CSE221

Lecture 12:
Ordered Maps and Skip Lists

Hyungon Moon

Acknowledgment: The content of this file is based on the slides of the textbook as well as the slides provided by Prof. Won-Ki Jeong and Prof. Tsz-Chiu Au.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

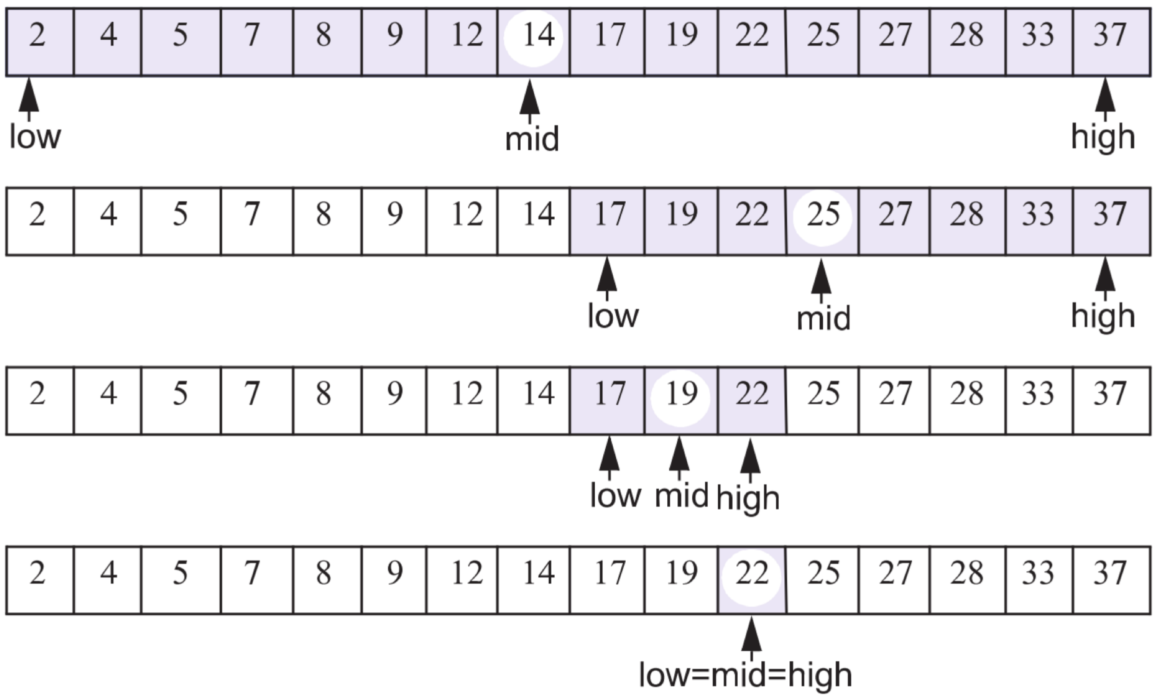
Outline

- Ordered Maps
- Skip Lists

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Search on Ordered Maps

- Entries are sorted (search table)
  - To make indexing easier, arrays are typically used



Find(K) : O(log n)

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Comparing Map Implementations

Method	*List*	*Hash Table*	*Search Table*
size, empty	$O(1)$	$O(1)$	$O(1)$
find	$O(n)$	$O(1)$ exp., $O(n)$ worst-case	$O(\log n)$
insert	$O(1)$	$O(1)$	$O(n)$
erase	$O(n)$	$O(1)$ exp., $O(n)$ worst-case	$O(n)$

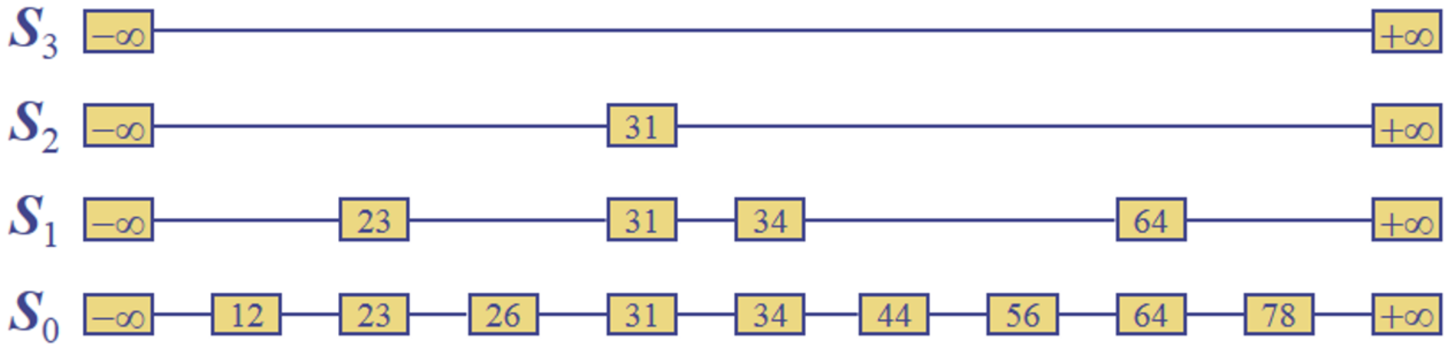
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Skip List

- A skip list for a set **S** of distinct (key, value) items is a series of lists $S_0, S_1, \ldots, S_h$ such that
  - Each list S_i contains the special keys $+\infty$ and $-\infty$
  - List S_0 contains all the keys of **S** in nondecreasing order
  - Each list is a subsequence of the previous one

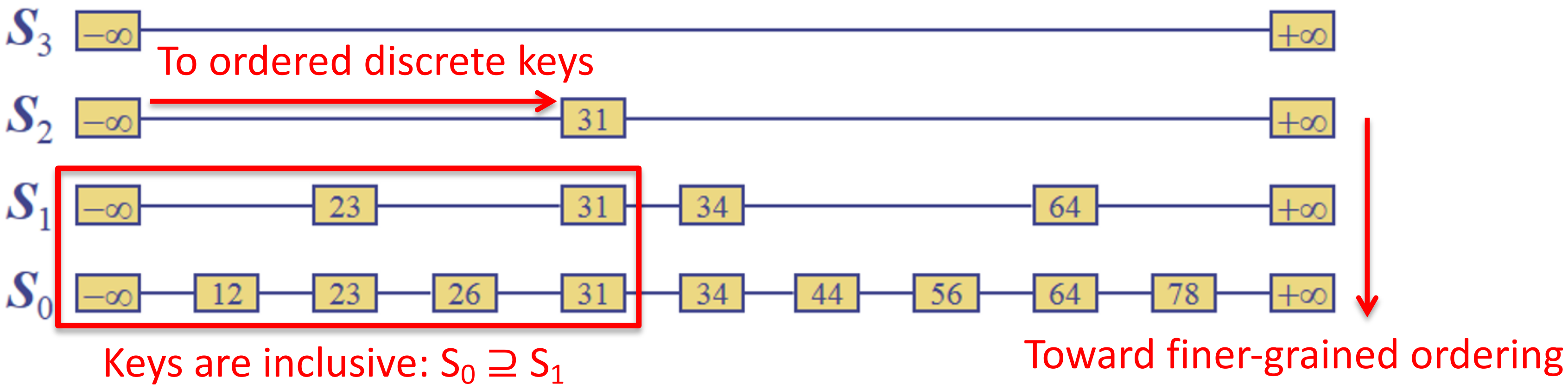
$$S_0 \supseteq S_1 \supseteq \ldots \supseteq S_h$$

  - List S_h contains only two special keys



ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

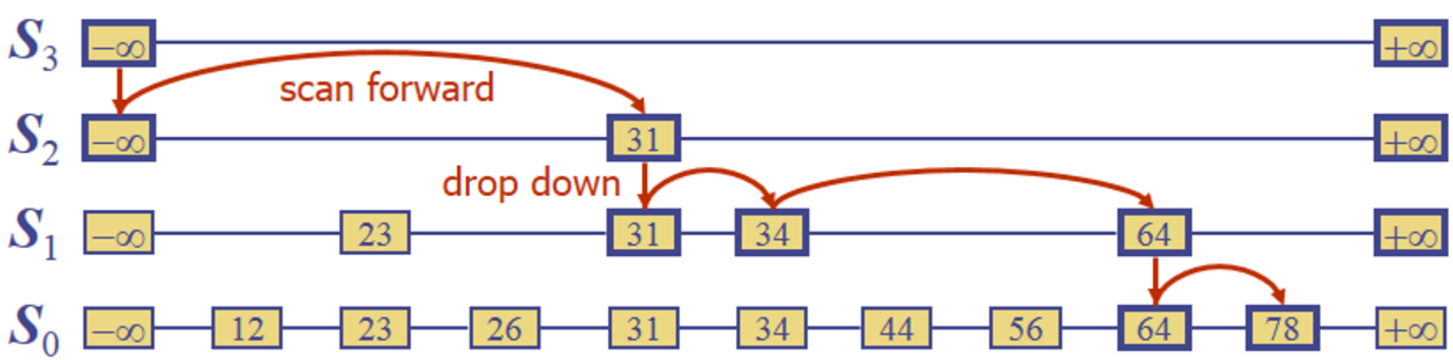
Properties



S_3
$-\infty$
$+\infty$
To ordered discrete keys
S_2
$-\infty$
31
$+\infty$
S_1
$-\infty$
23
31
34
64
$+\infty$
S_0
$-\infty$
12
23
26
31
34
44
56
64
78
$+\infty$
Keys are inclusive: $S_0 \supseteq S_1$
Toward finer-grained ordering

Search

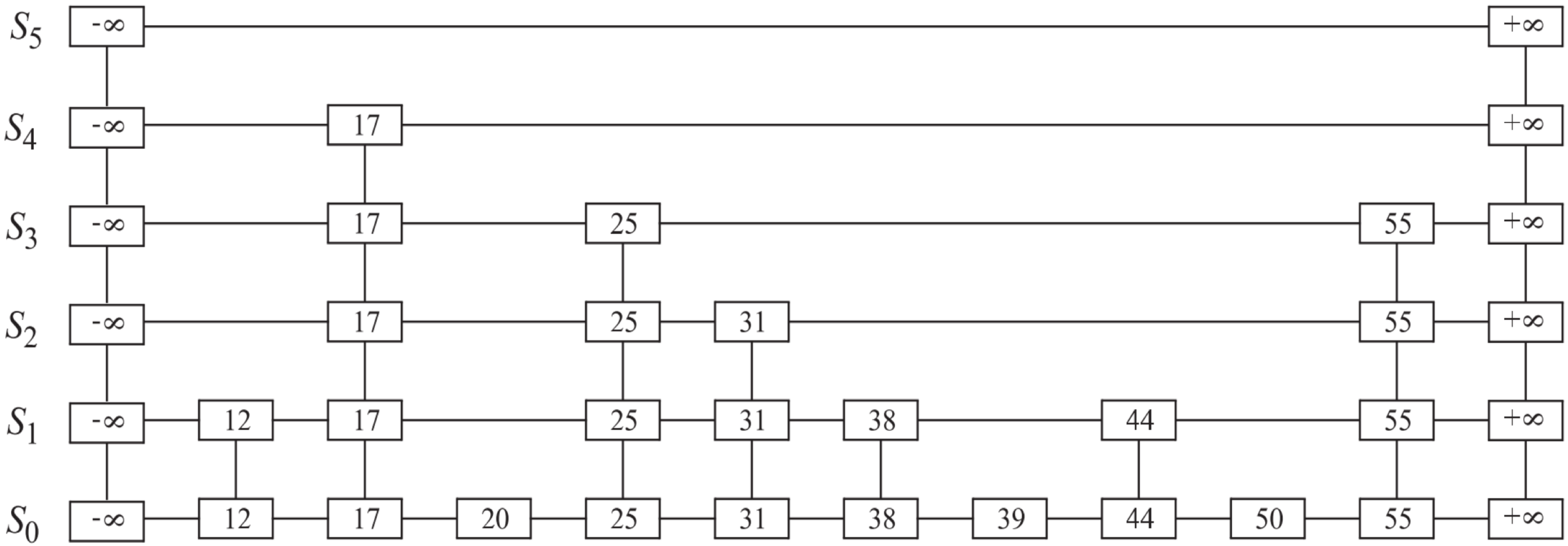
- We search for a key *x* in a skip list as follows:
  - We start at the first position of the top list
  - At the current position *p*, we compare *x* with *y* ← *key(next(p))*
    - *x* = *y*: we return *value(next(p))*
    - *x* > *y*: we "*scan forward*"
    - *x* < *y*: we "drop down"
  - If we try to drop down past the bottom list, we return *null*



Example: search for 78

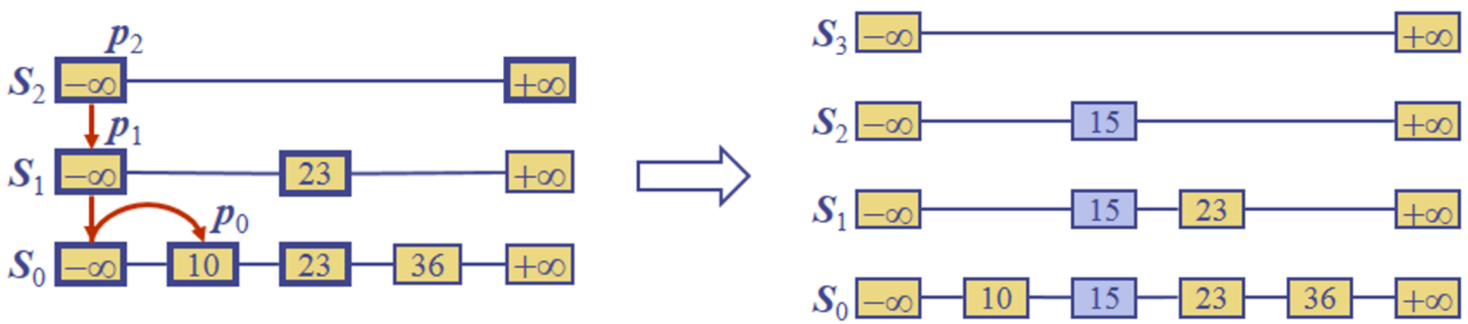
Insertion: What We Want

- About a half of items are promoted to next level
  - Purely driven by random events (e.g., coin tossing)
  - Height is approximately *log n*



Insertion

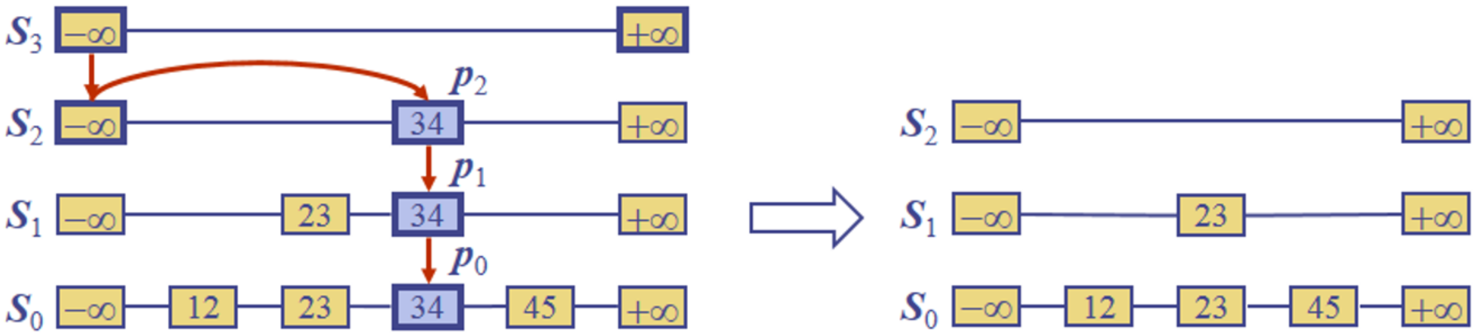
- To insert an entry *(x, o)* into a skip list:
  - We pick ***i*** through "randomized algorithm" such that its probability becomes $1/2^i$
    - *E.g.,* getting *i* consecutive heads when tossing a coin
  - If $i \geq h$, we add new lists $S_{h+1}, \ldots, S_{i+1}$, each cointaining special keys
  - We find the positions $p_0, p_1, \ldots, p_i$ of the items with largest key less than x in each list $S_0, S_1, \ldots, S_i$
  - For $j \leftarrow 0, \ldots, i$, we insert item (x, o) into list S_j after position p_j



Example: insert key 15, with *i = 2*

Deletion

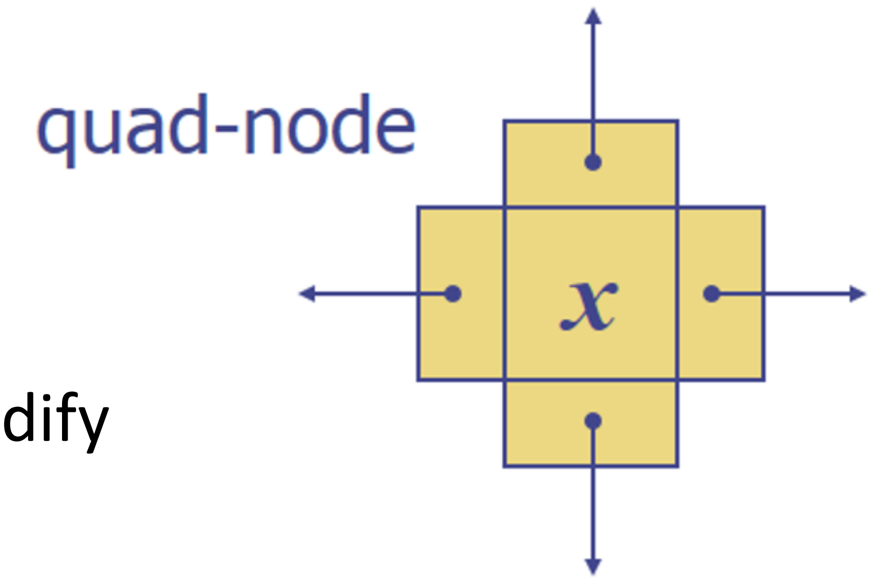
- To remove an entry with key x from a skip list:
  - We find the positions $p_0, p_1, \ldots, p_i$ of the items with key x, where position p_j is in list S_j
  - We remove positions $p_0, p_1, \ldots, p_i$ from the lists $S_0, S_1, \ldots, S_i$
  - We remove all but one list containing only special keys



Example: remove key 34

Implementation

- Uses quad-node
  - entry
  - Links to four nodes
    - (prev, next, below, above)
- Also, we define special keys (+∞ & -∞) and modify the key comparator to handle them



Space Usage

- Two probability facts:
  1) Getting *i* consecutive heads when tossing a coin: $1/2^i$
  2) If each of *n* entries is present in a set with probability *p*, the expected size of the set is *np*
- Consider a skip list with *n* entries
  - By 1) and 2), the expected size of list S_i is $n/2^i$
- The number of nodes used by the skip list is

$$\sum_{i=0}^{h} \frac{n}{2^i} = n \sum_{i=0}^{h} \frac{1}{2^i} < 2n$$

Expected space usage: O(n)

Search and Update Times

- The search time in a skip list is proportional to
  1) The number of drop-down steps, plus
     - Bounded by the height: known as *O(log n)* with high prob.
  2) The number of scan-forward steps

- To analyze 2), we use this probability fact:
  - The expected number of coin tosses required to get tails is 2

Search and Update Times

- Analyzing the scan-forward steps:
  - A scan-forward step is associated with a former coin toss that gave tails
  - In each list, the expected number of scan-forward steps is 2
  - The expected number of scan-forward steps is O(log n)
- Conclusion on search time:
  - By 1) plus 2), a search in a skip list takes O(log n)
  - Insertion and deletion give similar results

Comparison

Method	List	Hash Table	Ordered List	Skip List
size, empty	O(1)	O(1)	O(1)	O(1)
find	O(n)	O(1)/O(n)	O(log n)	O(log n)/O(n)
insert	O(1)	O(1)	O(n)	O(log n)/O(n)
erase	O(n)	O(1)/O(n)	O(n)	O(log n)/O(n)

Expected/Worst

Questions?