Finger Search

Searching in a sorted array

2	**3**	**5**	**7**	**8**	**11**	**13**	**14**	**15**	**17**	**18**	**20**	**24**	**25**	**26**	**28**	**29**	**31**	**33**	**34**

time O(log *n*)

Binary-search(13)

Finger

d

2	**3**	**5**	**7**	**8**	**11**	**13**	**14**	**15**	**17**	**18**	**20**	**24**	**25**	**26**	**28**	**29**	**31**	**33**	**34**

2^0 2^1 2^2

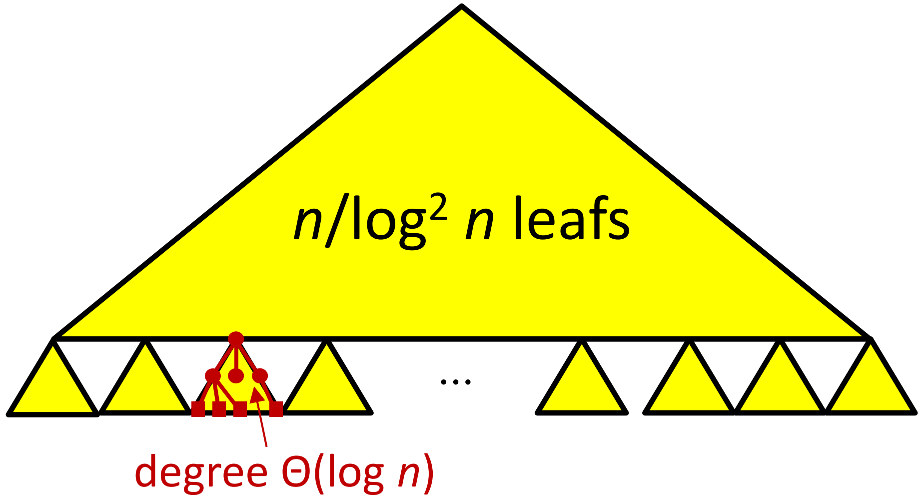
time O(log *d*)

Exponential-search(13)

Bently Yao 1976 $\sum_{i=1}^{\log^* n} \log^{(i)} x + O(\log^* n)$

O(1) Insertions

[C. Levcopoulos, M. Overmars, *A balanced search tree with* O(1) *worst-case update time*, Acta Informatica, 1988, 26(3), 269-277, 1988]



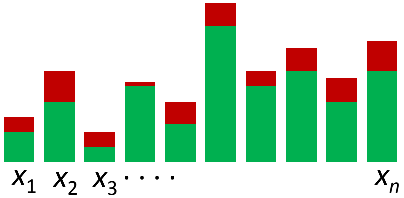
- Buckets $O(\log n)$ $\Rightarrow$ Amortized O(1) insertions (also by 2-4-trees)
- 2-level buckets $O(\log^2 n)$ size
- Incremental splitting of buckets
- Split largest bucket

$\Rightarrow$ Wost-case O(1) insertions

Zeroing Game

[P. Dietz, D. Sleator, *Two algorithms for maintaining order in a list*, Proc. 19th ACM Conf. on Theory of Computing, 365-372, 1987]

- Variables $x_1,\ldots,x_n \geq 0$ (initially $x_i = 0$)
- Players Z and A alternate to take turns
  - Z: Select j where $a_j = \max_i x_i$: $x_j := 0$
  - A: Select $a_1,\ldots,a_n \geq 0$ and $\sum_i a_i = 1$: $x_i \mathrel{+}= a_i$



Theorem $\forall i : x_i \leq H_{n-1}+1 \leq \ln n+2$

Proof

- Consider a vector $x^{(m)}$ after $m \geq n$ rounds
- $S_k \stackrel{\text{def}}{=}$ sum of k largest x_i of $x^{(m+1-k)}$
- $S_n \leq n$ (induction)
- $S_i \leq 1+ S_{i+1} \cdot i/(i+1)$
- $S_1 \leq 1+S_2/2 \leq 1+1/2+S_2/3 \leq 1+1/2+\cdots+1/(n-1)+S_n/n \leq H_{n-1}+1$

Corollary

For the halving game, Z : $x_i := x_i/2$

For the splitting game, Z : $x_i,x_{i'} := x_i/2$

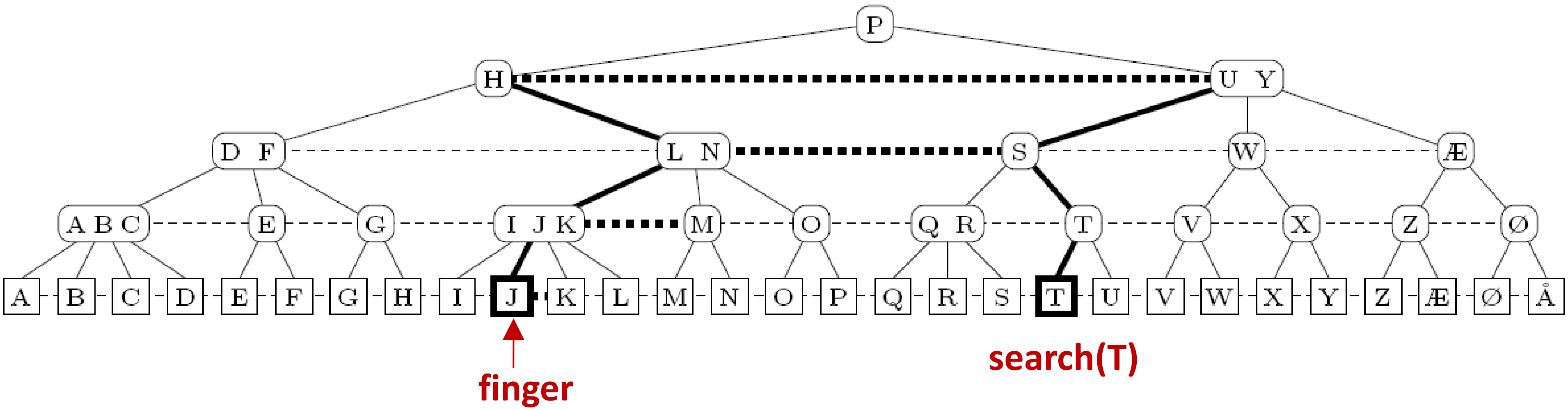
$\left.\right\}$ $\forall i : x_i \leq 2\cdot(H_{n-1}+1)$

Dynamic Finger Search

	Search	Insert/Delete
Search without fingers		
Red-black, AVL, 2-4-trees, ... Levcopolous, Overmars 1978	O(log n)	O(log n) O(1)
O(1) fixed fingers		
Guibas et al. 1977, ...	O(log ***d***)	O(1)
Each node a finger		
Level-linked (2,4)-trees	O(log ***d***)	O(log n) O(1) am.
Randomized Skip lists	O(log ***d***) exp.	O(1) exp.
Treaps	O(log ***d***) exp.	O(1) exp.
Brodal, Lagogiannis, Makris, Tsakalidis, Tsichlas 2003 Dietz, Raman 1994 (RAM)	O(log ***d***)	O(1)

Level-Linked (2,4)-trees

[S. Huddleston, K. Mehlhorn. *A new data structure for representing sorted lists*. Acta Informatica, 17:157–184, 1982]



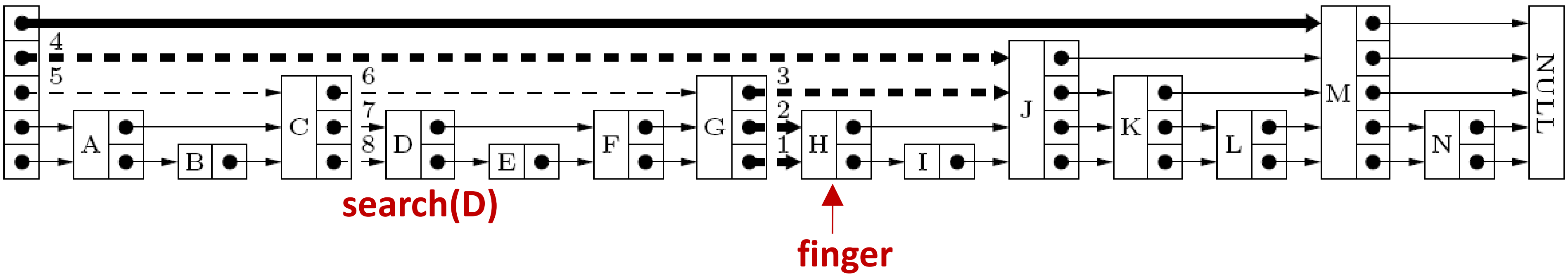
Updates Split nodes of degree >4, fusion nodes of degree <2

Search Search up **+** top-down search

Potential $\Phi = 2 \cdot \#\,\text{degree-4} + \#\,\text{degree-2}$

Randomized Skip Lists

[W. Pugh. *Skip lists: A probabilistic alternative to balanced trees. Communications of the ACM*, 33(6):668–676, 1990]

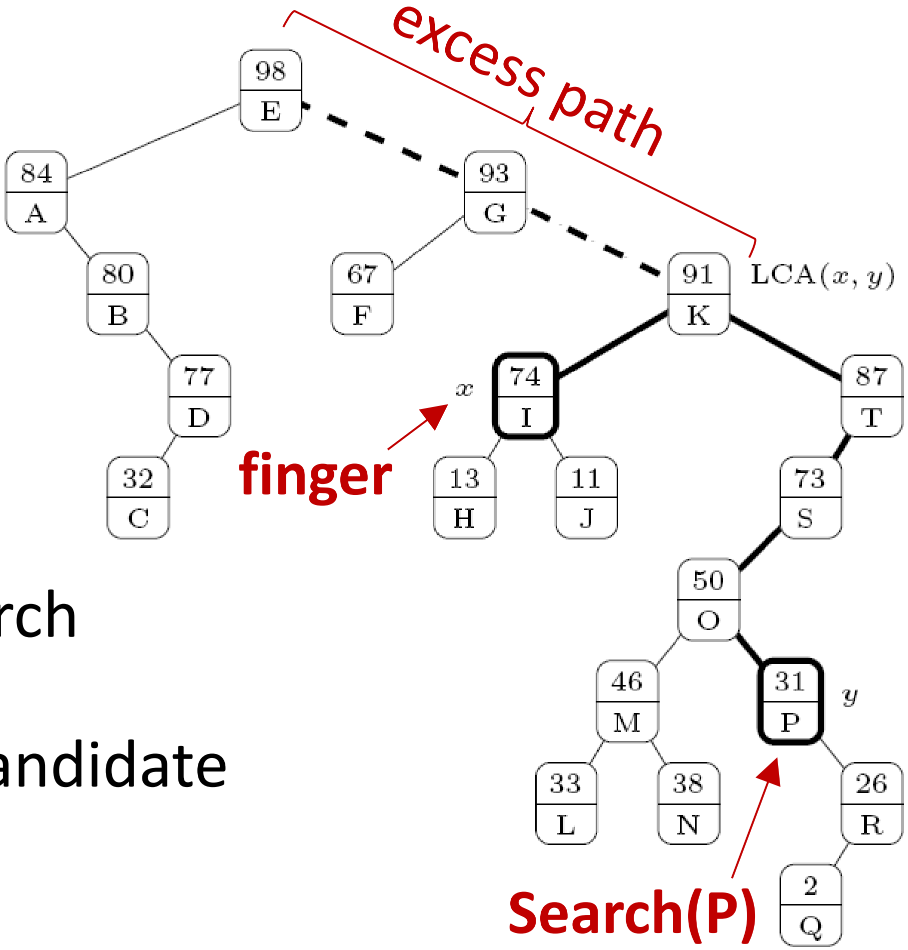


Insertion	Increase pile to next level with pr. = 1/2
Height	O(log *n*) expected with high probability
Pointer	Horizontally spans O(1) exp. piles one level below
Finger	Remember nodes on search path

Treaps – Randomized Binary Search Trees

[R. Seidel and C. R. Aragon. *Randomized search trees*. Algorithmica, 16(4/5):464–497, 1996]

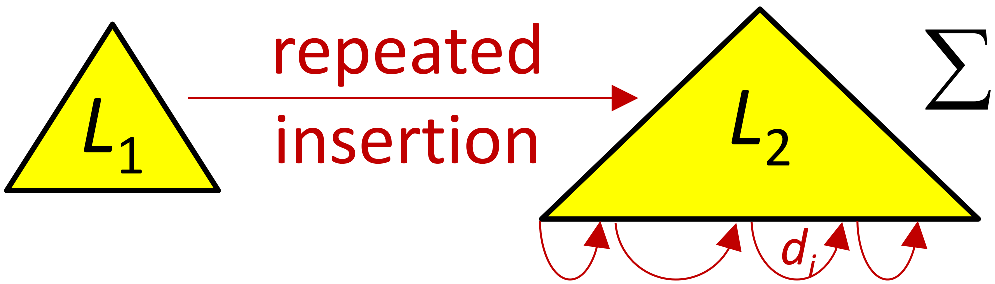
- Each element random priority
- Search tree wrt element
- Heap order wrt priority
- Height O(log *n*) expected
- Insert & deletion **rotations**
  O(1) expected time
- **Search** Go up to LCA, and search down – concurrently follow excess path to find next LCA candidate
  Search path O(log *d*) expected



Application: Binary Merging

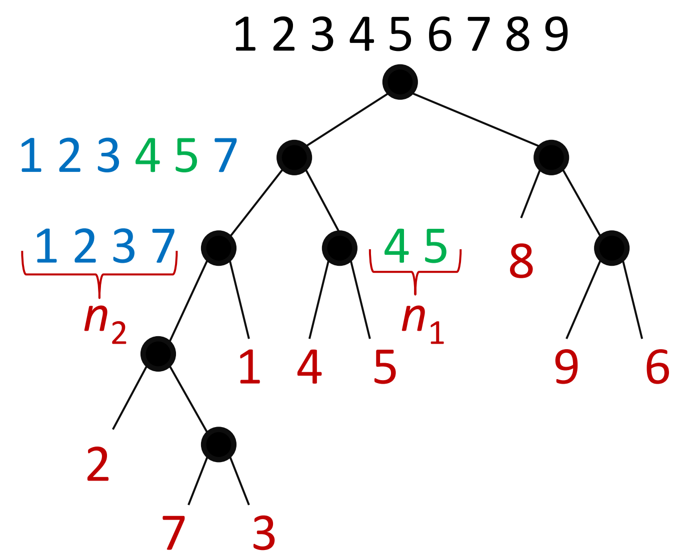
[S. Huddleston, K. Mehlhorn. *A new data structure for representing sorted lists*. Acta Informatica, 17:157–184, 1982]

- Merging sorted lists L_1 and L_2 / finger search trees



$$\sum \log(d_i) = |L_1| \log\left(\frac{|L_2|+|L_1|}{|L_1|}\right)$$

- Merging leaf lists in an **arbitrary** binary tree $O(n \cdot \log n)$



Proof Induction $O(\log n!)$

$O(\log n_1! + \log n_2! + n_1 \cdot \log((n_1+n_2)/n_1))$
$= O(\log n_1! + \log n_2! + \log \binom{n_1+n_2}{n_1})$
$= O(\log (n_1! \cdot n_2! \cdot \binom{n_1+n_2}{n_1})) = O(\log (n_1+n_2)!)$ □

Maximal Pairs with Bounded Gap

[G.S. Brodal, R.B. Lyngsø, C.N.S. Pedersen, J. Stoye. *Finding Maximal Pairs with Bounded Gap*, Journal of Discrete Algorithms, Special Issue of Matching Patterns, volume 1(1), pages 77-104, 2000]

left maximal ≠ ≠ right maximal

$$\text{ABC}\underbrace{\text{DABDBA}}_{P}\underbrace{\text{DAAD}}_{\text{gap} \in [\text{low},\text{high}]}\underbrace{\text{DABDBA}}_{P}\text{CABA}$$

$O(n \cdot \log n + k)$

- Build suffix tree (ST) & make it binary
- Create leaf lists at each node
- Right-maximal pairs = ST nodes
- Find maximal pairs = finger search at ST nodes