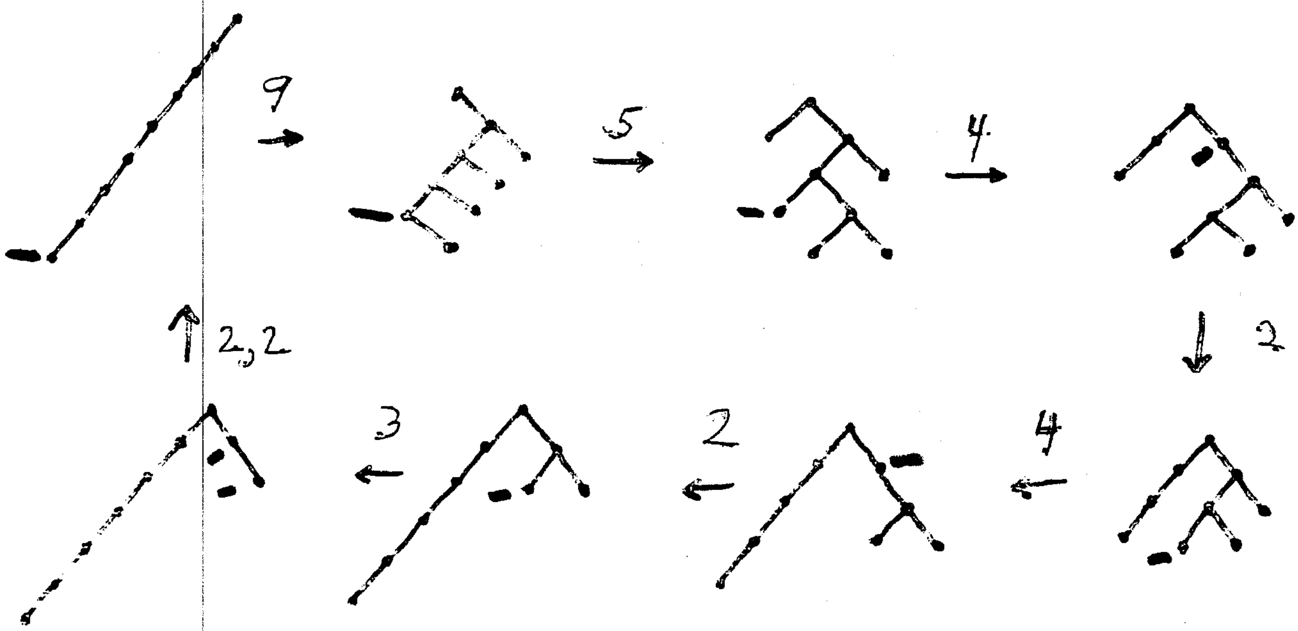
EXAMPLES



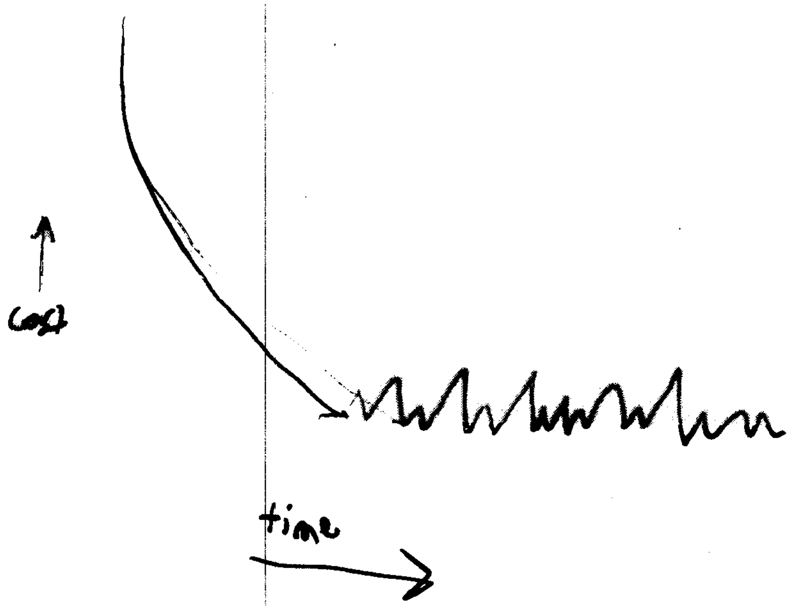


Accessed node moves to root, distance of the other nodes from the root essentially halves.

Splaying in Sequential Order



average = $3\,{}^{2}\!/_{3}$



What is Known

Let m be the number of accesses,
n the number of nodes.
Assume $m \geq n$.

Total time for m accesses $= O(m \log n)$: matches bound for balanced trees.

Total time for any access sequence is within a constant factor of that for an optimum *static* tree.

Total time for n accesses, one per item, in symmetric order, is $O(n)$.

Access Lemma

For any assignment of positive weights to items, the amortized time to access item i is at most

$$3 \log (W/w_i) + 1$$

where W = total weight and the cost of an access is the depth of the accessed node.

Note. The item weights are parameters of the analysis, not of the algorithm.

Potential: define the <u>total</u> <u>weight</u> of a node to be the sum of the individual weights of its descendants, including itself.

The potential of a tree is the sum of the (base-two) logarithms of the weights of its nodes.

$$\Phi = \sum_{i=1}^{n} \log_2 (tw_i)$$

Let $tw(x)$ = sum of weights of all items in subtree of x

rank of $x = r(x) = \log_2 tw(x)$

We shall show:

amortized time of a splay step at x is

$\leq 3(r'(x) - r(x))$ (+1 if zig)

↑ after ↑ before

Then total amortized time of splay is

$\leq 3(r_{final}(x) - r_{initial}(x)) + 1$

$\leq 3(\log W - \log w_i) + 1$

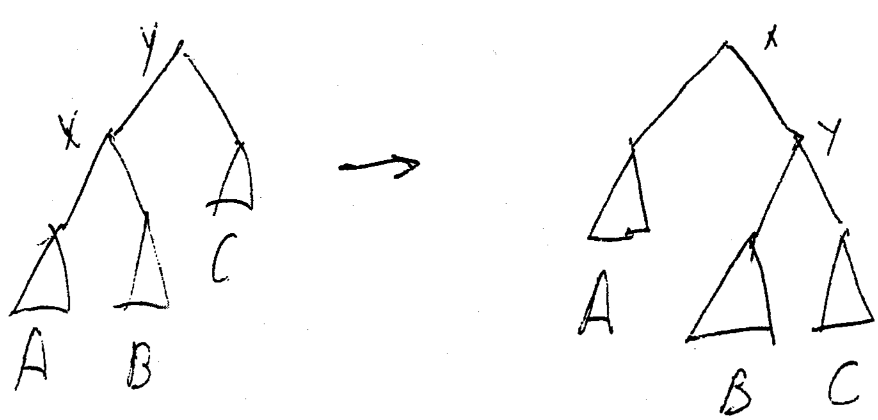
$\leq 3 \log (W/w_i) + 1$

zig

Am. time =

$$1 + r'(y) - r(x)$$

$$\leq 1 + (r'(x) - r(x))$$



zig-zag

$$\text{Am time} = 1 + r'(y) + r'(z) - r(x) - r(y)$$

$$\overset{?}{\leq} 2\left(r'(x) - r(x)\right)$$

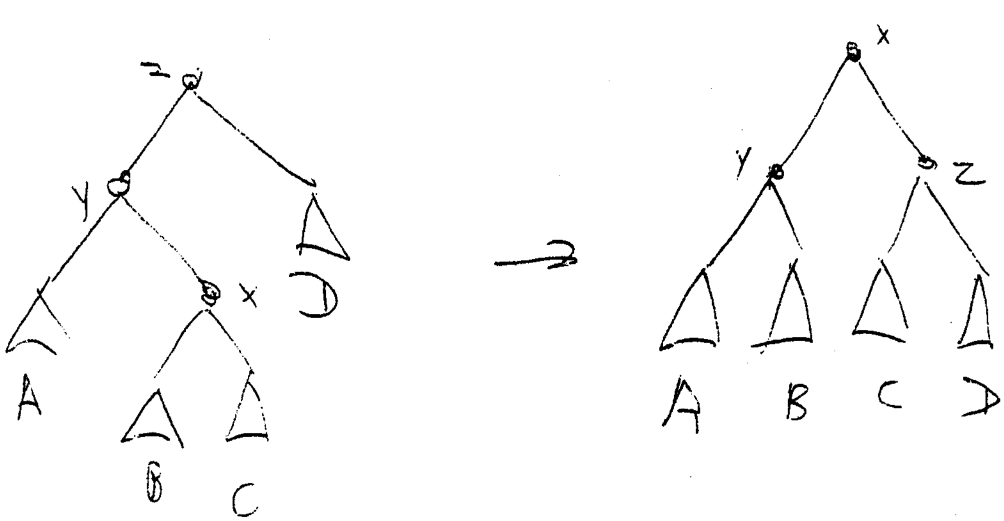
That is, $1 \overset{?}{\leq} \left(r'(x) - r'(y)\right) + \left(r'(x) - r'(z)\right)$

since $r(y) \geq r(x)$

But

$1 \leq r'(x) - r'(y)$ if $tw(y) \leq tw(z)$;

$1 \leq r'(x) - r'(z)$ if $tw(z) \leq tw(y)$.



Analysis of Case 2 (zig-zig) Step

Amortized time of step

$$= 1 + r'(y) + r'(z) - r(x) - r(y)$$

$$\le 1 + r'(x) + r'(z) - 2r(x) \quad \text{since } r'(x) \ge r'(y),\ r(y) \ge r(x)$$

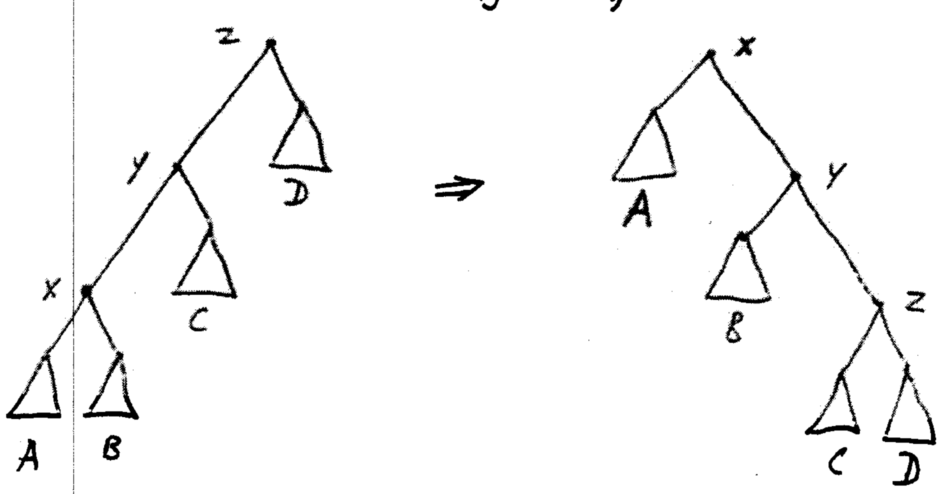
$$\le 3(r'(x) - r(x)) \quad \text{iff}$$

$$2r'(x) - r(x) - r'(z) \ge 1.$$

But $r'(x) \ge \max\{r(x), r'(z)\}$. Also $tw(x) + tw'(z) \le tw'(x)$

Thus $\min\{tw(x), tw'(z)\} \le tw'(x)/2$. I.e. $r'(x) \ge \min\{r(x), r'(z)\} + 1$.

$r(x) = \log tw(x)$



Access lemma holds for variants of splaying, including top-down and move half-way to root methods. For the latter, the constant factor is 2.

Corollaries

Balance Theorem

The total time for m accesses in an n-node tree is $O((m+n)\log(n+2))$.

Static Optimality Theorem

If every item is accessed at least once, the total access time is $O(m + \sum_{i=1}^{n} q_i \log(m/q_i))$, where q_i is the access frequency of item i.

Extension of argument shows that self-adjusting trees are as efficient (to within a constant factor) as optimum trees, over a sequence of operations.

Static Finger Theorem

The total access time is

$$O\left(n \log n + \sum_{j=1}^{m} \log(d(i_j, f) + 2)\right),$$

where f is any fixed item, i_j is the item accessed during the j^{th} access, and $d(i, i')$ is the (symmetric-order) distance between items i and i'.

"Working Set" Theorem

The total access time is

$$O\left(n \log n + \sum_{j=1}^{m} \log(t(i,j)+2)\right),$$

where $t(i,j)$ is the number of different items accessed before access j since the last access of item i.

x y z w y a z . . . x

Thm. Total time to access all items once, in symmetric order, using splaying

$$= O(n).$$

(any initial tree)

Conjecture

Dynamic Optimality

For any access sequence, splaying minimizes the total access time to within a constant factor among dynamic binary search tree algorithms, assuming unit cost per rotation and access cost equal to depth.

(Initial tree is given or $+O(n)$ term)