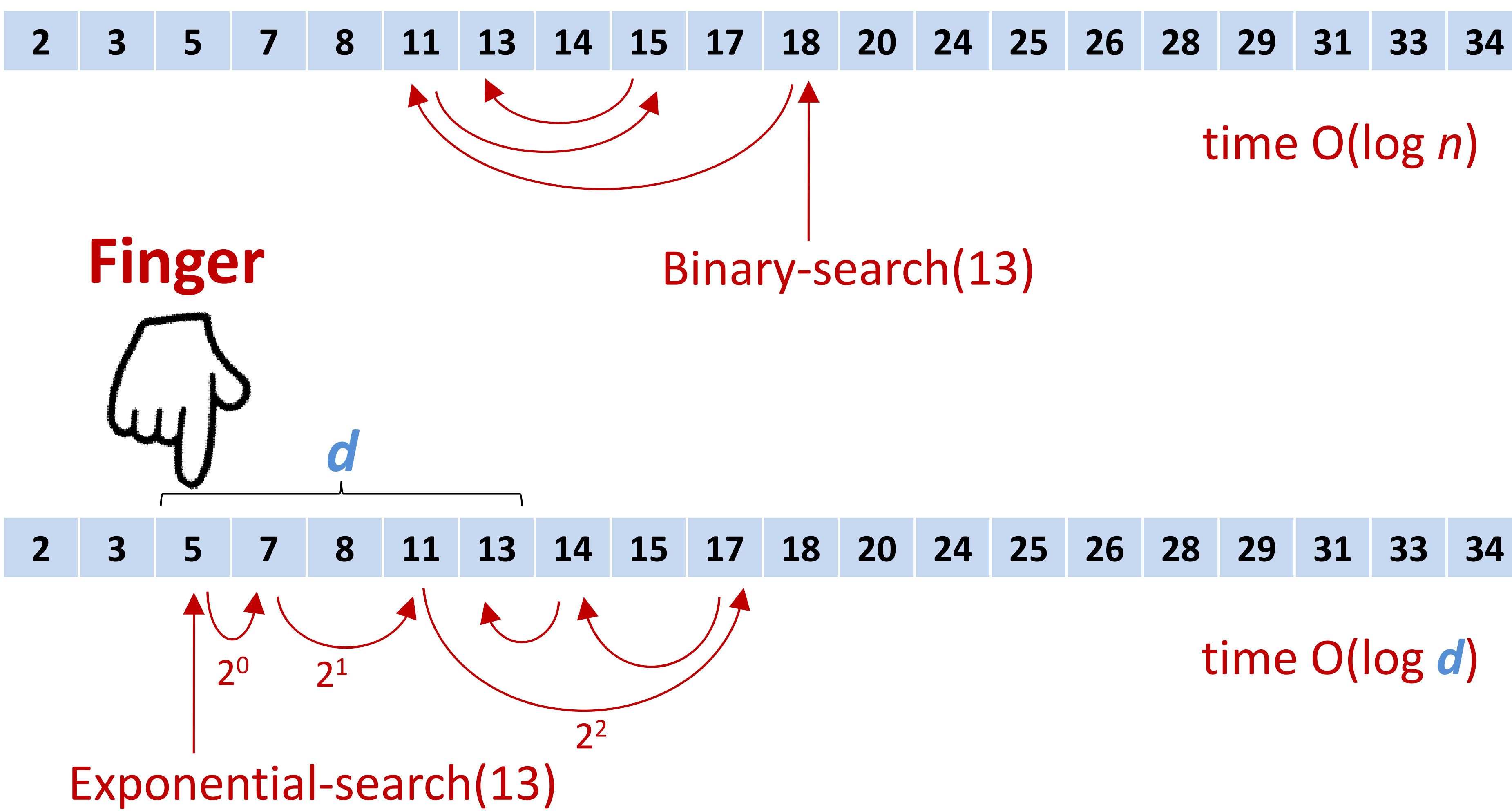
Finger Search

Searching in a sorted array

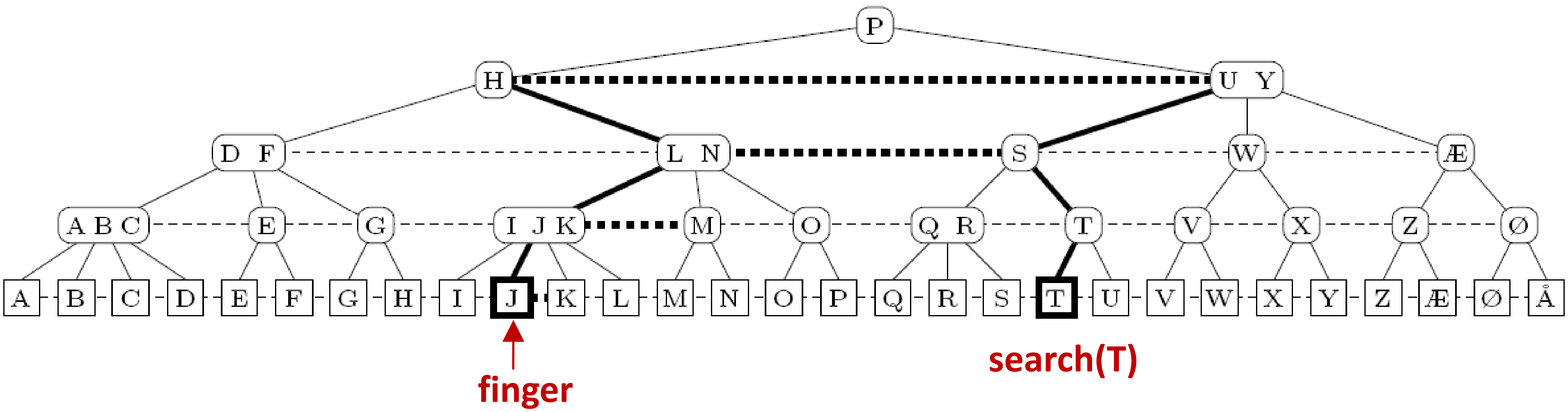


Dynamic Finger Search

	Search	Insert/Delete
No fingers		
Red-black, AVL, 2-4-trees, ...	O(log n)	O(log n)
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	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]



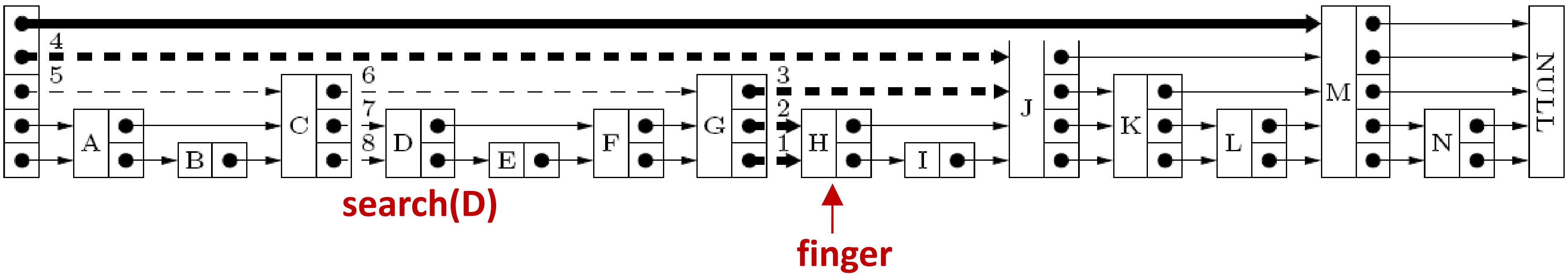
Updates Split nodes of degree >4, fusion nodes of degree <2

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

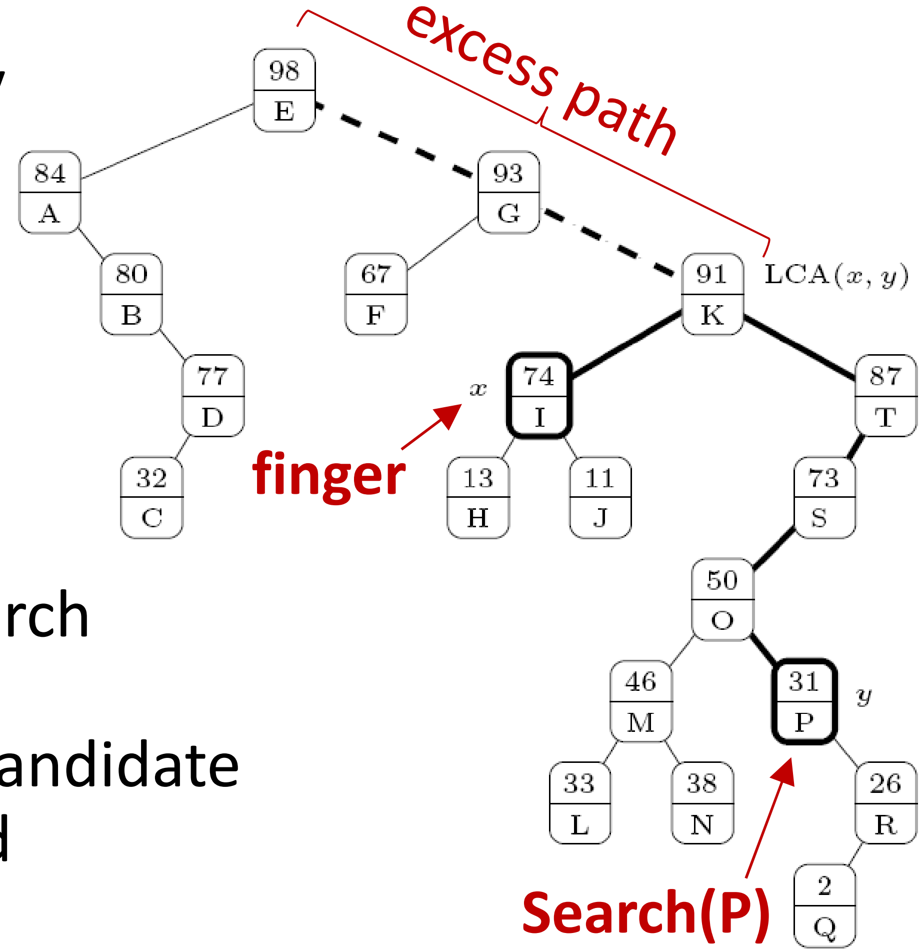
Pointer Horizontally spans O(1) exp. piles one level below

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]

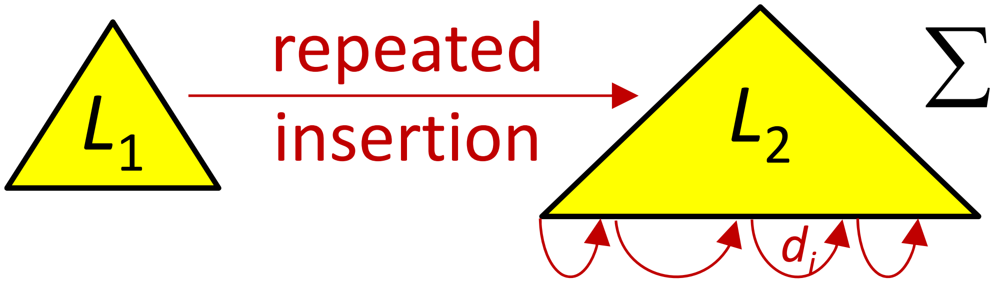
- Each element random priority
- Search tree wrt element
- Heap order wrt priority
- Height O(log *n*) expected
- Insert & deletion **rotations**
  O(1) expected time
- **Search** Go up to LCA, and search down – concurrently follow excess path to find next LCA candidate
  Search path O(log ***d***) expected



Application: Binary Merging

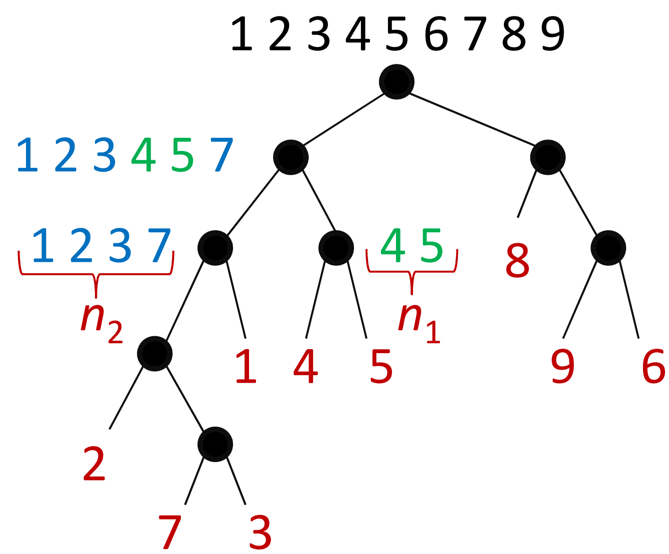
[S. Huddleston, K. Mehlhorn. *A new data structure for representing sorted lists*. Acta Informatica, 17:157–184, 1982]

- 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·log n)



Proof Induction O(log n!)

O(log n_1! + log n_2! + n_1·log ((n_1+n_2)/n_1))
= O(log n_1! + log n_2! + n_1·log $\binom{n_1+n_2}{n_1}$)
= O(log n_1! + log n_2! + (n_1+n_2)·log (n_1+n_2) - log n_1! - log n_2!) = 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

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

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

- 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