From O(n α(n)) to O(n α*(n))

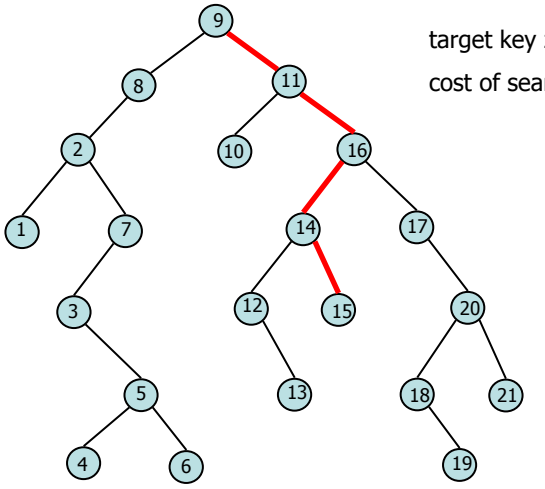
Recent Results for Splay Trees

Joan M. Lucas
The College at Brockport
State University of New York

Western New York Theory Day
May 2, 2008

Based on: “Splay Trees, Davenport-Schinzel Sequences, and the Deque Conjecture”, Seth Pettie, SODA, January 2008

Binary Search Tree - Search Operation



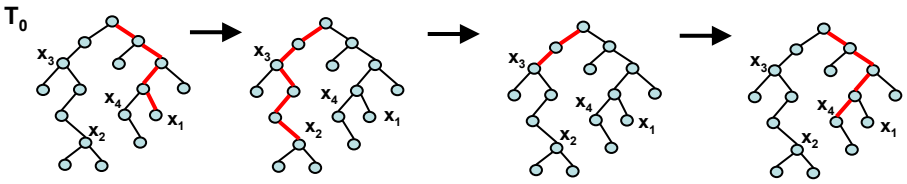
target key z = 15

cost of search = 5

Search of target key z → walk down from the root

Cost of search = depth(z) + 1

Search Problem

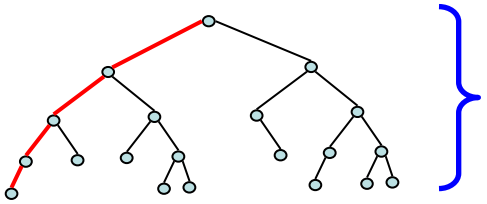


Given binary search tree T_0 and a sequence of target keys X= { x_1, x_2, x_3, …. x_m }

Search for and locate each key at total minimal cost

assume m ≥ n

Balanced Binary Search Trees



height of tree = O(log n)

AVL (height balanced)

Red-black

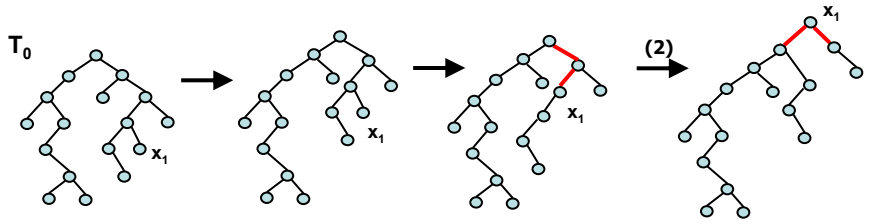
Drawbacks:

- Extra memory (balance factor, color)
- Not adaptive: X= { 1, 1, 1, 1, 1, …. 1 }

cost = O(m log n)

optimal cost = O(n + m)

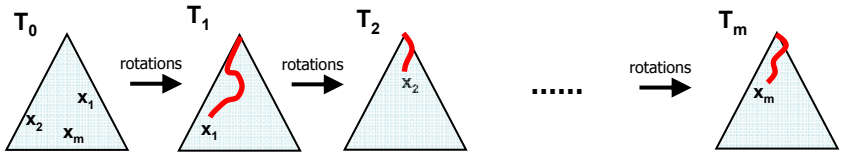
Alter Tree Shape using Rotations



We can alter the shape of the tree by **rotating** an edge

- constant time operation
- preserves symmetric order
- changes the depths of some nodes in the tree

Search Problem using Rotations

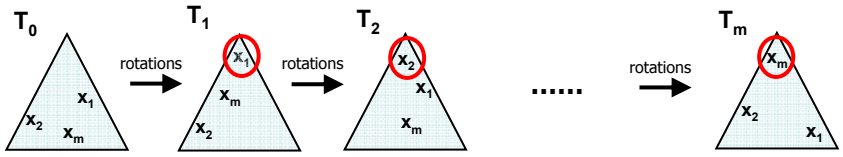


Given binary search tree T_0 and a sequence of target keys X= { x_1, x_2, x_3, ... x_m }

OPT (T_0, X) = cost of the sequence of rotations and searches which minimize the total cost

Nodes higher in the tree are cheaper to find

Locality of Reference Principle

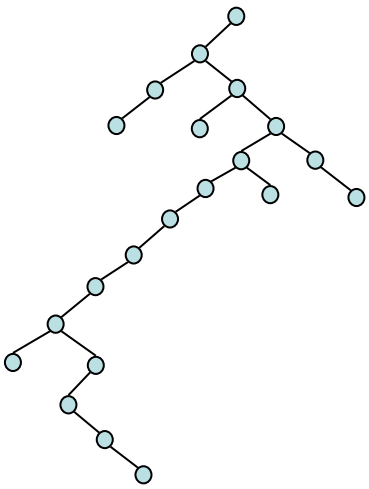


Target key **X** of a search is likely to be a target of another search in near future

So move **X** to the root to make subsequent searches faster

(e.g., LRU, cache, working set)

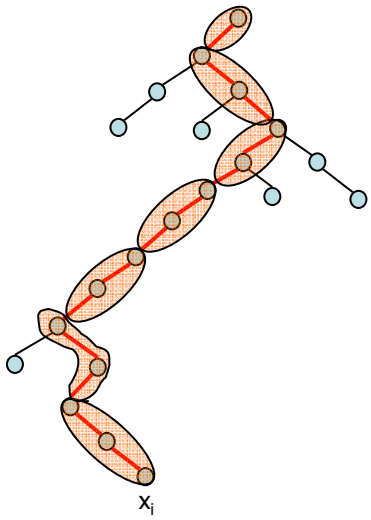
Splay Tree



No limit on the shape of the tree

Reshape the tree after each search to move the target key to the root

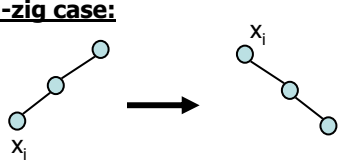
Splay Tree



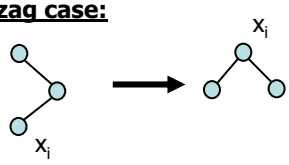
Zig case: odd number of edges on path

final rotation is of edge (x_i, root)

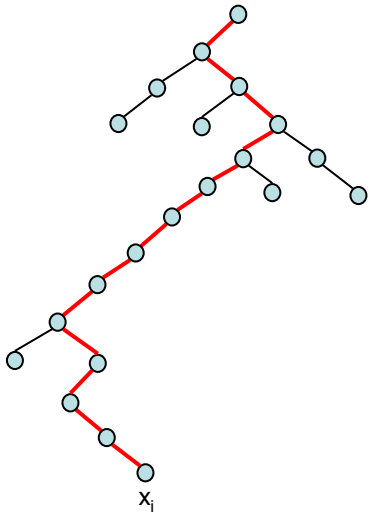
Zig-zig case:

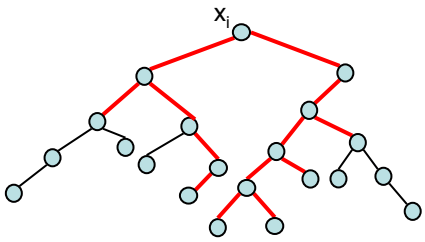


Zig-zag case:



Before and After

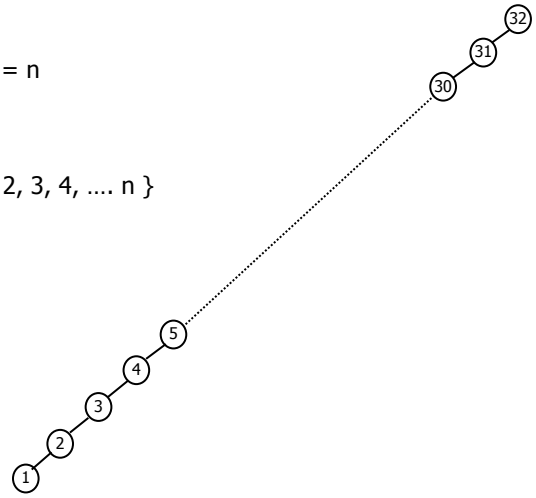




Sequential Access Case

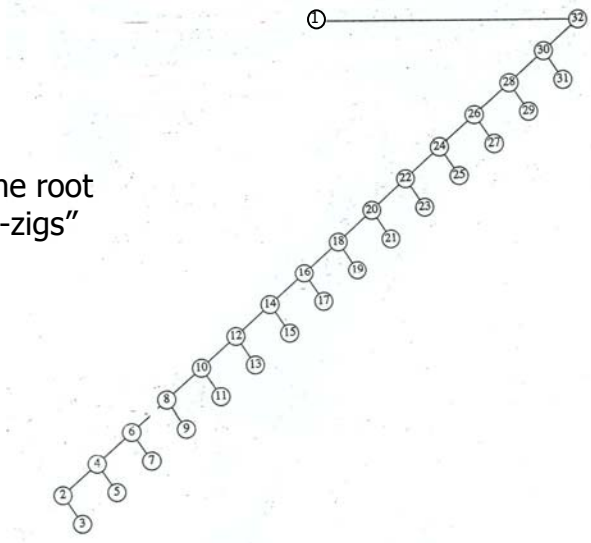
Most extreme initial tree, height = n

sequence of target keys X= { 1, 2, 3, 4, …. n }

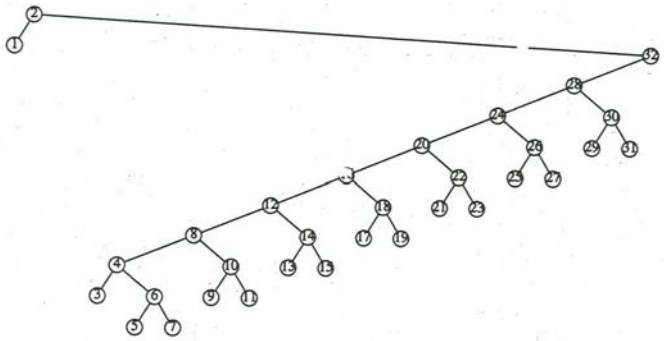


Sequential Access Case

node 1 rotates to the root
using repeated "zig-zigs"

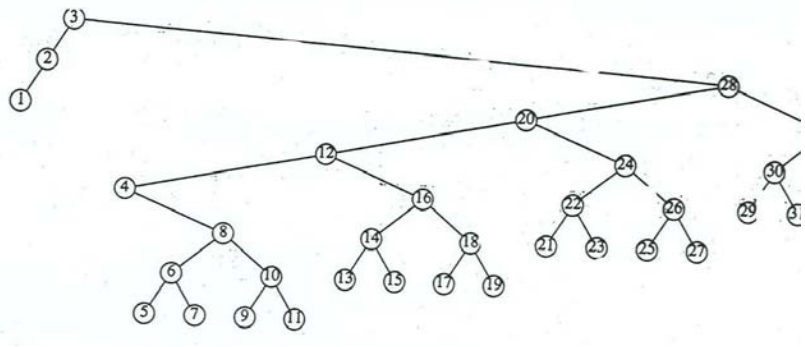


Sequential Access Case



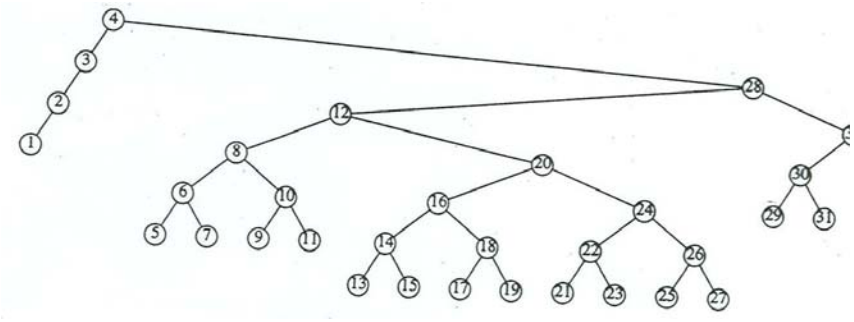
node 2 rotates to the root
using repeated "zig-zigs"

Sequential Access Case



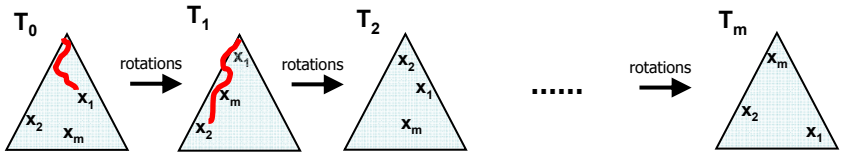
node 3 rotates to the root using repeated "zig-zigs"

Sequential Access Case



Splay trees are "building in balance" automatically

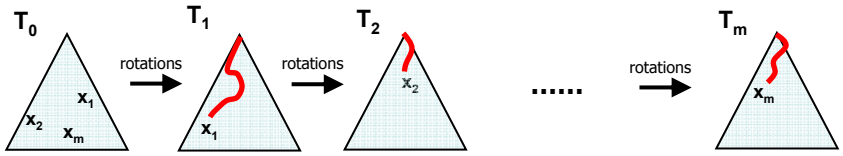
Splay trees match Balanced trees



Claim: Splay trees behave very well over the entire sequence of operations.

Theorem (Sleator and Tarjan, 1985): Given binary search tree T_0 and a sequence of target keys X= { x_1, x_2, x_3, …. x_m }, the total cost of performing these searches is **O (m log n)**

Dynamic Optimality



A binary search tree algorithm **A** is **dynamically optimal** if for every (T_0,X)

$$cost_A(T_0, X) \leq c\ OPT\ (T_0, X)$$

for some constant c.

Conjecture (Sleator and Tarjan, 1985): Splay trees are dynamically optimal.

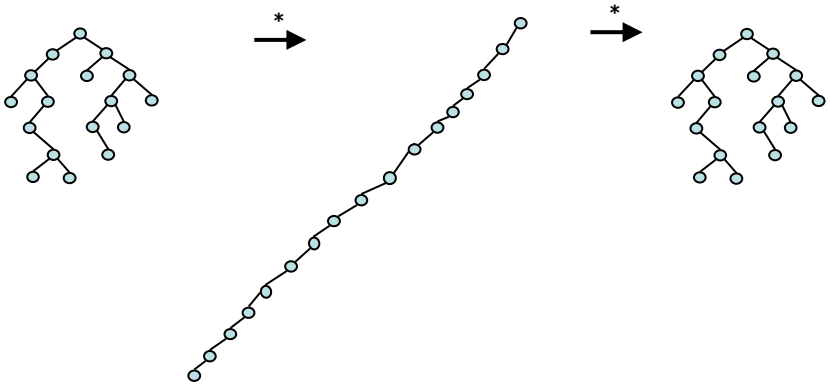
Corollaries of Dynamic Optimality

- **Static Optimality Theorem (1985):** Let q_i be the number of times i is accessed, then the total access time for splay trees is $O (m + \sum q_i \log (m/q_i))$
- **Working Set Theorem (1985):** Let t_j be the number of distinct items accessed between the last access of i(j), and the current access, then the total access time for splay trees is $O(n \log n + m + \sum \log(t_j + 1))$
- **Dynamic Finger Theorem (2000):** the total access time for splay trees is $O(m + \sum \log(| i(j) – i(j+1) | + 1)$

What Access Sequences are Easy?

- **Static Optimality Theorem (1985):** Let q_i be the number of times i is accessed, then the total access time for splay trees is $O (m + \sum q_i \log (m/q_i))$
- **Working Set Theorem (1985):** Let t_j be the number of distinct items accessed between the last access of j, and the current access, then the total access time for splay trees is $O(n \log n + m + \sum \log(t_j + 1))$
- **Dynamic Finger Theorem (2000):** the total access time for splay trees is $O(m + \sum \log(| i(j) – i(j+1) | + 1)$
- **Sequential Access Theorem (1985):** The total time for X = { 1, 2, 3, … n } is O(n)

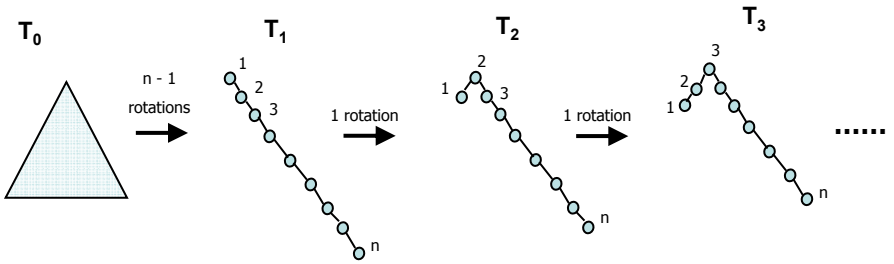
Unfolding a Tree into Vine



Any tree can be **unfolded** into a **left vine** using at most (n-1) rotations

The left vine tree can be **folded** into any tree using at most (n-1) rotations

Sequential Access is Easy



OPT (T_0, X) ≤ (n - 1) + n – 1 = O(n)

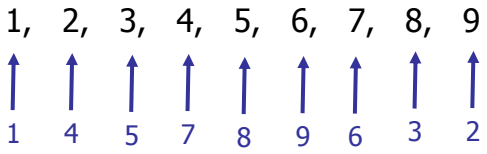
when X = { 1, 2, 3, 4, 5, …. n }

Theorem (Tarjan 1985): Given binary search tree T_0 the total cost of performing a sequence of n accesses in sequential order is **O (n)**

Deque Access

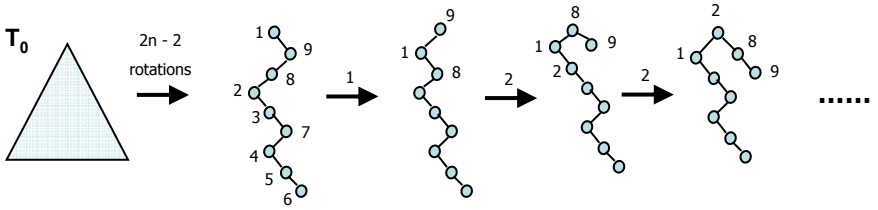
Deque (double-ended queue)

Access each element once in an "from the outside in" fashion



X = { 1, 9, 8, 2, 3, 7, 4, 5, 6 }

Deque Access is Easy



OPT (T_0, X) ≤ (2n - 2) + 1 + 2(n-2) = O(n)

Theorem (Sundar 1992): Given binary search tree T_0 the total cost of performing a sequence of n deque-ordered accesses is **O (n α (n))**

Ackermann's function:

$A_1(j) = 2j$

$A_{i+1}(j) = A_i^{(j)}(1) = A_i\ A_i\ A_i\ A_i\ (1)$

$A_4(n)$	2	4	2^{16}	$2^{2^{\cdots^{2}}}$ (65,536)			....
$A_3(n)$	2	2^2 = 4	2^2^2 =16	2^2^2^2 =2^{16}	$2^{65,536}$		....
$A_2(n)$	2	4	8	16	32	64	128 ....
$A_1(n)$	2	4	6	8	10	12	14 ....
n	1	2	3	4	5	6	7 ....

Inverse Ackermann: α(n)

$A_4(n)$	2	4	2^{16}	$2^{2^{\cdots^{2}}}$ (65,536)			....
$A_3(n)$	2	2^2 = 4	2^2^2 =16	2^2^2^2 =2^{16}	$2^{65,536}$		....
$A_2(n)$	2	4	8	16	32	64	128 ....
$A_1(n)$	2	4	6	8	10	12	14 ....
n	1	2	3	4	5	6	7 ....

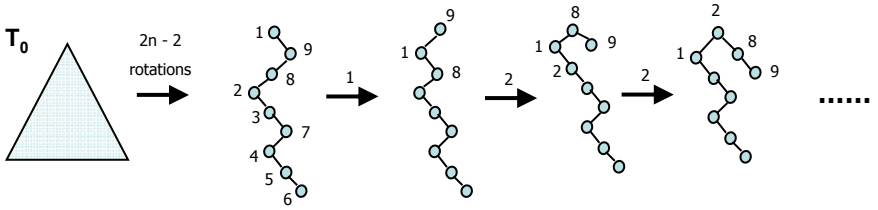
Define: $\alpha(n) = \min \{ i \geq 1 : A_i(4) \geq \log(n) \}$

Iterated Inverse Ackermann: α*(n)

$A_4(n)$	2	4	2^{16}	$2^{2^{\cdots^{2}}}$ (65,536)			....
$A_3(n)$	2	2^2 = 4	2^2^2 =16	2^2^2^2 =2^{16}	$2^{65,536}$		....
$A_2(n)$	2	4	8	16	32	64	128 ....
$A_1(n)$	2	4	6	8	10	12	14 ....
n	1	2	3	4	5	6	7 ....

Define: $\alpha^*(n) = \min \{ i \geq 1 : \underbrace{\alpha\,\alpha \ldots \alpha}_{i}\,\alpha(n) \leq 2 \}$

New Result

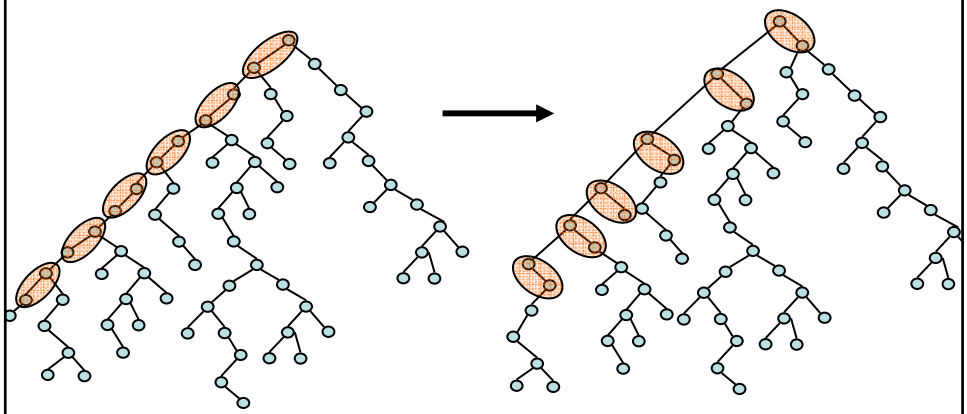


OPT (T_0, X) ≤ (2n - 2) + 1 + 2(n-2) = O(n)

Theorem (Sundar 1992): Given binary search tree T_0 the total cost of performing a sequence of n deque-ordered accesses is **O (n α (n))**

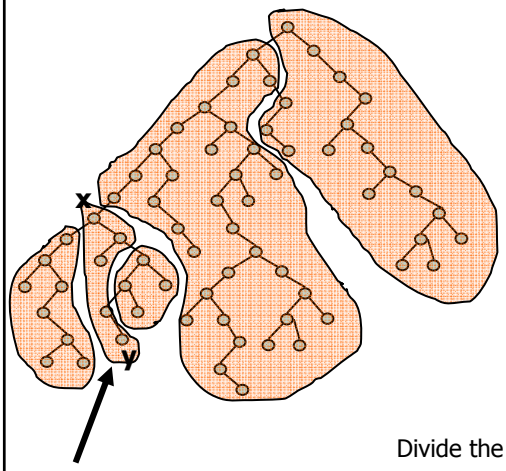
Theorem (Pettie 2008): Given binary search tree T_0 the total cost of performing a sequence of n deque-ordered accesses is **O (n $α^*$ (n))**

What is the cost of deque-splaying?



WLOG : Splay is "spinal" along the left-path (not necessarily to the root)

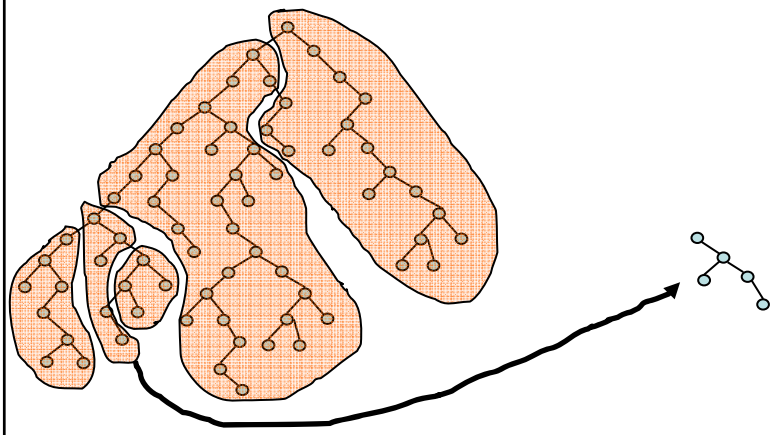
Sundar: divide-and-conquer



Divide the tree into "blocks" of consecutive keys.

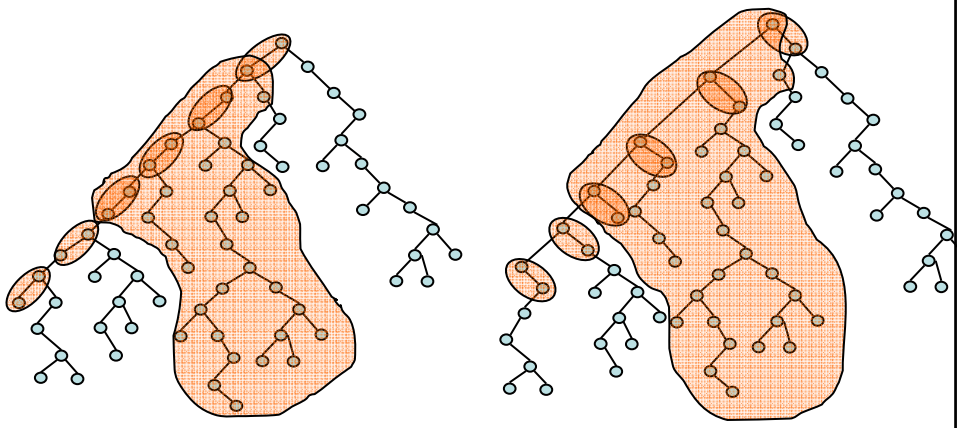
B[x,y]

Sundar: divide-and-conquer



Each block corresponds to a well-defined sub-tree

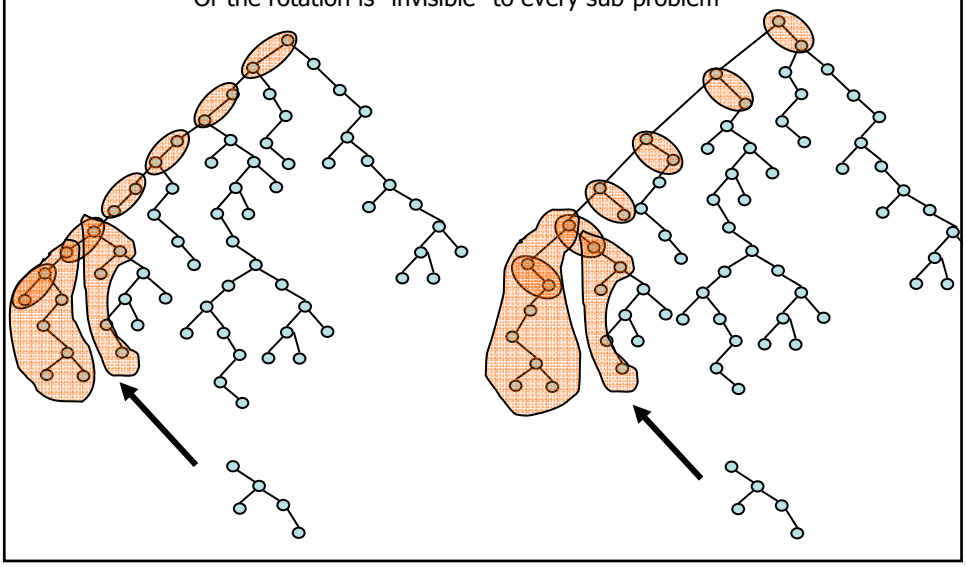
Sub-problem rotations



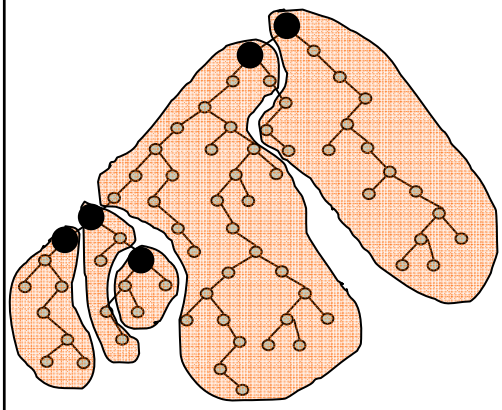
Each rotation is either entirely inside a sub-problem,
forming a "deque splaying" operation in that sub-problem

Or cross-block rotations

Or the rotation is "invisible" to every sub-problem



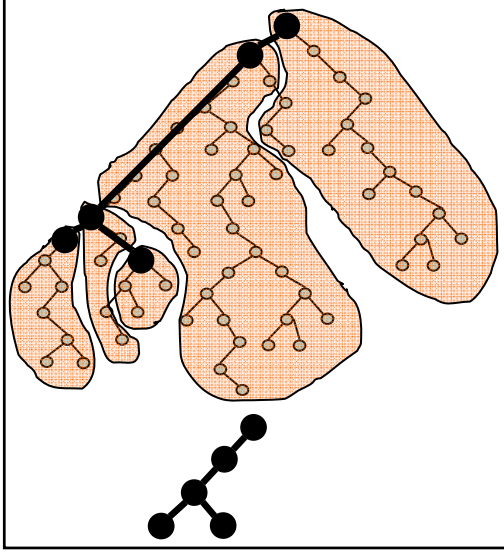
Accounting for cross-block rotations



We need to account for the rotations that are between sub-problems

Shrink every sub-problem into a single node, at the common ancestor of all nodes in that block.

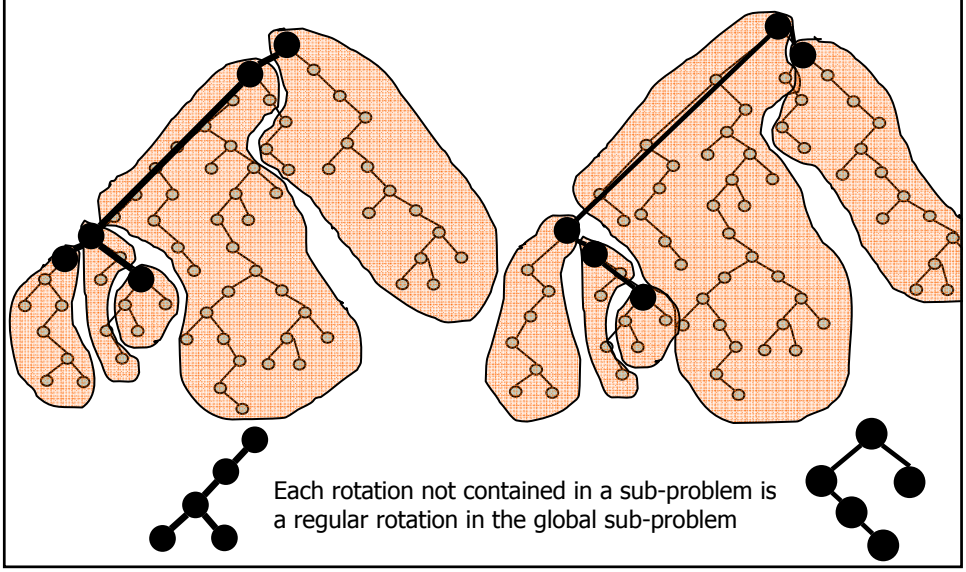
Global sub-problem



Set the parent of each "block subtree root" to be the nearest black ancestor

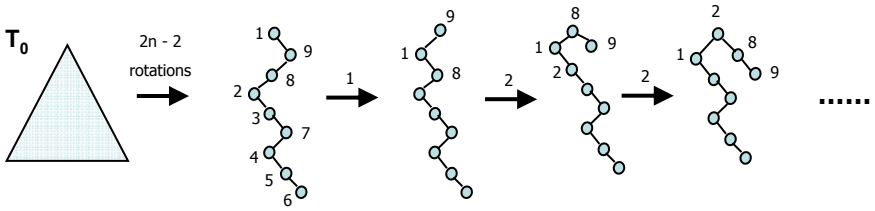
This creates a well-defined binary search tree of these "block roots"

How does a splay effect global sub-problem?



Each rotation not contained in a sub-problem is a regular rotation in the global sub-problem

Seth Pettie (2008): α to α *



NEW TECHNIQUE using Davenport-Schinezel sequences

OPT (T_0, X) ≤ (2n - 2) + 1 + 2(n-2) = O(n)

Theorem (Sundar 1992): Given binary search tree T_0 the total cost of performing a sequence of n deque-ordered accesses is **O (n α (n))**

Theorem (Pettie 2008): Given binary search tree T_0 the total cost of performing a sequence of n deque-ordered accesses is **O (n $α^*$ (n))**

Davenport-Schinzel Sequences (1965)

A s-DS sequence is any finite sequence

$$u = a_1\, a_2\, a_3\, a_4 \ldots\ldots a_l$$

over the infinite alphabet A = { 1, 2, 3, 4, …. } such that:

- u has no immediate repetitions
- u does not contain a sub-sequence isomorphic to v = ababa$\underbrace{\text{ba}}_{s}$, (i.e., no alternating sub-sequence of length s)

Extremal function $\lambda_s(n)$

$\lambda_s(n) = \max \{ |u| : u \text{ is an } (s+2)\text{-DS sequence and } ||u|| \leq n \}$

What is the longest sequence you can form, using only n symbols, with no immediate repetitions, and avoiding the sub-sequence aba…..ba?

$\underbrace{\text{aba}\ldots\ldots\text{ba}}_{s+2}$

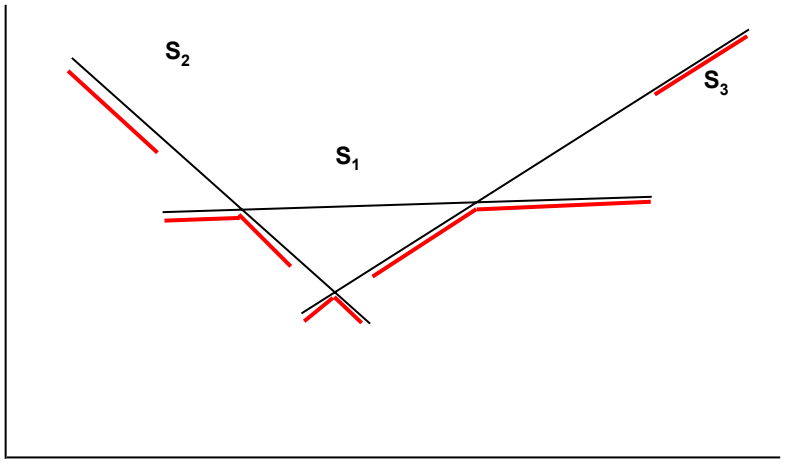
Extremal function $\lambda_2(3)$

$\lambda_2(3) = \max \{ |u| : u \text{ is an 4-DS sequence and } ||u|| \leq 3 \}$

What is the longest sequence you can form, using only 3 symbols, with no immediate repetitions, and avoiding the sub-sequence abab?

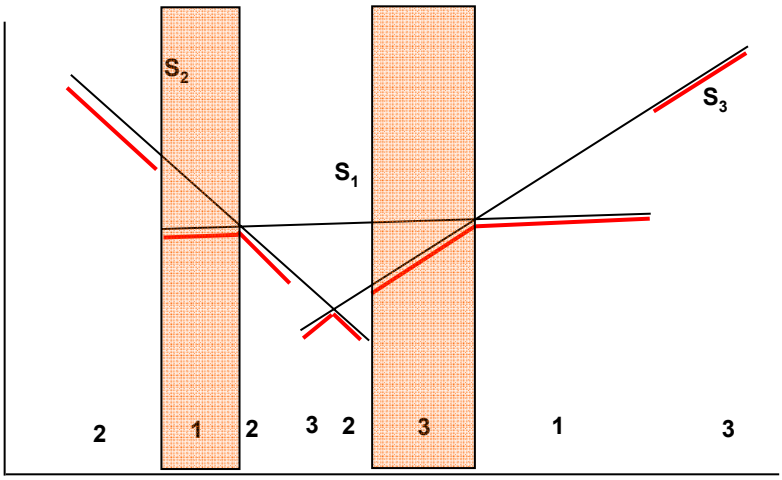
λ, 1, 12, 121, 1213, 12131, 123, 1231, 1232, 12321

Geometric application



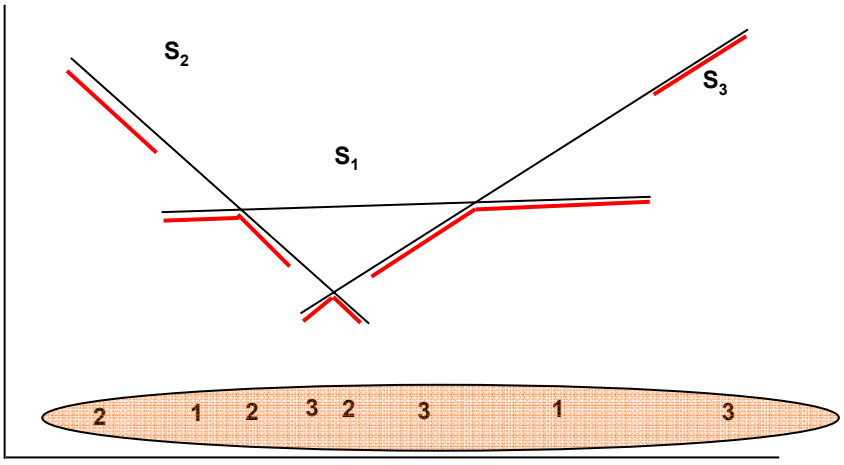
Consider the "lower envelope" of these segments

Geometric application



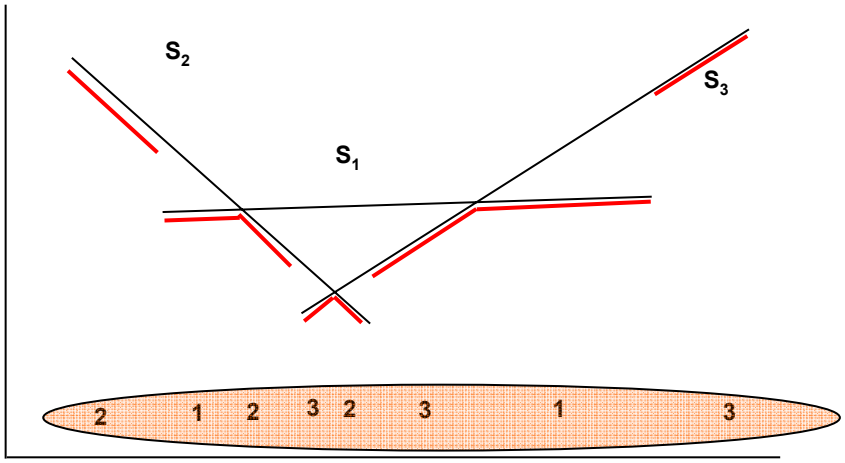
Label each region by the line segment S_i that is minimal on that region

Geometric application



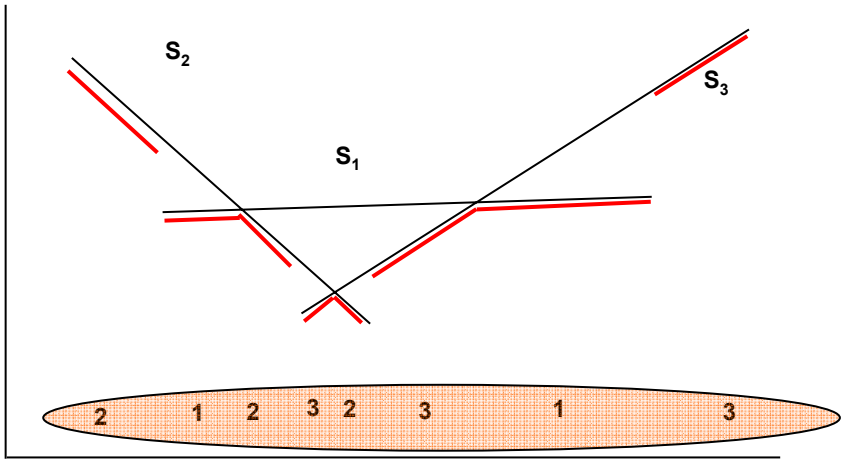
Sequence: 2, 1, 2, 3, 2, 3, 1, 3 cannot contain subsequence "ababa"

$\lambda_3(n)$ is complexity of lower envelope



$\lambda_3(n)$ = max { |u| : u is a 5-DS sequence and ||u|| ≤ n }

$\lambda_s(n)$



$\lambda_s(n)$ = complexity when the S_i can intersect ≤ (s-2) times

$\lambda_2(n)$ is linear

Theorem (Davenport-Schnizel 1965): $\lambda_2(n) = 2n - 1$

The longest possible sequence using n symbols, with no immediate repetition and avoiding the sub-sequence abab, has length 2n - 1

Proof: $2n - 1 \leq \lambda_2(n) \leq 2n - 1$

$\lambda_2(n) \geq 2n - 1$ → 1, 2, 3, …… n-1, n, n-1, ……. 2, 1

Agarwal, Sharir, and Shor (1989)

What is the longest sequence you can form, using only n symbols, with no immediate repetitions, and avoiding the sub-sequence aba…..ba?

s+2

	$\lambda_1(n)$	n
	$\lambda_2(n)$	2n - 1
	$\lambda_3(n)$	$\Theta (n \alpha(n))$
	$\lambda_4(n)$	$\Theta (n 2^{\alpha(n)})$
$\Omega (n 2^{\alpha(n)})$	$\lambda_5(n)$	$O(n \alpha(n)^{(1+o(1))\,\alpha(n)})$
$O(n 2^{(1+o(1))\,\alpha(n)^2/2})$	$\lambda_6(n)$	$O(n 2^{(1+o(1))\,\alpha(n)^2})$

Deque Splaying

If X is a deque-ordered sequence, then

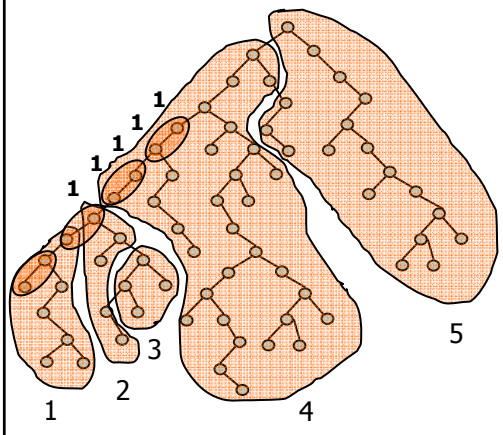
$$OPT(T_0, X) = O(n)$$

Theorem (Sundar 1992): Given binary search tree T_0 the total cost of performing a sequence of n deque-ordered accesses is **$O(n \alpha(n))$**

Theorem (Pettie 2008): Given binary search tree T_0 the total cost of performing a sequence of n deque-ordered accesses is **$O(n \alpha^*(n))$**

Proof: Characterize the cost of Deque Splaying as a Davenport-Schnizel sequence, then cut-and-paste the results of Agarwal et. al.

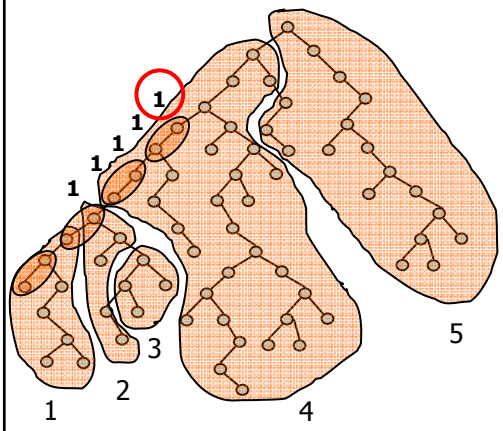
Describe rotations as DS sequence



Same idea of dividing nodes into consecutive sub-problems

Non-sub-problem nodes that are touched by a splay from a node in sub-problem j are "affiliated" with j

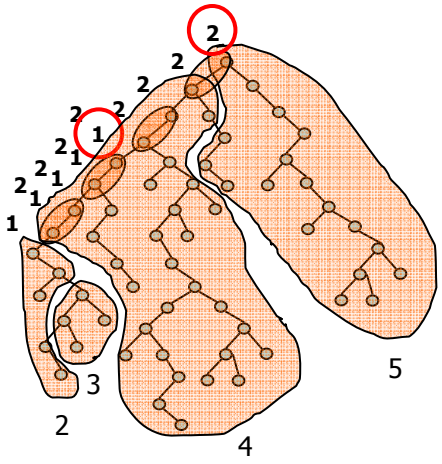
Describe rotations as DS sequence



Non-sub-problem nodes that are touched by a splay from a node in sub-problem j are "affiliated" with j

Node receives this label if no ancestor is in the same block, or has the same affiliation.

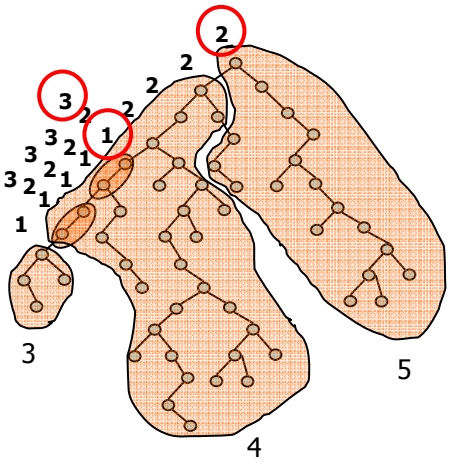
Describe rotations as DS sequence



Non-sub-problem nodes that are touched by a splay from a node in sub-problem j are "affiliated" with j

Node receives this label if no ancestor is in the same block, or has the same affiliation.

Describe rotations as DS sequence



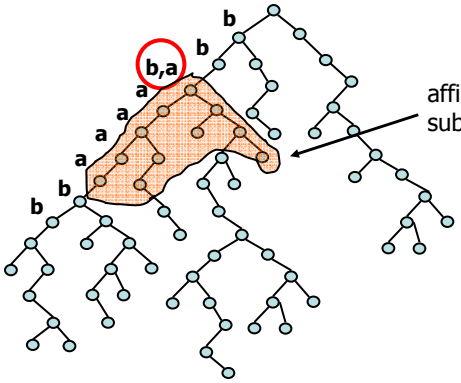
Non-sub-problem nodes that are touched by a splay from a node in sub-problem j are "affiliated" with j

Node receives this label if no ancestor is in the same block, or has the same affiliation.

Multiple labels on a node are given in descending order

Sequence: …… 3, 1, 2, …….

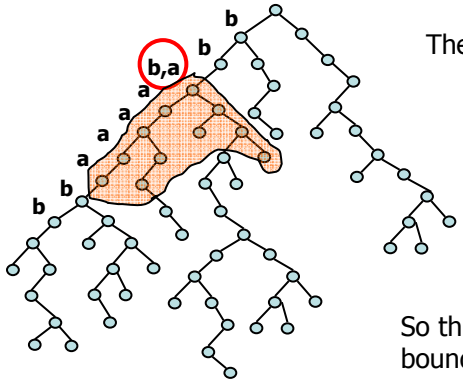
Forbidden sub-sequence babba



Cannot have second appearance of 'b' after the appearance of an 'a'

Sequence: ……. b …. a ….. b ……. b …….. a …….. …….

Forbidden sub-sequence babba



Therefore abababa is also forbidden

So this is a 5-DS sequence, whose length is bounded by $O(n \alpha(n)^{(1+o(1))\alpha(n)})$

Sequence: ……. b …. a ….. b ……. b …….. a …….. …….

Deque Splaying

If X is a deque-ordered sequence, then OPT (T_0, X) = O(n)

Theorem (Sundar 1992): Given binary search tree T_0 the ~~total cost of~~ performing a sequence of n deque-ordered accesses is **$O (n \alpha (n))$**

Theorem (Pettie 2008): Given binary search tree T_0 the ~~total cost of~~ performing a sequence of n deque-ordered accesses is **$O (n \alpha^* (n))$**

Proof: Characterize the cost of Deque Splaying as a Davenport-Schnizel sequence, then cut-and-paste the results of Agarwal et. al.

Improvements? Use generalized DS sequences (with known linear bounds)?

Generalized DS Sequences

$Ex(v,n) = \max \{ |u| : u \text{ does not contain } v,$

$u \text{ is } ||v|| \text{ regular},$

$\text{and } ||u|| \leq n \}$

What is the longest sequence you can form, using only n symbols, with no symbol repetition in any ||v|| substring, and avoiding the sub-sequence v?

Theorem (Klazar, 1995) : Ex(abba, n) = O(n)

Ex(abcdabcd, n) = O(n)

Ex(aabbccaabbcc, n) = O(n)

Open Problem: Ex(abacabc, n) = ??? Linear? Super-linear?