Finger Search Trees

Gerth Stølting Brodal

Finger Search Trees

A finger search tree is a data structure to store a sorted list of elements.

Fingers are pointers to particular elements in the list. Operations are done relative to the fingers.

Operations:

search (S, x, f)
insert (S, x, f)
delete (S, x, f)
create-finger (S, x)
destroy-finger (S, f)

$S =$ join (S_1, S_2)
$(S_1, S_2) =$ split (S, x, f)

Let d denote the distance of x from f in the list. The operations should take the following time:

search, insert, delete : $O(\log d)$
create-finger, destroy-finger : $O(1)$
join, split : $O(\log d + \log \min\{|S_1|, |S_2|\})$

Adaptive Sorting

– an application of finger search trees

The Problem:

Input: X a vector of elements

Output: X sorted

Algorithm: Should be adaptive to the presortedness of X.

Measures of presortedness:

$\mathrm{Inv}(X)$ = # pairwise inversions in X

$\mathrm{SMS}(X) = \min\{k \mid X$ can be partitioned into k monotone sequences$\}$

(SMS = Shuffled Monotone Subsequence)

Optimally adaptive algorithms with respect to a measure of presortedness match the corresponding lower bound for the number of comparisons within a constant factor.

Marge sort

[Moffat, Petersson, Wormald]

14	15	13	6	1	4	5	10	12	16	8	3	7	9

↓

Divide

↙ ↘

14	15	13	6	1	4	5

10	12	16	8	3	7	9

↓ Recursion ↓

1	4	5	6	13	14	15

3	7	8	9	10	12	16

↘ ↙

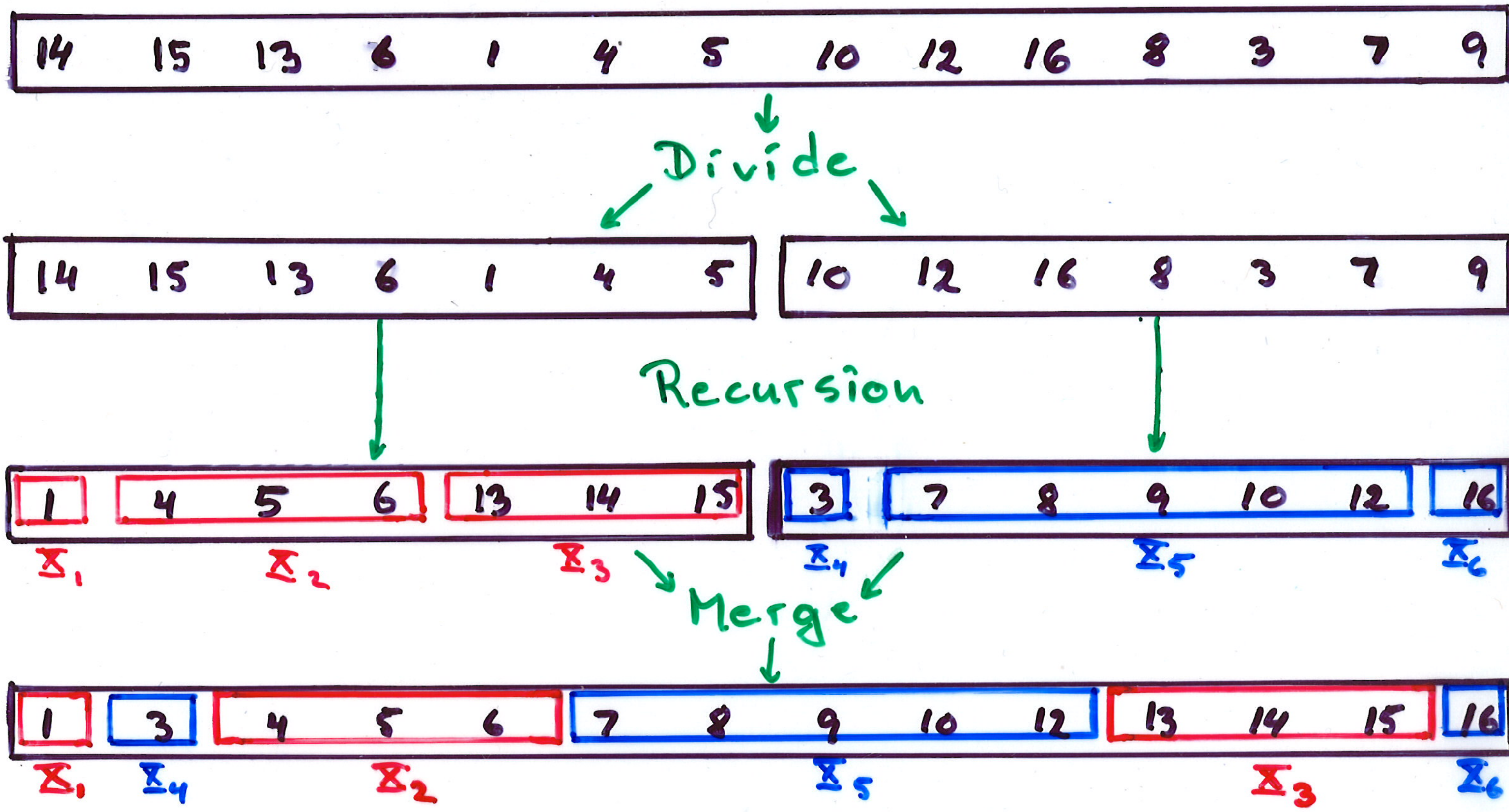
Merge

↓

1	3	4	5	6	7	8	9	10	12	13	14	15	16

Margesort

[Moffat, Petersson, Wormald]



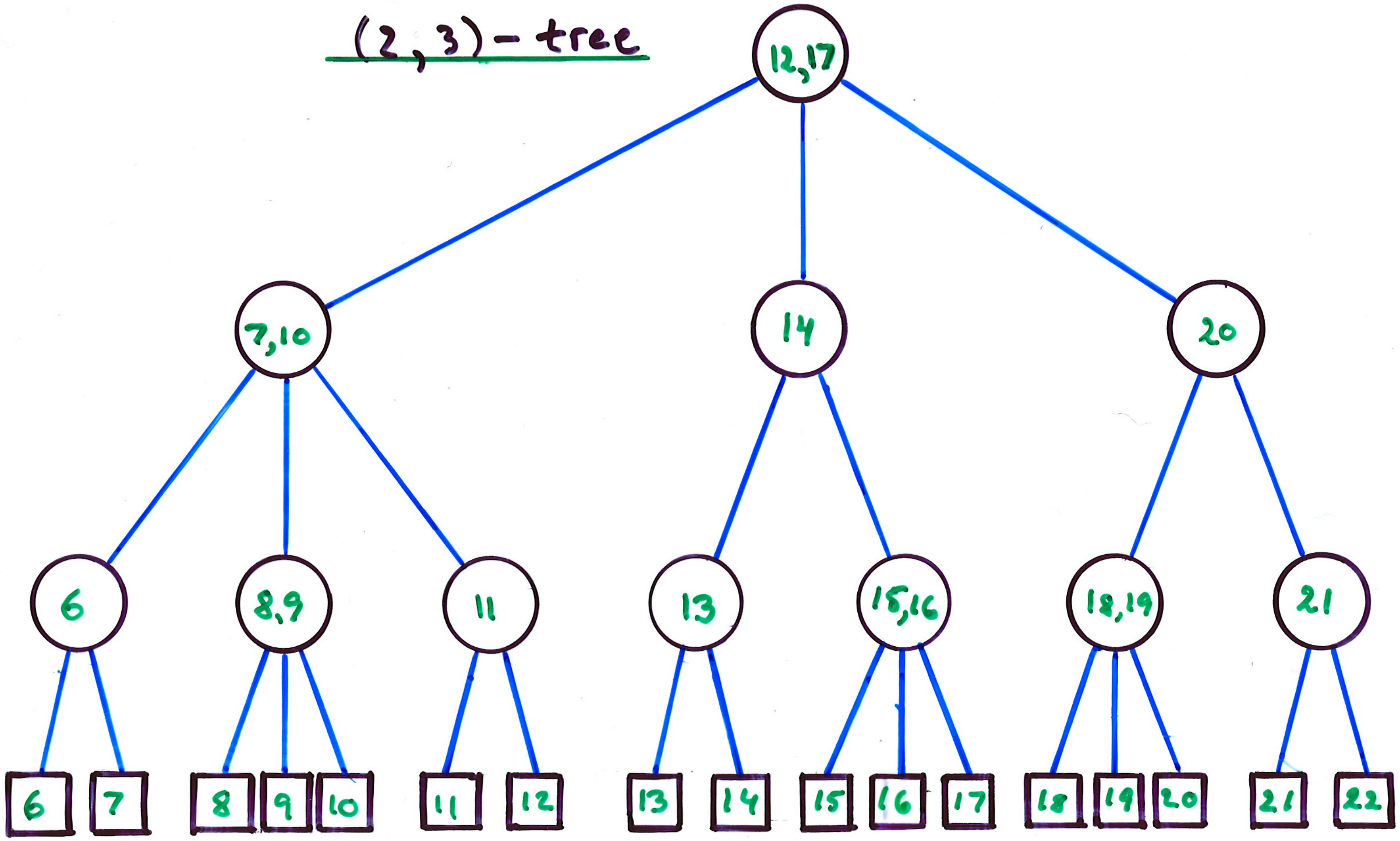
Merge can be done in worstcase time:

$$O\left(\sum_{i=1}^{s} \log |X_i|\right)$$

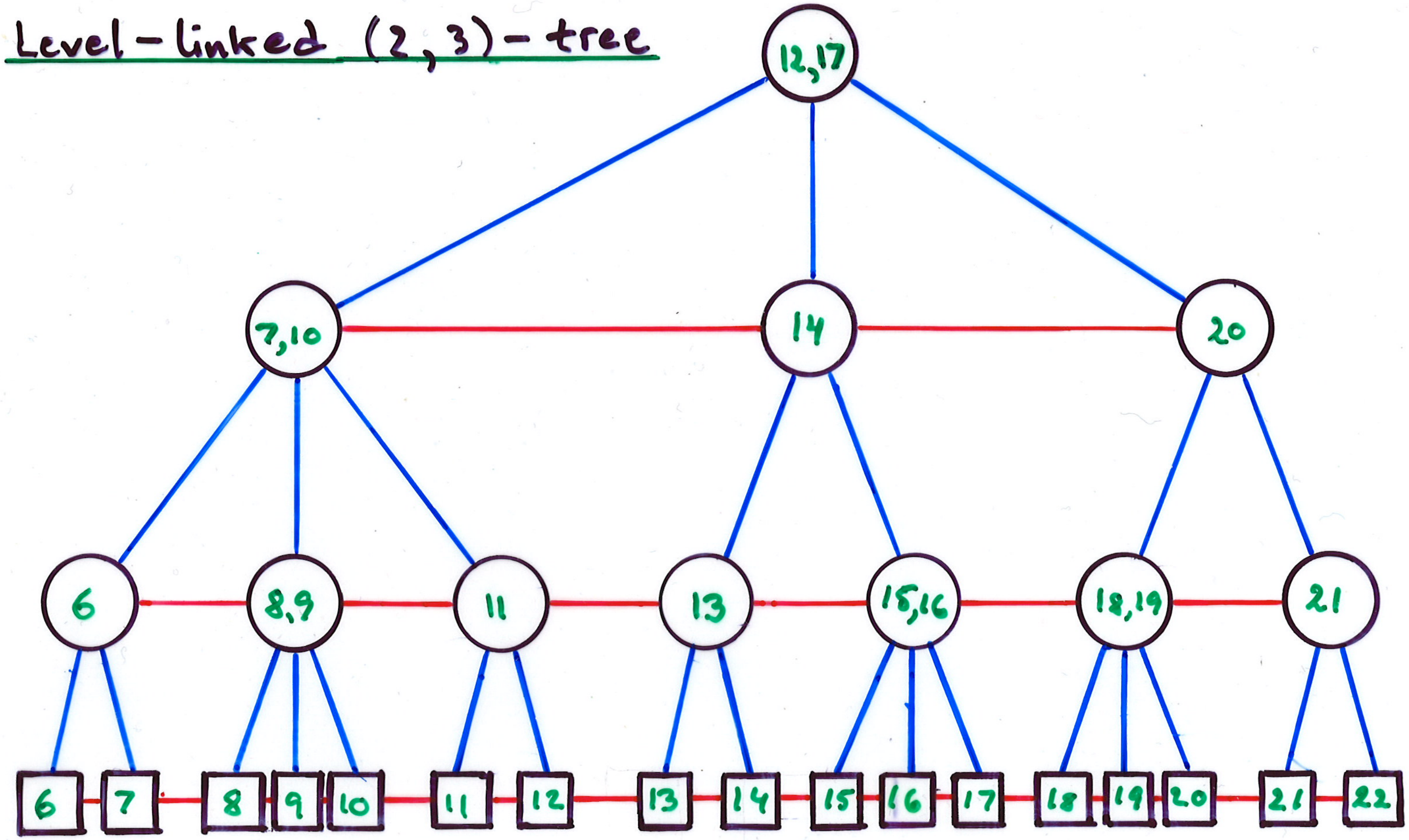
when representing the lists as finger search trees with fingers at the leftmost elements.

Theorem: Margesort is SMS-optimally adaptive, and runs in time:

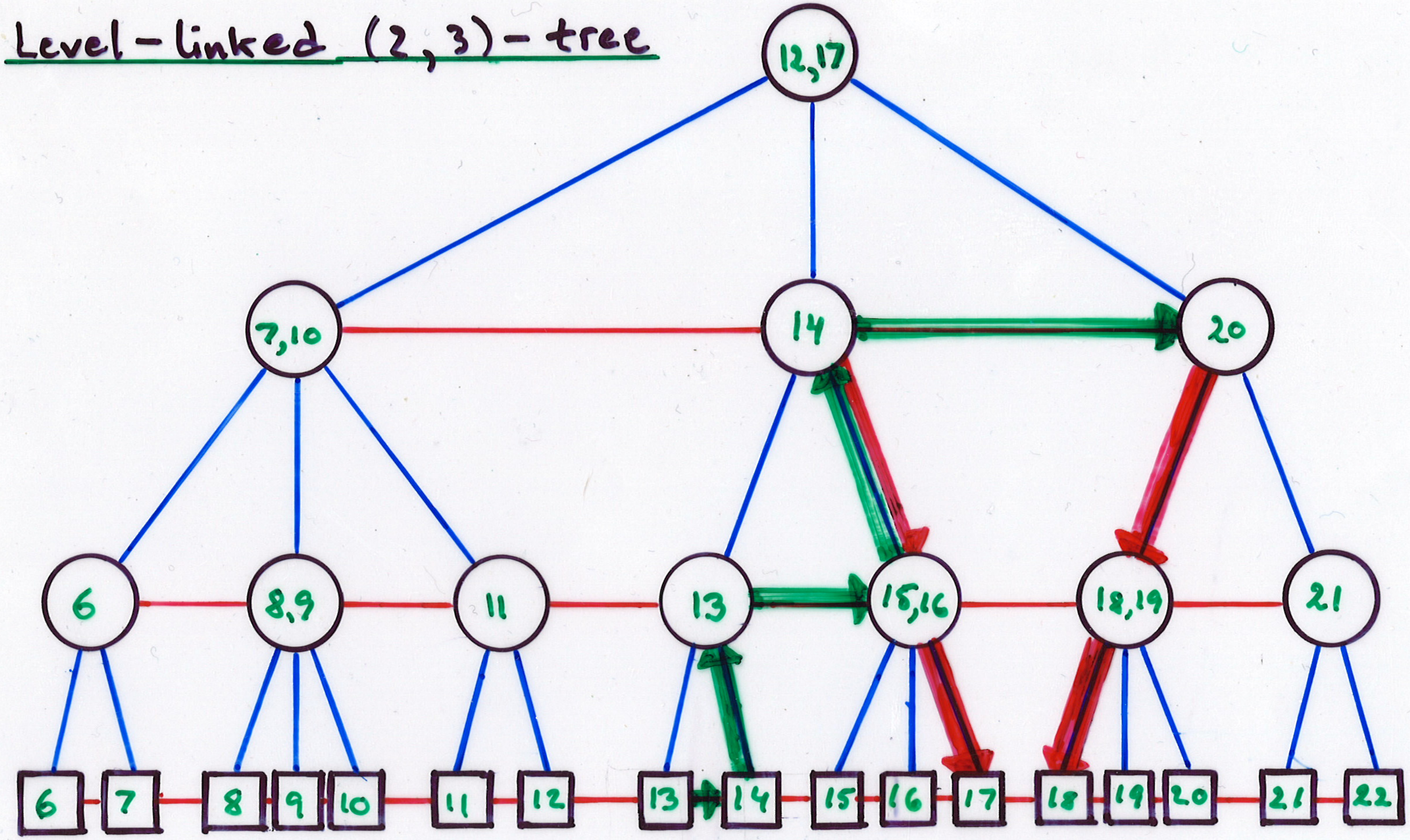
$O(|X| \log SMS(X))$. [MPW 92]


(2,3)-tree
12,17
7,10
14
20
6
8,9
11
13
15,16
18,19
21
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

Level-linked (2,3)-tree



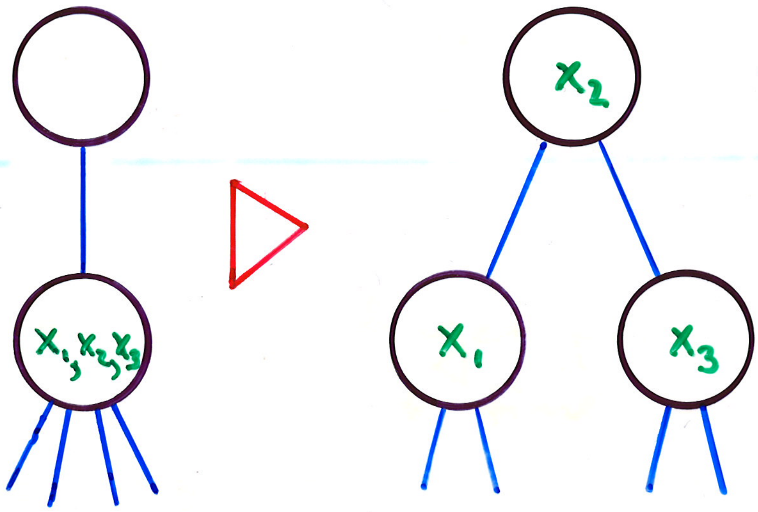
Level-linked (2,3)-tree



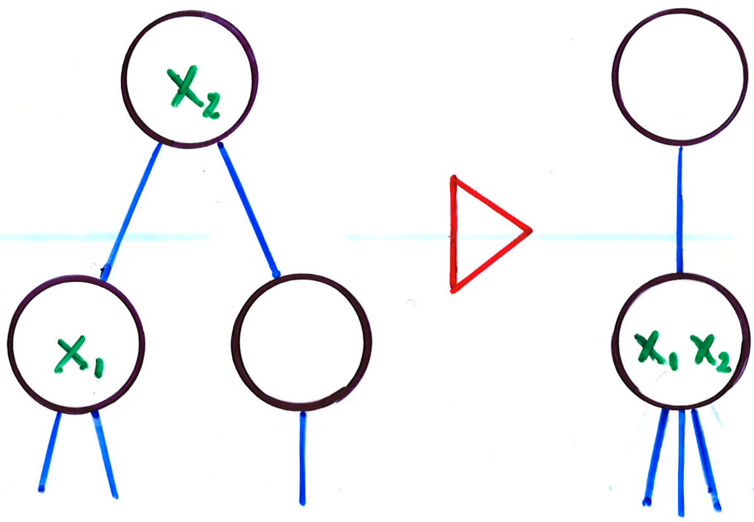
search(5, 17, 13).

Transition system for (2,3)-trees

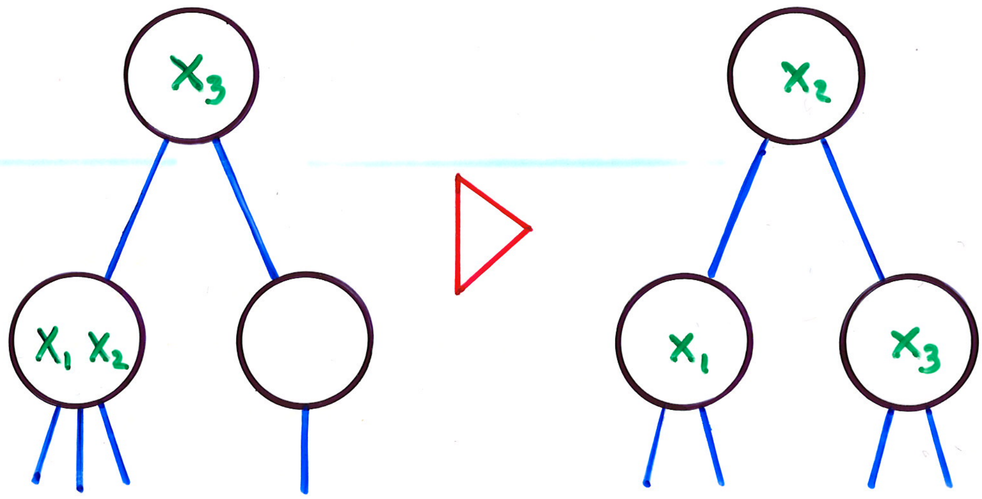
Split:



Fusion:



Sharing:



Finger Search Trees

Guibas, McCreight, Plass and Roberts 1977:

B-trees + Finger Paths + Regularity condition

Central property:
Updates on the finger paths can be done in constant time because of the regularity condition.

Brown and Tarjan 1980:

(2, 3)-trees + Level linking

Central property:
Simple.

Huddleston and Mehlhorn 1982:

(2, 4)-trees + Level linking

Central property:
Updates can be done in amortized constant time – and simple.

Finger Search Trees

	Guibas et.al.	Brown et.al.	Huddleston et.al.
Search:	$O(\log d)$	$O(\log d)$	$O(\log d)$
Insert:	$O(f + \log d)$	$O(\log n)$	$O(\log d)$ *
Delete:	$O(f + \log d)$	$O(\log n)$	$O(\log d)$ *
Create finger:	$O(\log n)$	$O(1)$	$O(1)$
Destroy finger:	$O(\log n)$	$O(1)$	$O(1)$
Join$^{\Delta}$:	$O(\log n)$	$O(\log n)$	$O(\log \min\{\lvert S_1\rvert, \lvert S_2\rvert\})$ *
Split$^{\Delta}$:	$O(\log n)$	$O(\log n)$	

* = Amortized.
Worst case is $O(\log n)$.

Δ = Join and split are not considered in the articles!

Balanced Search Trees with O(1) Update Time

Operations:

Insert, Delete, Search

Requirement:

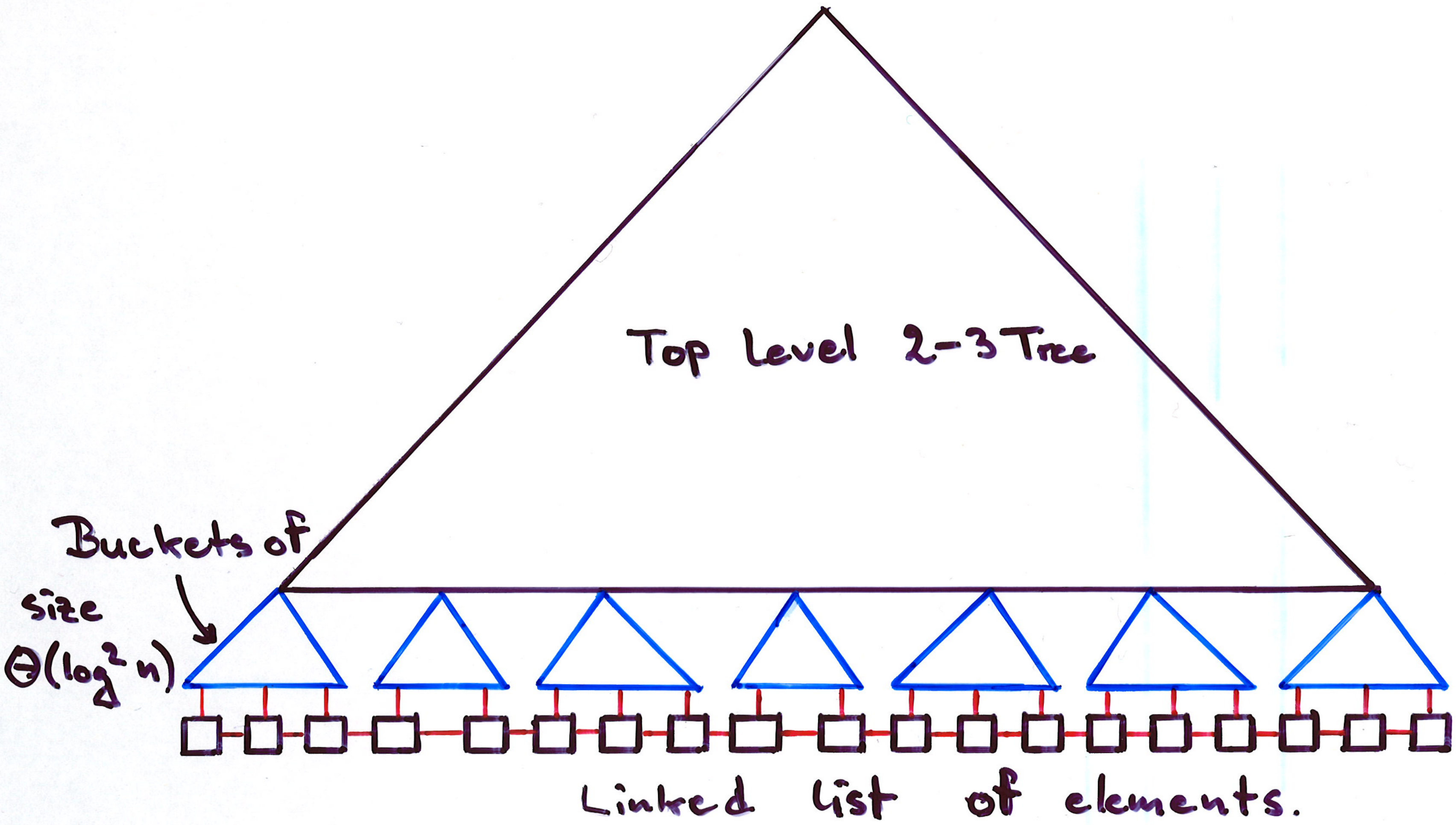
The update times for insert and delete should be worst case O(1) – once the position of the element is known.

Levcopoulos and Overmars 1988:

Search Tree + Bucketing (size $O(\log^2 n)$) + Global Rebuilding

Fleischer 1992:

Reduces the bucket size to $O(\log n)$.


Top Level 2-3 Tree
Buckets of size $\Theta(\log^2 n)$
Linked list of elements.

11

Finger Search Trees with $O(1)$ Update Time – RAM

Dietz and Raman 1990:

(2,3)-trees + Level linking
+ Bucketing (size $O(\log^2 n)$)
+ sets of size $\log n / \log\log n$ can be represented by a single word.

Join and split not supported.

A two player game related to finger search trees.

The game:

Player A: Can insert/delete an element from one of the leaves of a tree.

Player B: Can do one of the actions:

i) Split an internal node

ii) Fusion an internal node with one of its brothers.

iii) Move a number of sons from an internal node to its neighbour brother.

Theorem:

A strategy for player B exists that results in trees with internal nodes of degrees between a and b where $2 \leq a < b$ and a, b are constants.

Problem:

How to find the place to do the update in $O(1)$ time!