CS11921 - **Spring** 2026

Algorithm Design & Analysis

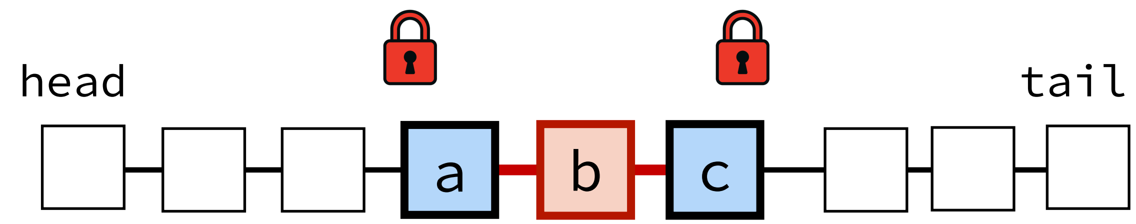
Skip Lists

Ibrahim Albluwi

Motivation

Linked Lists

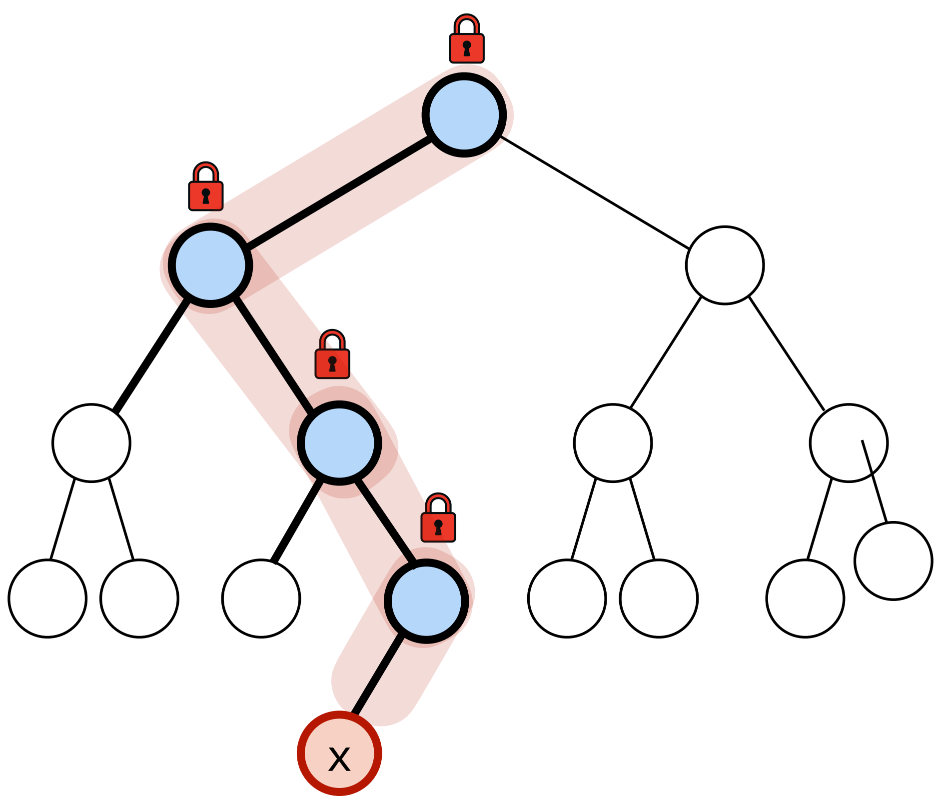
- **–** Getting to a node requires going through all the nodes before it.
- **+** Insertion and deletion make local changes (good in multithreaded applications).



deleting *b* requires **locking** *a* and *c* only

Balanced Binary Search Trees

- **+** Getting to a node is quick ($O(\log n)$ for a tree of n nodes).
- **–** Insertion and deletion can require rotations along the search path from the root to the node (bad for multithreaded applications).

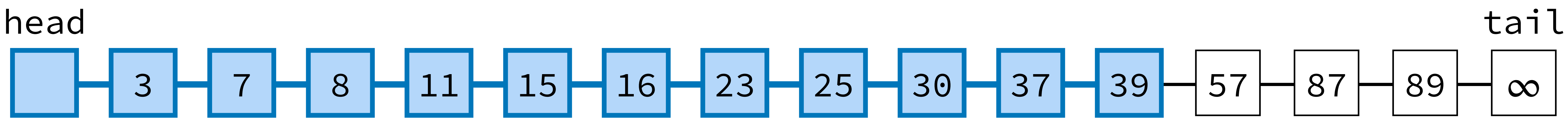


Deleting x requires **locking** the whole search path

> **Goal.** Design an ***efficient*** data structure that does not require ***locking*** large portions of it when performing insertion and deletion.

Skip Lists

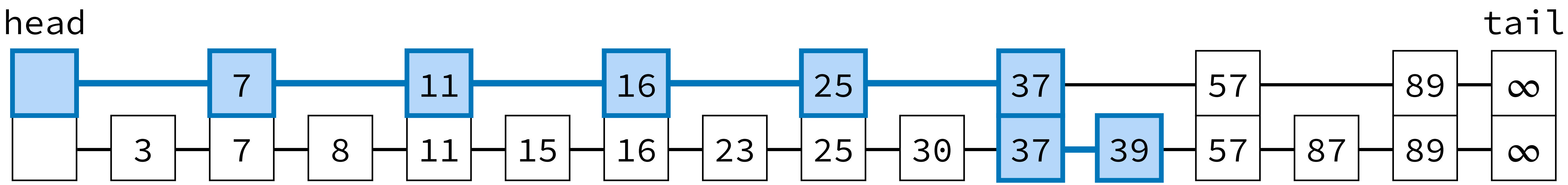
Main Idea. Modify linked lists to allow *skipping* over nodes.



Sorted Linked List. No skipping possible. To get to 39, we need to see **11** elements.

Skip Lists

Main Idea. Modify linked lists to allow *skipping* over nodes.

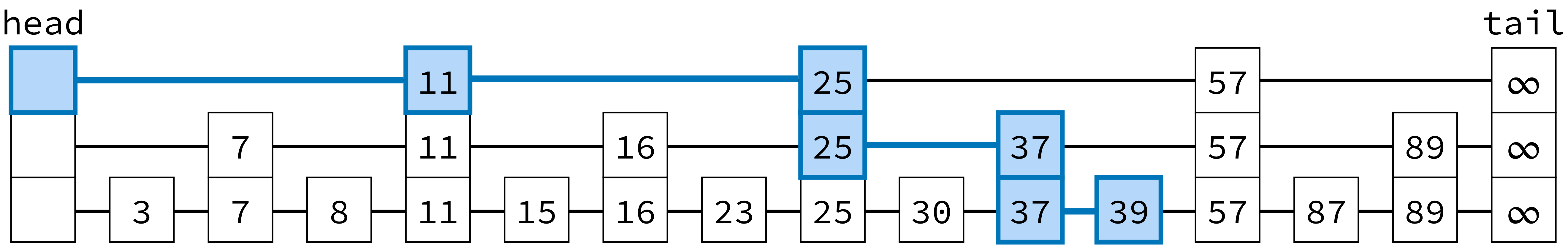


Duplicate and link every 2nd node.

Total number of seen elements reduced to **6**.

Skip Lists

Main Idea. Modify linked lists to allow *skipping* over nodes.

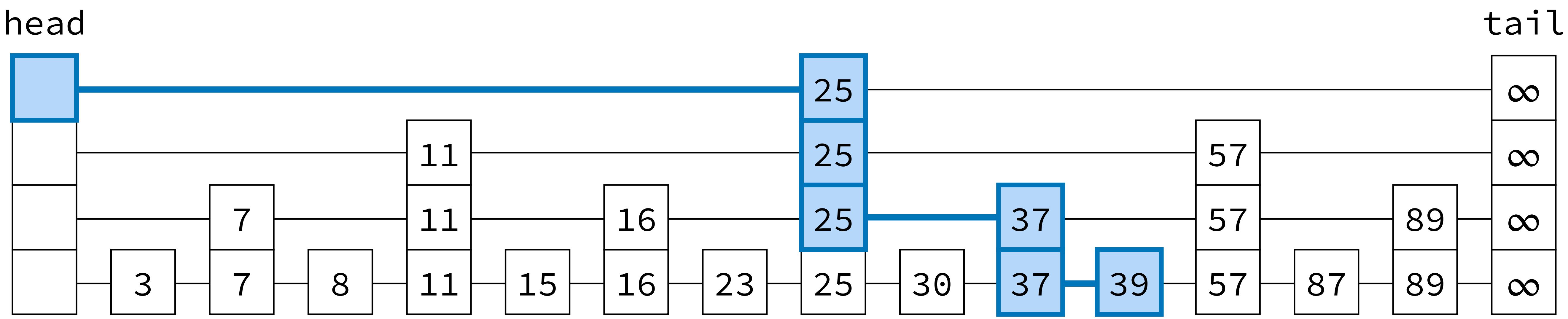


Duplicate and link every 4th node.

Total number of seen elements reduced to **4**.

Skip Lists

Main Idea. Modify linked lists to allow *skipping* over nodes.

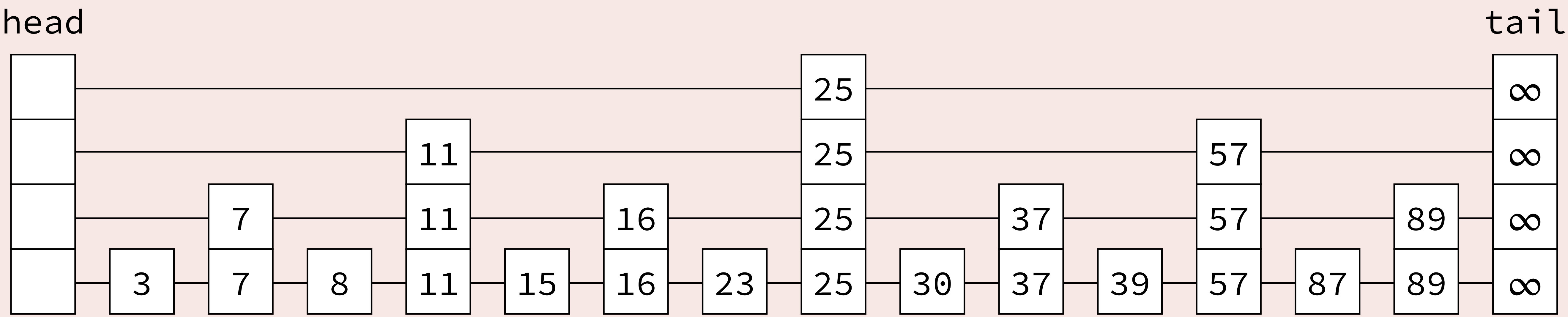


Duplicate and link every 8th node.

Total number of seen elements reduced to **3**.

Analysis

Question 1. How many comparisons are needed (in the worst case) to search for a key K in a *skip list* of size n structured like the one below?



Choose the *best* answer.

A. $O(1)$

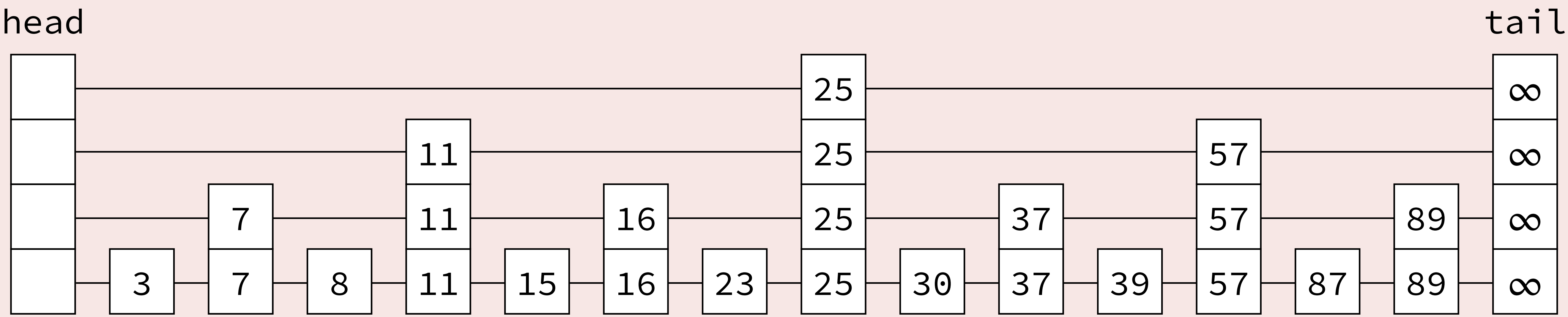
B. $O(\log n)$

C. $O(n)$

D. $O(n \log n)$

Analysis

Question 1. How many comparisons are needed (in the worst case) to search for a key K in a *skip list* of size n structured like the one below?



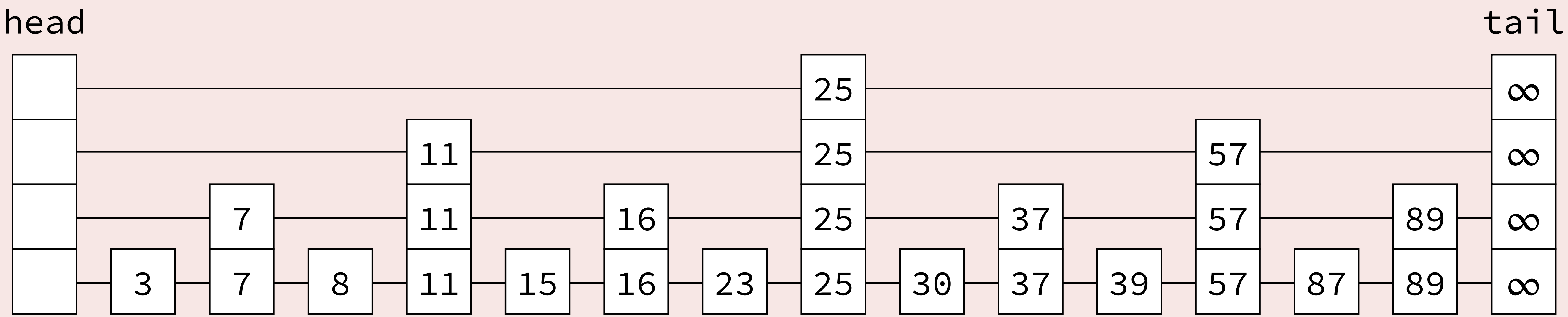
Choose the *best* answer.

- **A.** $O(1)$
- 😁 $O(\log n)$
- **C.** $O(n)$
- **D.** $O(n \log n)$

Explanation. As in a perfect binary search tree, in every level, we perform a comparison and eliminate half of the remaining nodes in the level below.

Analysis

Question 2. What is the total number of nodes in such a skip list?



Choose the *best* answer.

A. $O(n)$

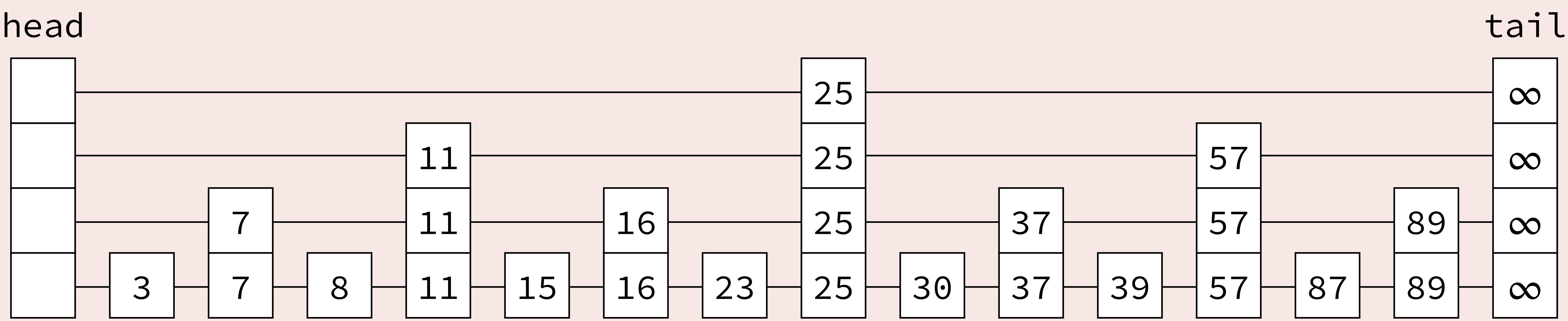
B. $O(n \log n)$

C. $O(n \log^2 n)$

D. $O(n^2 \log n)$

Analysis

Question 2. What is the total number of nodes in such a skip list?



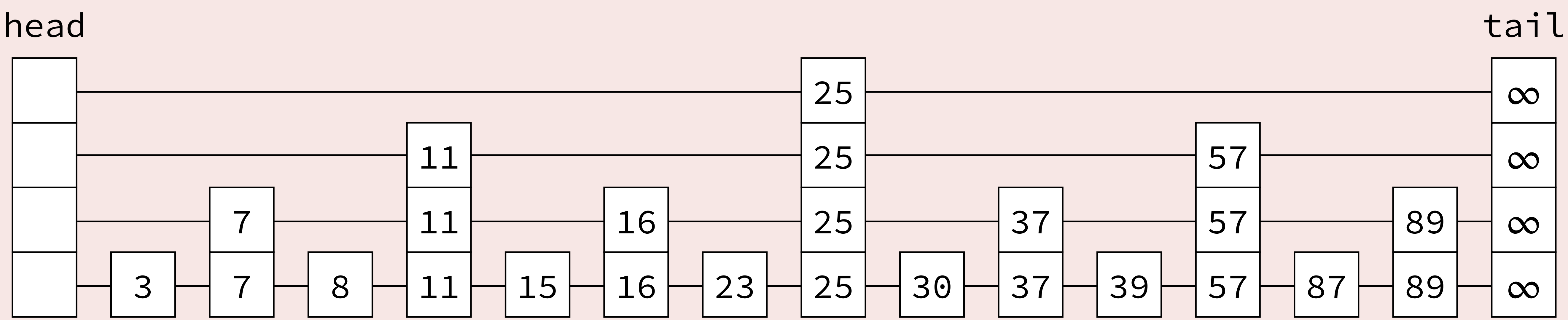
Choose the *best* answer.

- $O(n)$
- **B.** $O(n \log n)$
- **C.** $O(n \log^2 n)$
- **D.** $O(n^2 \log n)$

Explanation.

$$n + \frac{1}{2}n + \frac{1}{4}n + \ldots + 4 + 2 + 1$$
$$= n(1 + \frac{1}{2} + \frac{1}{4} + \ldots + \frac{4}{n} + \frac{2}{n} + \frac{1}{n})$$
$$= O(n)$$

Analysis



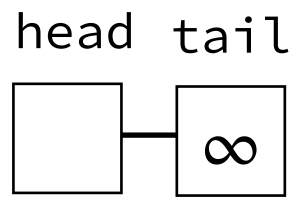
🤔 How can we construct such a skip list?

A Randomized Solution

Main Idea. Insert normally at Level 0. **Flip a coin**, and duplicate the node to the level above if the coin turns **heads**. Repeat until the coin turns **tails**.

Example. Insert `1, 5, 3, 7`.

Empty List

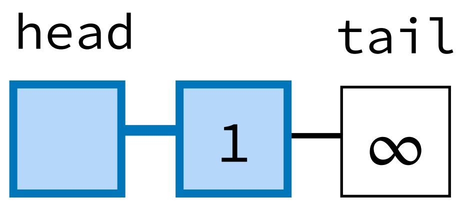


A Randomized Solution

Main Idea. Insert normally at Level 0. **Flip a coin**, and duplicate the node to the level above if the coin turns **heads**. Repeat until the coin turns **tails**.

Example. Insert `1, 5, 3, 7`.

insert 1 at $L0$

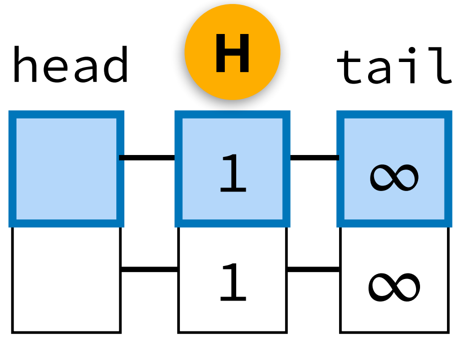


A Randomized Solution

Main Idea. Insert normally at Level 0. **Flip a coin**, and duplicate the node to the level above if the coin turns **heads**. Repeat until the coin turns **tails**.

Example. Insert 1, 5, 3, 7.

coin flip = Heads
duplicate 1 to $L1$

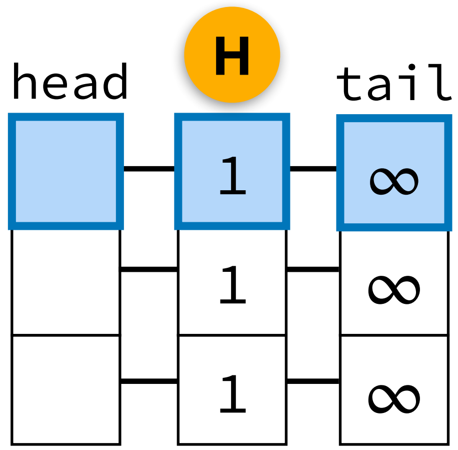


A Randomized Solution

Main Idea. Insert normally at Level 0. **Flip a coin**, and duplicate the node to the level above if the coin turns **heads**. Repeat until the coin turns **tails**.

Example. Insert `1, 5, 3, 7`.

coin flip = Heads
duplicate 1 to *L*2

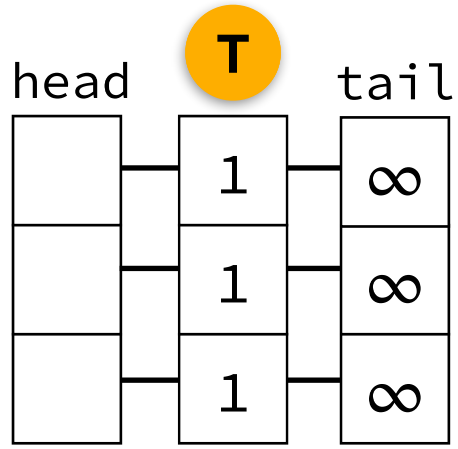


A Randomized Solution

Main Idea. Insert normally at Level 0. **Flip a coin**, and duplicate the node to the level above if the coin turns **heads**. Repeat until the coin turns **tails**.

Example. Insert `1, 5, 3, 7`.

coin flip = Tails
Stop

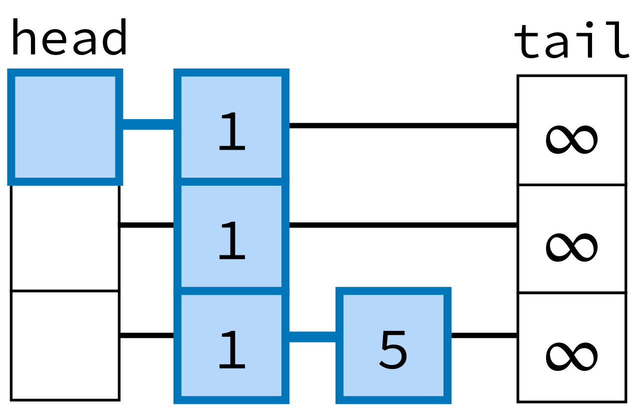


A Randomized Solution

Main Idea. Insert normally at Level 0. **Flip a coin**, and duplicate the node to the level above if the coin turns **heads**. Repeat until the coin turns **tails**.

Example. Insert `1, 5, 3, 7`.

insert 5 at $L0$

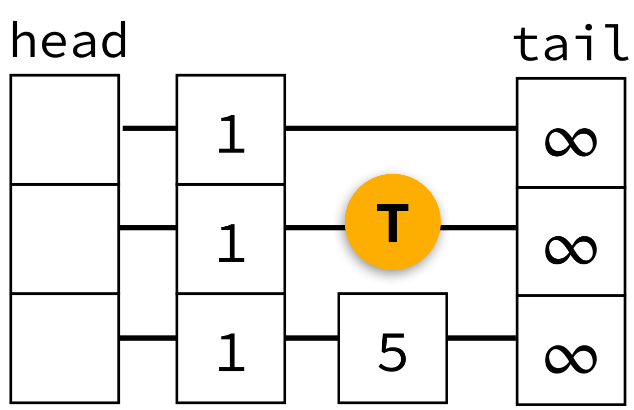


A Randomized Solution

Main Idea. Insert normally at Level 0. **Flip a coin**, and duplicate the node to the level above if the coin turns **heads**. Repeat until the coin turns **tails**.

Example. Insert `1, 5, 3, 7`.

coin flip = Tails
Stop

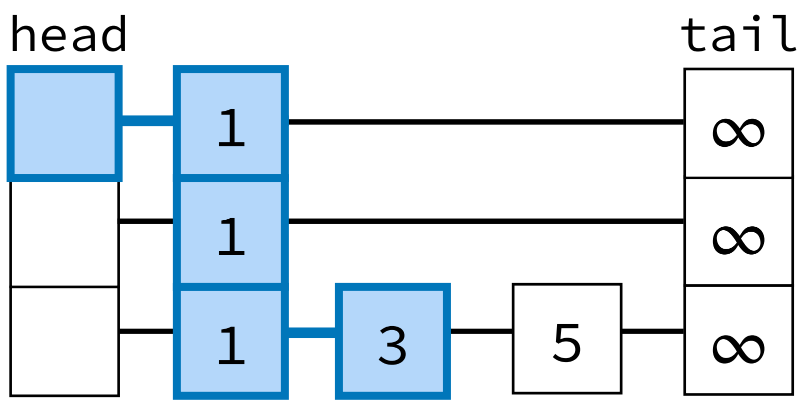


A Randomized Solution

Main Idea. Insert normally at Level 0. **Flip a coin**, and duplicate the node to the level above if the coin turns **heads**. Repeat until the coin turns **tails**.

Example. Insert `1, 5, 3, 7`.

insert 3 at $L0$

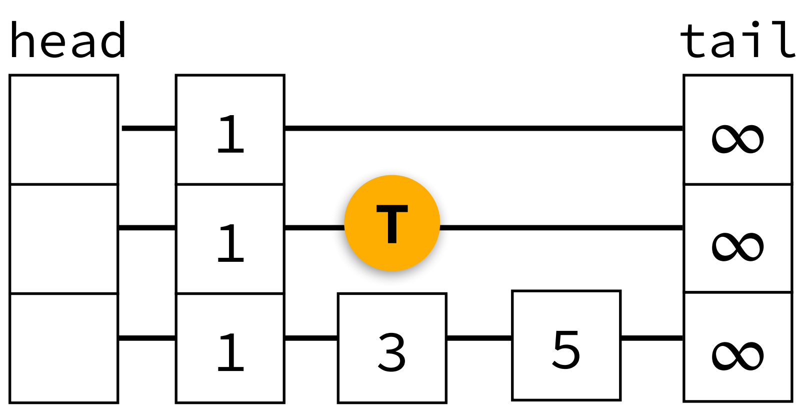


A Randomized Solution

Main Idea. Insert normally at Level 0. **Flip a coin**, and duplicate the node to the level above if the coin turns **heads**. Repeat until the coin turns **tails**.

Example. Insert `1, 5, 3, 7`.

coin flip = Tails
Stop

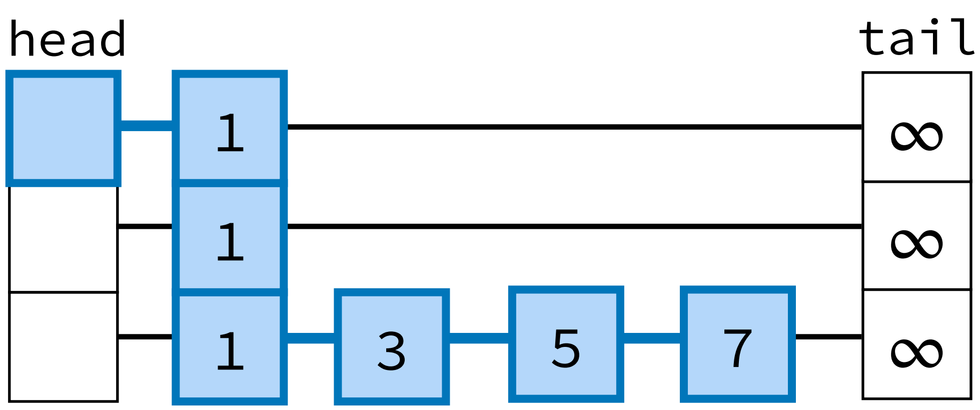


A Randomized Solution

Main Idea. Insert normally at Level 0. **Flip a coin**, and duplicate the node to the level above if the coin turns **heads**. Repeat until the coin turns **tails**.

Example. Insert `1, 5, 3, 7.`

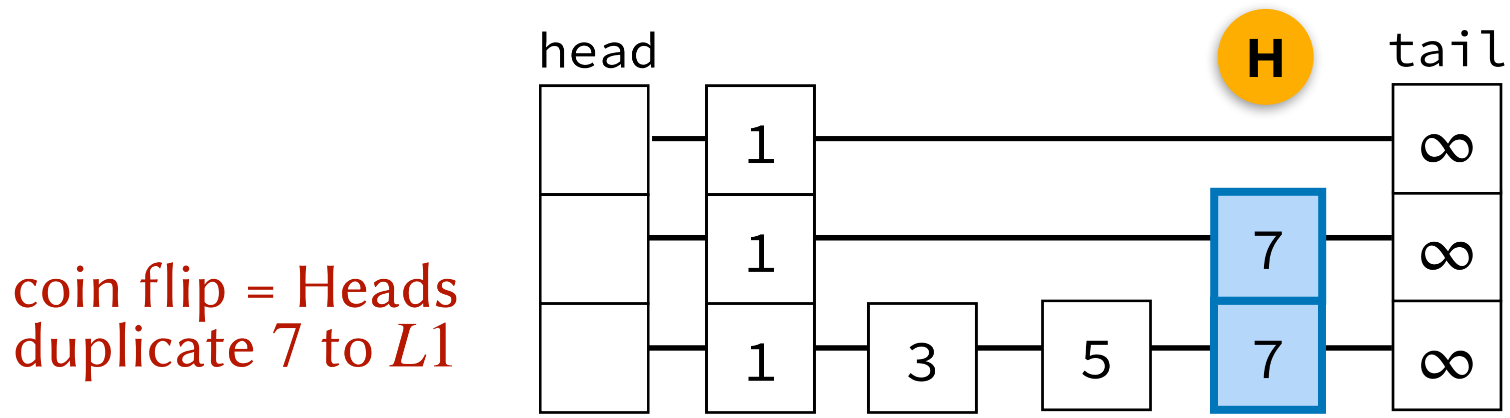
insert 7 at $L0$



A Randomized Solution

Main Idea. Insert normally at Level 0. **Flip a coin**, and duplicate the node to the level above if the coin turns **heads**. Repeat until the coin turns **tails**.

Example. Insert `1, 5, 3, 7`.

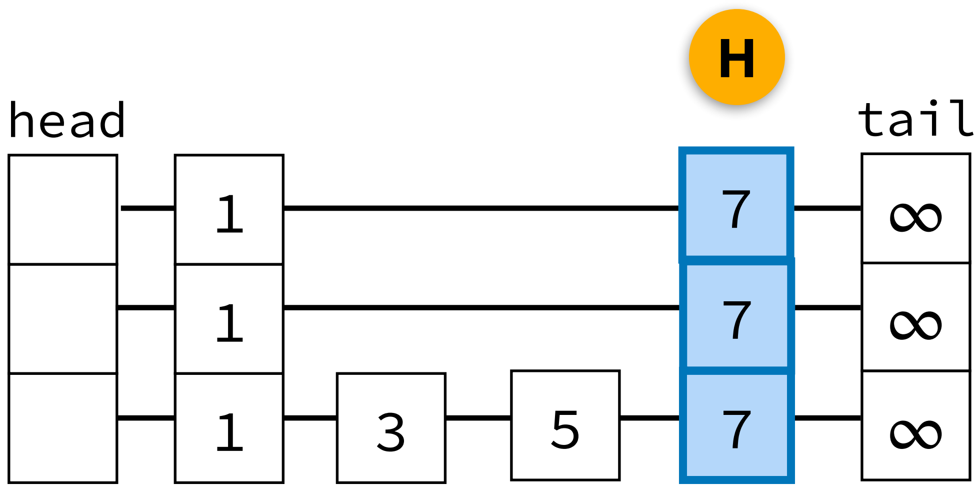


A Randomized Solution

Main Idea. Insert normally at Level 0. **Flip a coin**, and duplicate the node to the level above if the coin turns **heads**. Repeat until the coin turns **tails**.

Example. Insert `1, 5, 3, 7.`

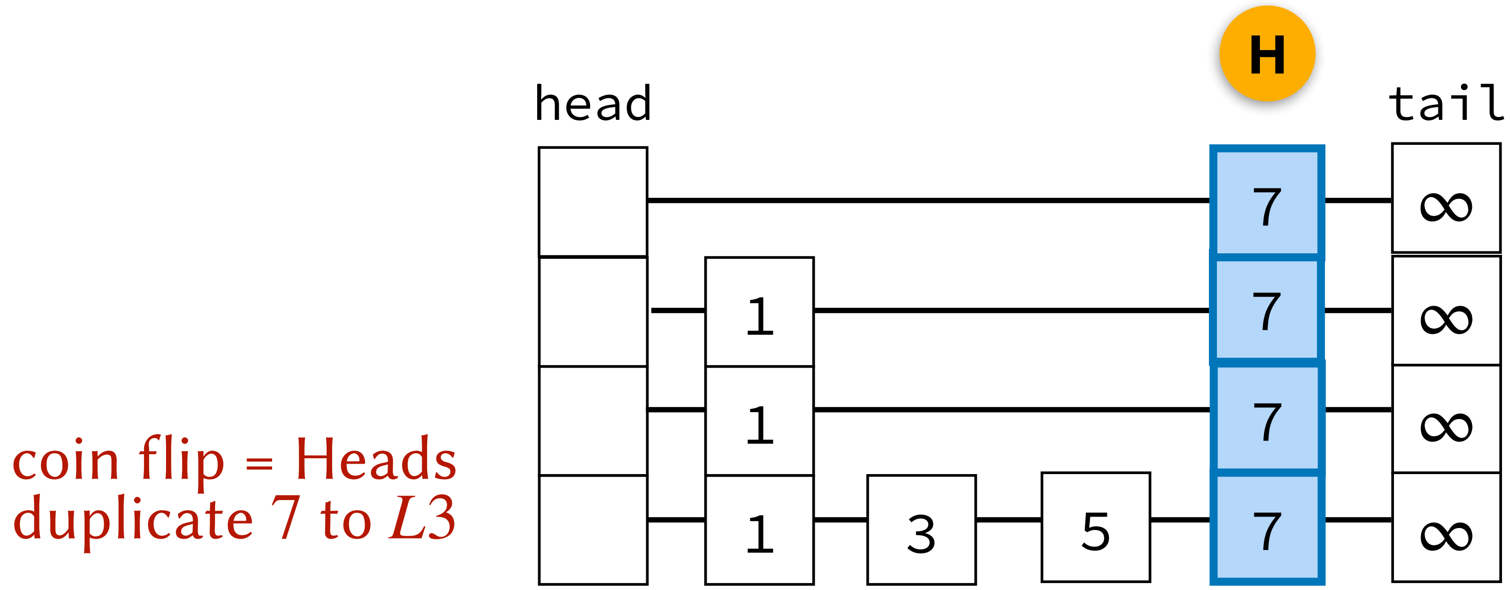
coin flip = Heads
duplicate 7 to *L2*



A Randomized Solution

Main Idea. Insert normally at Level 0. **Flip a coin**, and duplicate the node to the level above if the coin turns **heads**. Repeat until the coin turns **tails**.

Example. Insert `1, 5, 3, 7.`

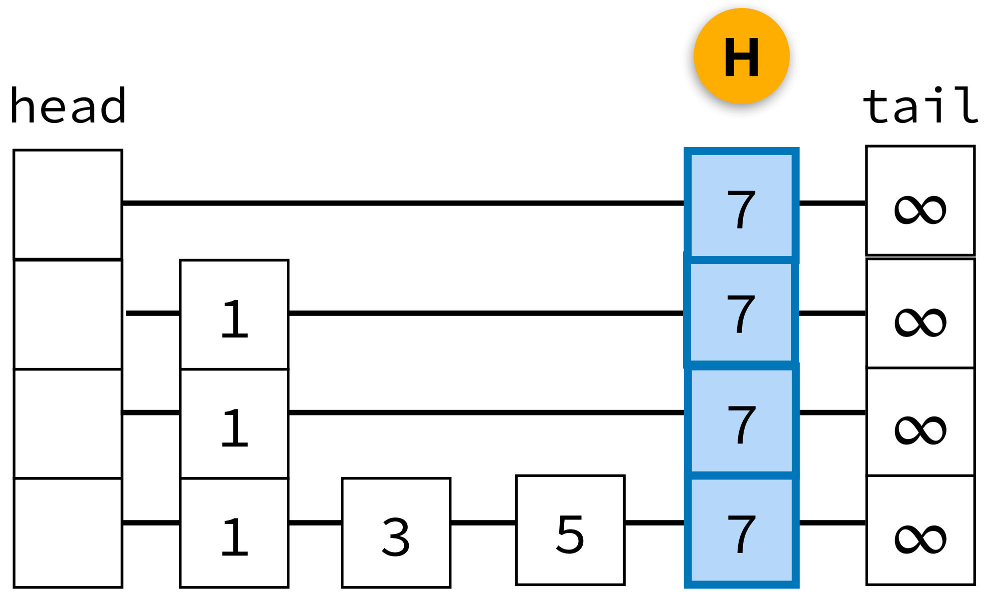


A Randomized Solution

Main Idea. Insert normally at Level 0. **Flip a coin**, and duplicate the node to the level above if the coin turns **heads**. Repeat until the coin turns **tails**.

Example. Insert `1, 5, 3, 7.`

coin flip = Heads
duplicate 7 to $L3$



🤔 **What is the expected number of levels?**

🤔 **What is the expected amount of used memory?**

🤔 **What is the expected search time?**

Number of Levels

Theorem. The probability that the number of levels in a randomized skip list with n nodes exceeds $c \log n$ is less than or equal to:

$$\frac{1}{n^{c-1}}$$

Proof. For any inserted element x,

$P(x \text{ is duplicated to } L1) = 1/2$
$P(x \text{ is duplicated to } L2) = 1/2^2$
$P(x \text{ is duplicated to } Li\,) = 1/2^i$
$P(x \text{ is duplicated to a level } > c \log n, \text{ where } c \geq 1)$ is less than:

$$\frac{1}{2^{c \log_2 n}} = \frac{1}{(2^{\log_2 n})^c} = \frac{1}{n^c}$$

The probability that either the 1^{st} element, the 2^{nd}, or the 3^{rd}, etc., is duplicated to a level > $c \log_2 n$ is less than the sum of the individual probabilities:

$$n \times \frac{1}{n^c} = \frac{1}{n^{c-1}}$$

Memory

Theorem. The expected number of nodes in a randomized skip list of n nodes is:

$$O(n)$$

Proof.

- An element has probability $1/2^i$ of being at level i.
- The expected number of elements at level i is $n/2^i$.
- The expected number elements at levels $0 \longrightarrow L$:

$$\sum_{i=0}^{L} \frac{n}{2^i} = n \sum_{i=0}^{L} \frac{1}{2^i} \leq n \sum_{i=0}^{\infty} \frac{1}{2^i} \leq 2n$$

Search Time

Theorem. The search time for a key in a randomized skip list of n nodes is expected (with probability $1/n^{c-1}$) to be less than or equal to:

$$2c \log n$$

Proof. If we are at a node at level i, there is a 50% chance that there is a copy of the node above it (i.e., the search came from above) and 50% chance that there is no copy above it (i.e., the search came from left). Therefore, the expected time is:

$$T(i) \quad = \quad 1 \quad + \quad \frac{1}{2}T(i-1) \quad + \quad \frac{1}{2}T(i)$$

- $T(i)$: Time spent at level i
- 1: one comparison
- $\frac{1}{2}$: 50% chance
- $T(i-1)$: time spent at the level above
- $\frac{1}{2}$: 50% chance
- $T(i)$: time spent at the same level

Search Time

Theorem. The search time for a key in a randomized skip list of n nodes is expected (with probability $1/n^{c-1}$) to be less than or equal to:

$$2c \log n$$

Proof. If we are at a node at level i, there is a 50% chance that there is a copy of the node above it (i.e., the search came from above) and 50% chance that there is no copy above it (i.e., the search came from left). Therefore, the expected time is:

$$\begin{aligned}
T(i) &= 1 + \tfrac{1}{2}T(i-1) + \tfrac{1}{2}T(i) \\
2T(i) &= 2 + T(i-1) + T(i) \\
T(i) &= 2 + T(i-1) \\
T(i) &= 2 + 2 + \ldots + 2 \quad (c \log n \text{ times}) \\
T(i) &= 2c \log n \qquad \left(\text{with probability } \frac{1}{n^{c-1}}\right)
\end{aligned}$$