

# Design and Analysis of Algorithms

---

CSE 5311

Lecture 12 Skip Lists

Song Jiang, Ph.D.

Department of Computer Science and Engineering



# Skip lists

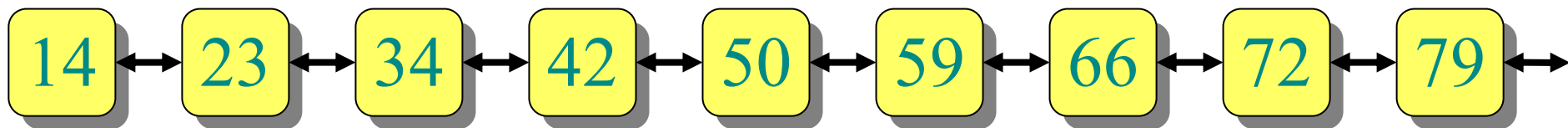
- Simple randomized dynamic search structure
  - Invented by William Pugh in 1989
  - Easy to implement
- Maintains a dynamic set of  $n$  elements in  $O(\lg n)$  time per operation in expectation and *with high probability*
  - Strong guarantee on tail of distribution of  $T(n)$
  - $O(\lg n)$  “almost always”



# One linked list

Start from simplest data structure:  
**(sorted) linked list**

- Searches take  $\Theta(n)$  time in worst case
- How can we speed up searches?

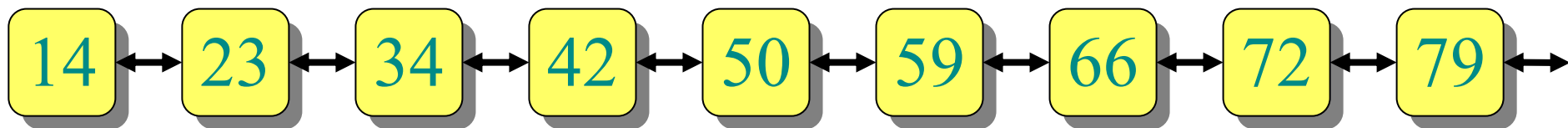




# Two linked lists

Suppose we had *two* sorted linked lists  
(on subsets of the elements)

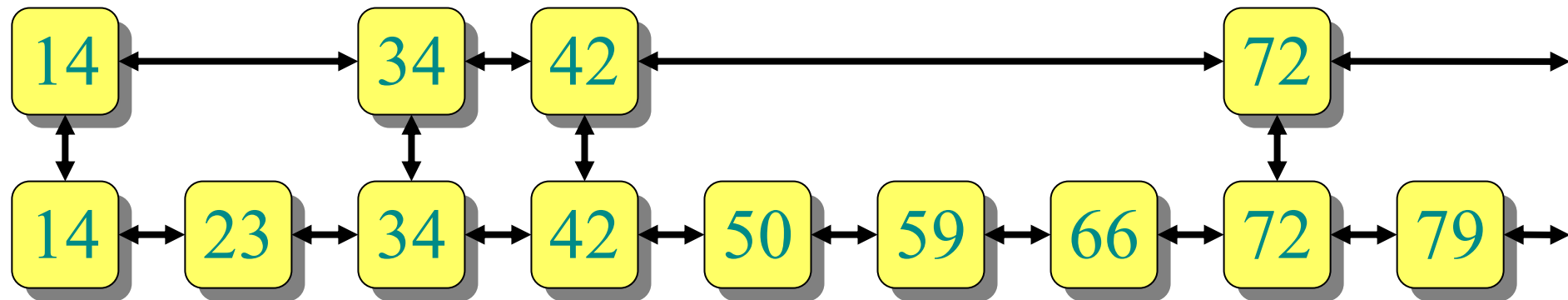
- Each element can appear in one or both lists
- How can we speed up searches?

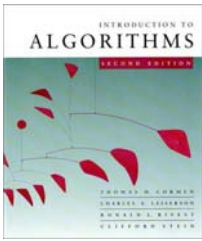




# Two linked lists as a subway

- IDEA:** Express and local subway lines  
(à la New York City 7th Avenue Line)
- Express line connects a few of the stations
  - Local line connects all stations
  - Links between lines at common stations

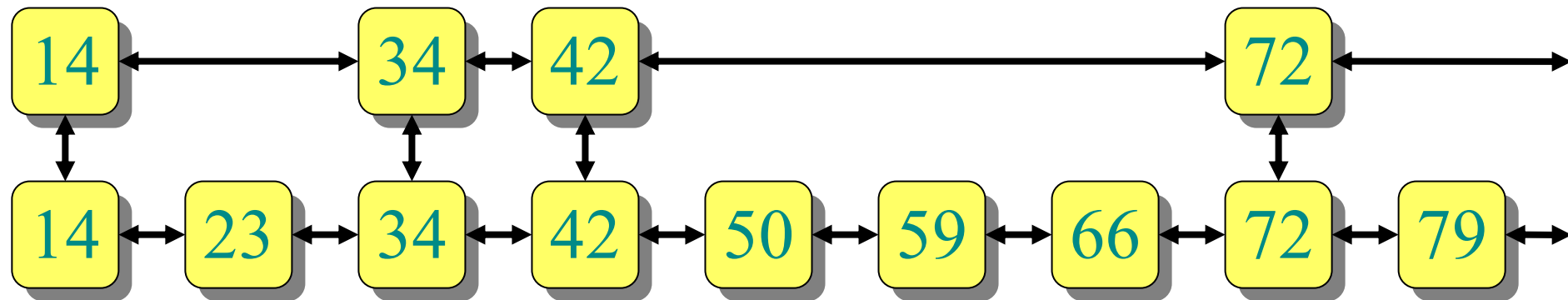


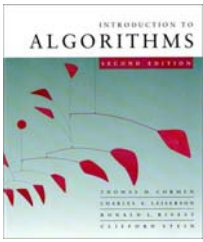


# Searching in two linked lists

SEARCH( $x$ ):

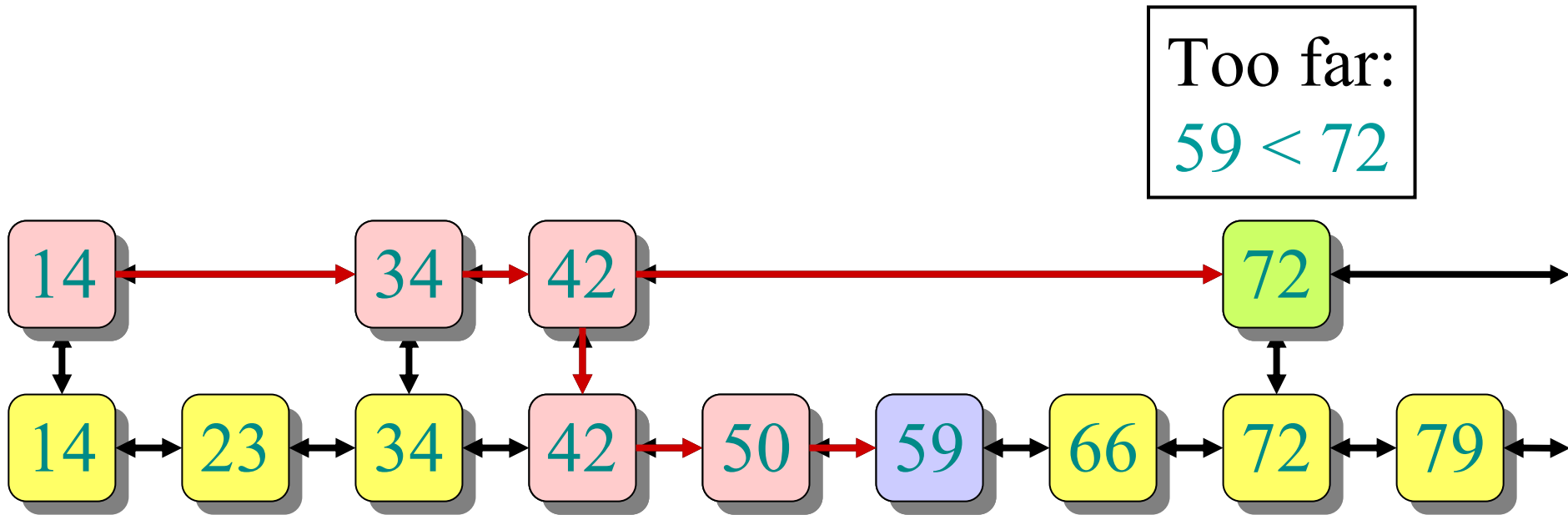
- Walk right in top linked list ( $L_1$ ) until going right would go too far
- Walk down to bottom linked list ( $L_2$ )
- Walk right in  $L_2$  until element found (or not)





# Searching in two linked lists

**EXAMPLE:** SEARCH(59)

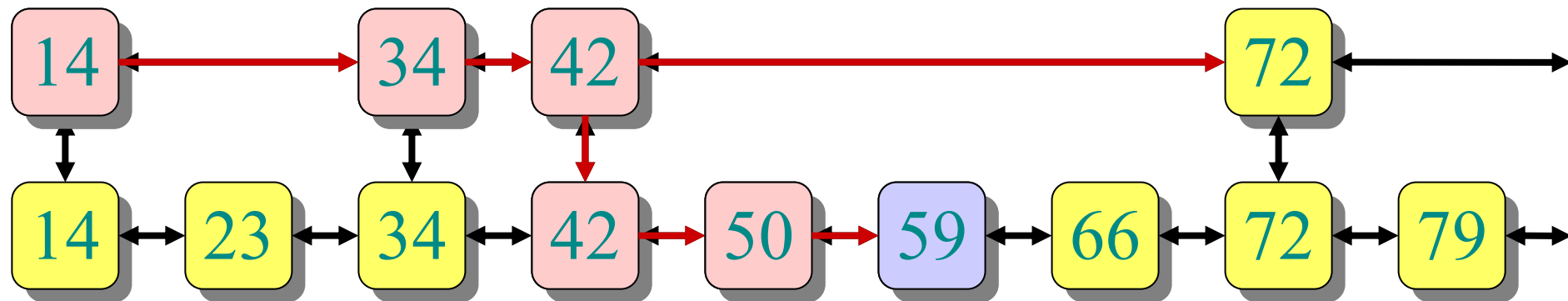




# Design of two linked lists

**QUESTION:** Which nodes should be in  $L_1$ ?

- In a subway, the “popular stations”
- Here we care about *worst-case performance*
- **Best approach:** Evenly space the nodes in  $L_1$
- But *how many nodes* should be in  $L_1$ ?



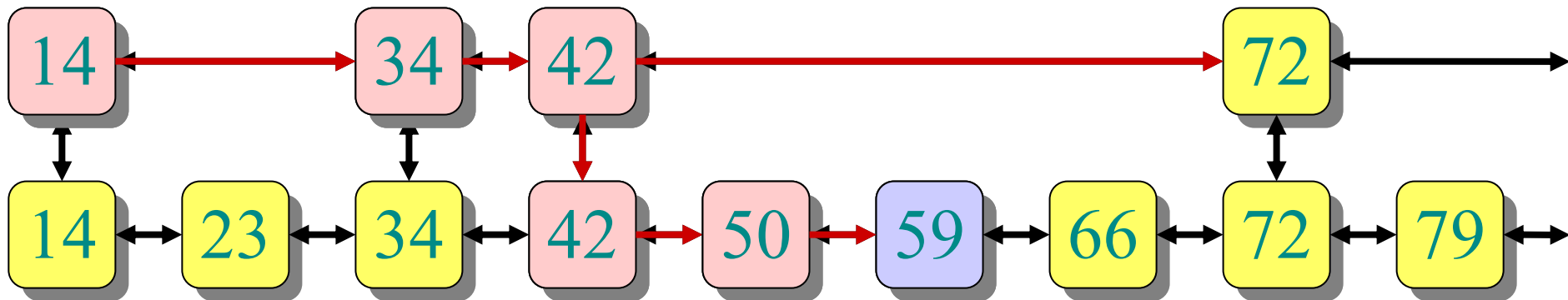




# Analysis of two linked lists

## ANALYSIS:

- Search cost is roughly  $|L_1| + \frac{|L_2|}{|L_1|}$
- Minimized (up to constant factors) when terms are equal
- $|L_1|^2 = |L_2| = n \Rightarrow |L_1| = \sqrt{n}$



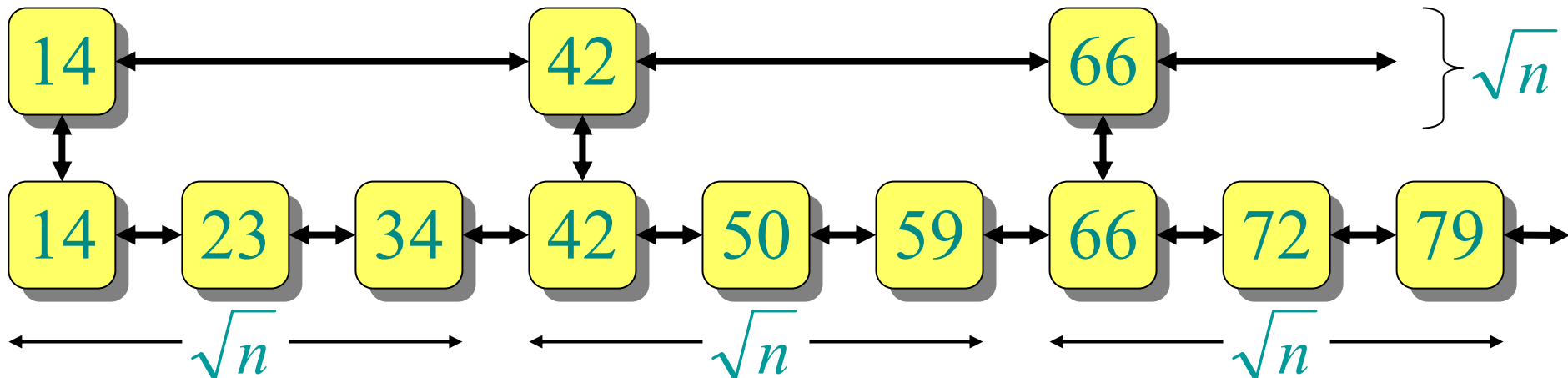


# Analysis of two linked lists

## ANALYSIS:

- $|L_1| = \sqrt{n}$  ,  $|L_2| = n$
- Search cost is roughly

$$|L_1| + \frac{|L_2|}{|L_1|} = \sqrt{n} + \frac{n}{\sqrt{n}} = 2\sqrt{n}$$

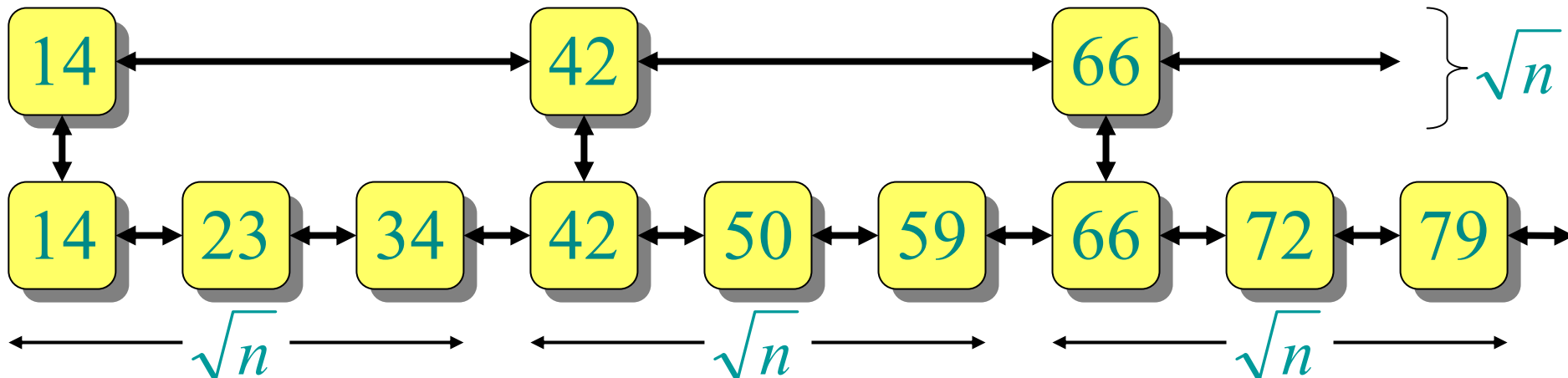


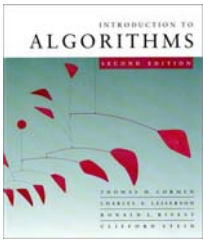


# More linked lists

What if we had more sorted linked lists?

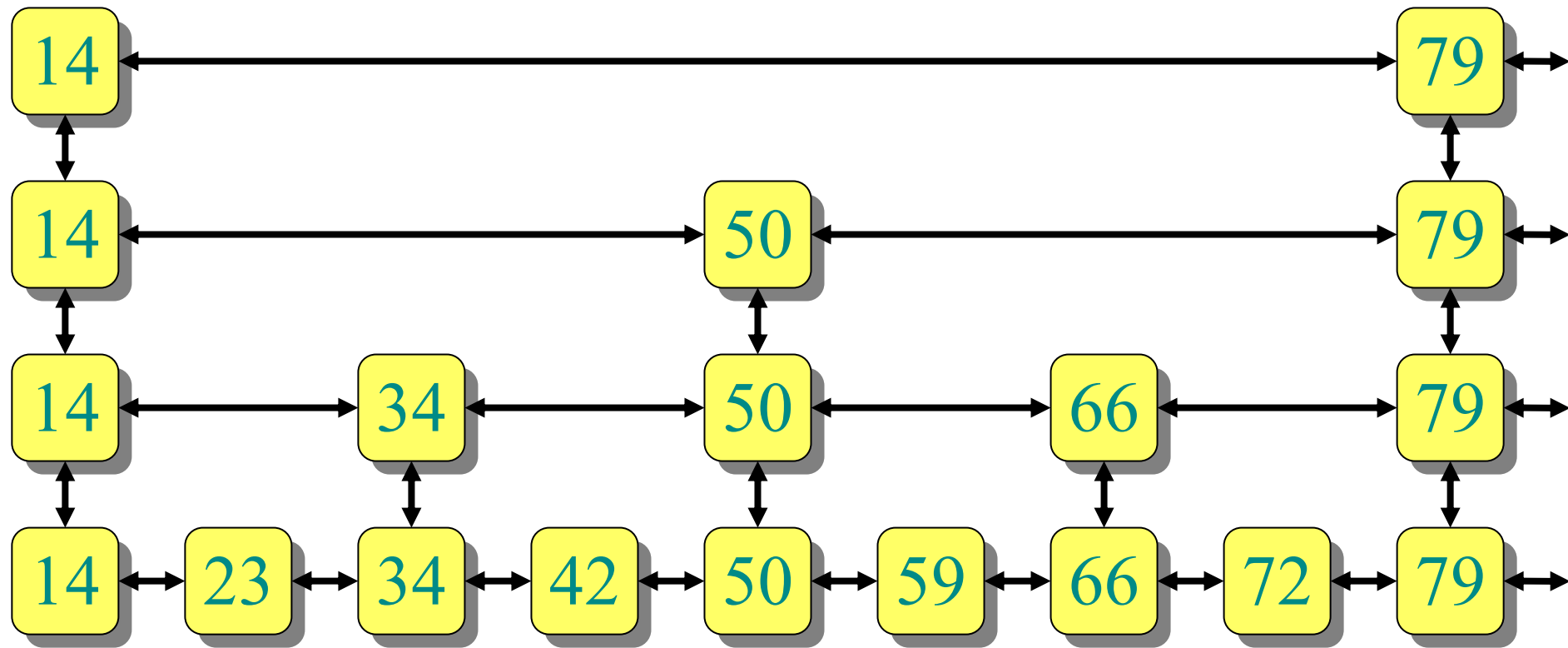
- 2 sorted lists  $\Rightarrow 2 \cdot \sqrt{n}$
- 3 sorted lists  $\Rightarrow 3 \cdot \sqrt[3]{n}$
- $k$  sorted lists  $\Rightarrow k \cdot \sqrt[k]{n}$
- $\lg n$  sorted lists  $\Rightarrow \lg n \cdot \sqrt[\lg n]{n} = 2 \lg n$

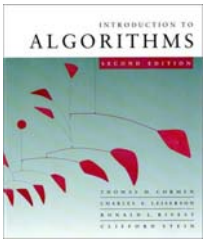




# $\lg n$ linked lists

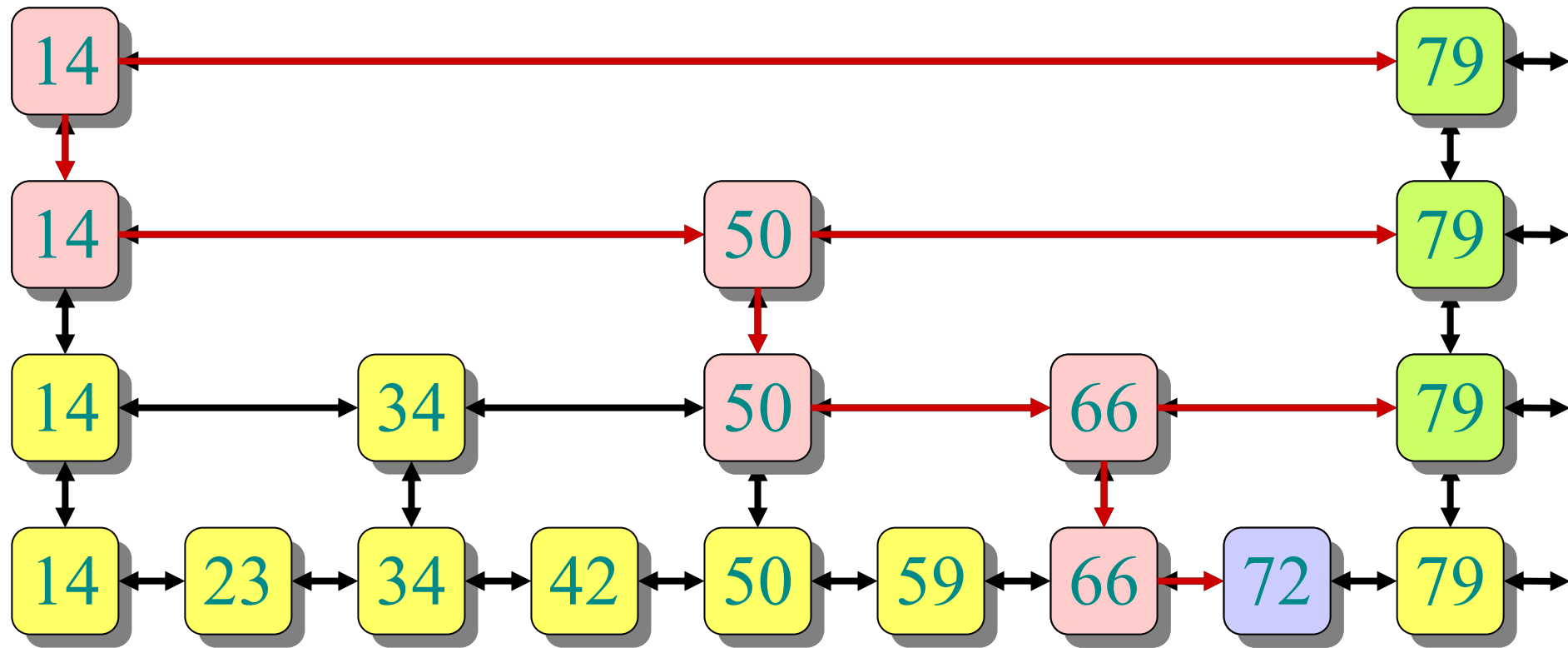
$\lg n$  sorted linked lists are like a binary tree  
(in fact, level-linked  $B^+$ -tree)

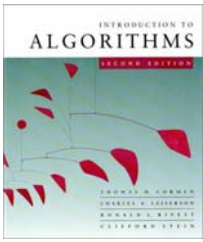




# Searching in $\lg n$ linked lists

**EXAMPLE:** SEARCH(72)

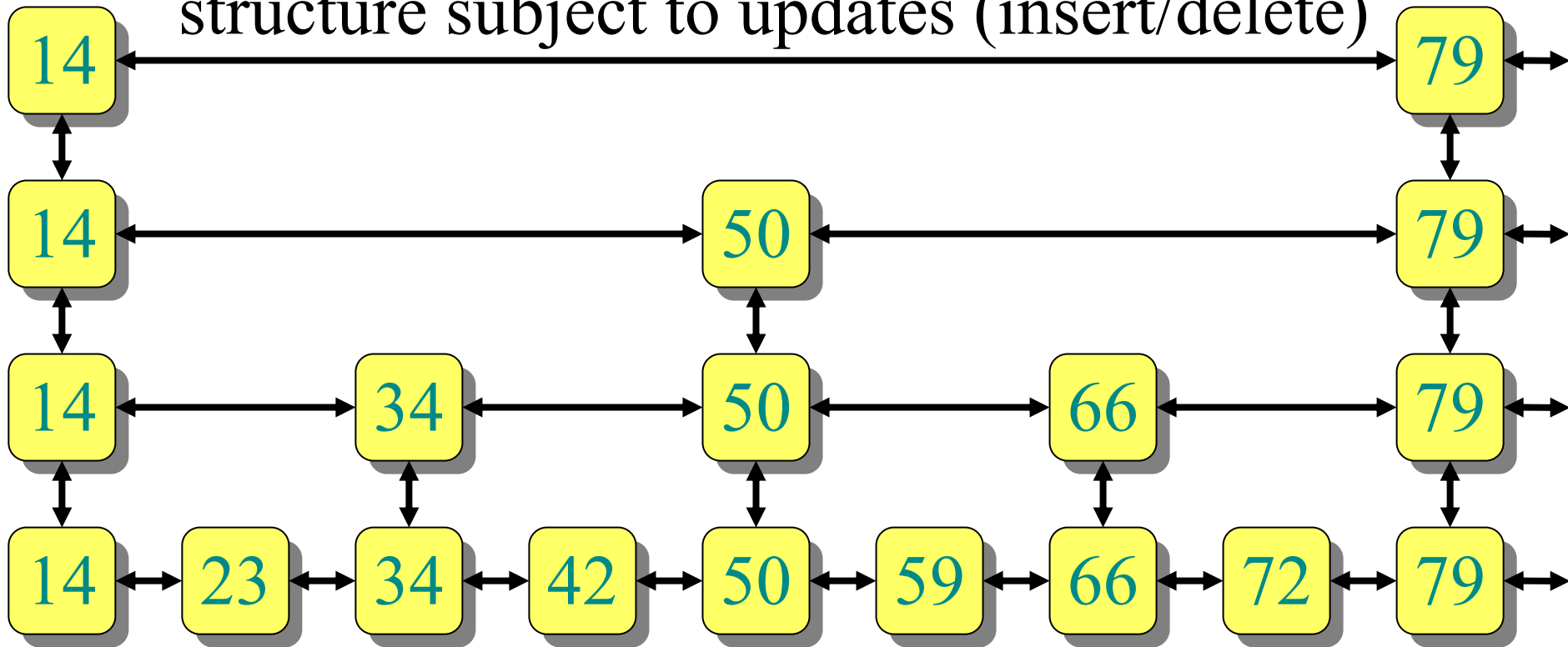




# Skip lists

*Ideal skip list* is this  $\lg n$  linked list structure

*Skip list data structure* maintains roughly this structure subject to updates (insert/delete)





# INSERT( $x$ )

To insert an element  $x$  into a skip list:

- SEARCH( $x$ ) to see where  $x$  fits in bottom list
- Always insert into bottom list

**INVARIANT:** Bottom list contains all elements

- Insert into some of the lists above...

**QUESTION:** To which other lists should we add  $x$ ?



# INSERT( $x$ )

**QUESTION:** To which other lists should we add  $x$ ?

**IDEA:** Flip a (fair) coin; if HEADS,  
*promote*  $x$  to next level up and flip again

- Probability of promotion to next level =  $1/2$
- On average:
  - $1/2$  of the elements promoted 0 levels
  - $1/4$  of the elements promoted 1 level
  - $1/8$  of the elements promoted 2 levels
  - etc.

Approx.  
balanced  
?



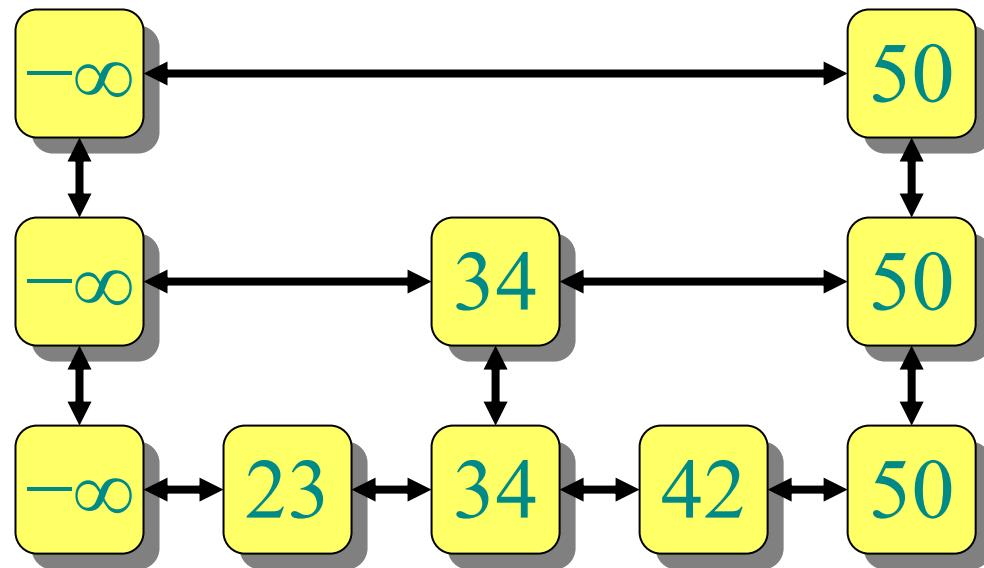


# Example of skip list

**EXERCISE:** Try building a skip list from scratch by repeated insertion using a real coin

## Small change:

- Add special  $-\infty$  value to *every* list  $\Rightarrow$  can search with the same algorithm





# Skip lists

A *skip list* is the result of insertions (and deletions) from an initially empty structure (containing just  $-\infty$ )

- INSERT( $x$ ) uses random coin flips to decide promotion level
- DELETE( $x$ ) removes  $x$  from all lists containing it



# Skip lists

A *skip list* is the result of insertions (and deletions) from an initially empty structure (containing just  $-\infty$ )

- INSERT( $x$ ) uses random coin flips to decide promotion level
- DELETE( $x$ ) removes  $x$  from all lists containing it

**How good are skip lists? (speed/balance)**

- **INTUITIVELY:** Pretty good on average
- **CLAIM:** Really, really good, almost always



# With-high-probability theorem

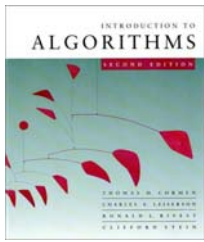
**THEOREM:** *With high probability, every search in an  $n$ -element skip list costs  $O(\lg n)$*



# With-high-probability theorem

**THEOREM:** *With high probability*, every search in a skip list costs  $O(\lg n)$

- **INFORMALLY:** Event  $E$  occurs *with high probability* (*w.h.p.*) if, for any  $\alpha \geq 1$ , there is an appropriate choice of constants for which  $E$  occurs with probability at least  $1 - O(1/n^\alpha)$ 
  - In fact, constant in  $O(\lg n)$  depends on  $\alpha$
- **FORMALLY:** Parameterized event  $E_\alpha$  occurs *with high probability* if, for any  $\alpha \geq 1$ , there is an appropriate choice of constants for which  $E_\alpha$  occurs with probability at least  $1 - c_\alpha/n^\alpha$



# With-high-probability theorem

**THEOREM:** With high probability, every search in a skip list costs  $O(\lg n)$

- **INFORMALLY:** Event  $E$  occurs *with high probability* (*w.h.p.*) if, for any  $\alpha \geq 1$ , there is an appropriate choice of constants for which  $E$  occurs with probability at least  $1 - O(1/n^\alpha)$
- **IDEA:** Can make *error probability*  $O(1/n^\alpha)$  very small by setting  $\alpha$  large, e.g., 100
- Almost certainly, bound remains true for entire execution of polynomial-time algorithm



# Boole's inequality / union bound

*Recall:*

## BOOLE'S INEQUALITY / UNION BOUND:

For any random events  $E_1, E_2, \dots, E_k$ ,

$$\Pr\{E_1 \cup E_2 \cup \dots \cup E_k\} \\ \leq \Pr\{E_1\} + \Pr\{E_2\} + \dots + \Pr\{E_k\}$$

**Application to with-high-probability events:**

If  $k = n^{O(1)}$ , and each  $E_i$  occurs with high probability, then so does  $E_1 \cap E_2 \cap \dots \cap E_k$



# Analysis Warmup

**LEMMA:** With high probability,  
 $n$ -element skip list has  $O(\lg n)$  levels

**PROOF:**

- Error probability for having at most  $c \lg n$  levels  
=  $\Pr\{\text{more than } c \lg n \text{ levels}\}$   
 $\leq n \cdot \Pr\{\text{element } x \text{ promoted at least } c \lg n \text{ times}\}$   
*(by Boole's Inequality)*  
=  $n \cdot (1/2^{c \lg n})$   
=  $n \cdot (1/n^c)$   
=  $1/n^{c-1}$



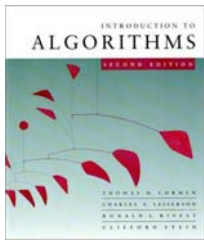


# Analysis Warmup

**LEMMA:** With high probability,  
 $n$ -element skip list has  $O(\lg n)$  levels

**PROOF:**

- Error probability for having at most  $c \lg n$  levels  
 $\leq 1/n^{c-1}$
- This probability is *polynomially small*,  
i.e., at most  $1/n^\alpha$  for  $\alpha = c - 1$ .
- We can make  $\alpha$  arbitrarily large by choosing the  
constant  $c$  in the  $O(\lg n)$  bound accordingly. ◻

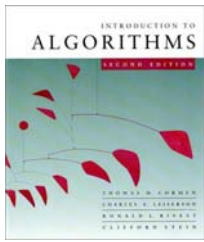


# Proof of theorem

**THEOREM:** With high probability, every search in an  $n$ -element skip list costs  $O(\lg n)$

**COOL IDEA:** Analyze search backwards—leaf to root

- Search starts [ends] at leaf (node in bottom level)
- At each node visited:
  - If node wasn't promoted higher (got TAILS here), then we go [came from] left
  - If node was promoted higher (got HEADS here), then we go [came from] up
- Search stops [starts] at the root (or  $-\infty$ )



# Proof of theorem

**THEOREM:** With high probability, every search in an  $n$ -element skip list costs  $O(\lg n)$

**COOL IDEA:** Analyze search backwards—leaf to root

**PROOF:**

- Search makes “up” and “left” moves until it reaches the root (or  $-\infty$ )
- Number of “up” moves  $<$  number of levels  
 $\leq c \lg n$  w.h.p. (*Lemma*)
- $\Rightarrow$  w.h.p., number of moves is at most the number of times we need to flip a coin to get  $c \lg n$  HEADS



# Coin flipping analysis

**CLAIM:** Number of coin flips until  $c \lg n$  HEADS  
=  $\Theta(\lg n)$  with high probability

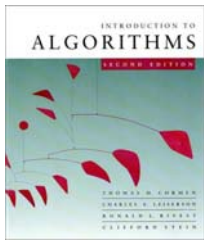
**PROOF:**

Obviously  $\Omega(\lg n)$ : at least  $c \lg n$

Prove  $O(\lg n)$  “by example”:

- Say we make  $10 c \lg n$  flips
- When are there at least  $c \lg n$  HEADS?

(Later generalize to arbitrary values of  $10$ )

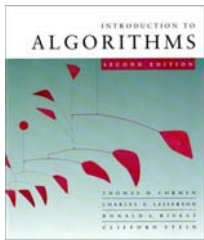


# Coin flipping analysis

**CLAIM:** Number of coin flips until  $c \lg n$  HEADS  
=  $\Theta(\lg n)$  with high probability

**PROOF:**

$$\begin{aligned} \bullet \Pr\{\text{exactly } c \lg n \text{ HEADS}\} &= \underbrace{\binom{10c \lg n}{c \lg n}}_{\text{orders}} \cdot \underbrace{\left(\frac{1}{2}\right)^{c \lg n}}_{\text{HEADS}} \cdot \underbrace{\left(\frac{1}{2}\right)^{9c \lg n}}_{\text{TAILS}} \\ \bullet \Pr\{\text{at most } c \lg n \text{ HEADS}\} &\leq \underbrace{\binom{10c \lg n}{c \lg n}}_{\text{overestimate on orders}} \cdot \underbrace{\left(\frac{1}{2}\right)^{9c \lg n}}_{\text{TAILS}} \end{aligned}$$



# Coin flipping analysis (cont'd)

- Recall bounds on  $\binom{y}{x}$ :  $\left(\frac{y}{x}\right)^x \leq \binom{y}{x} \leq \left(e \frac{y}{x}\right)^x$
- $\Pr\{\text{at most } c \lg n \text{ HEADS}\} \leq \binom{10c \lg n}{c \lg n} \cdot \left(\frac{1}{2}\right)^{9c \lg n}$ 
$$\leq \left(e \frac{10c \lg n}{c \lg n}\right)^{c \lg n} \cdot \left(\frac{1}{2}\right)^{9c \lg n}$$
$$= (10e)^{c \lg n} 2^{-9c \lg n}$$
$$= 2^{\lg(10e) \cdot c \lg n} 2^{-9c \lg n}$$
$$= 2^{[\lg(10e) - 9] \cdot c \lg n}$$
$$= 1/n^\alpha \text{ for } \alpha = [9 - \lg(10e)] \cdot c$$



# Coin flipping analysis (cont'd)

- $\Pr\{\text{at most } c \lg n \text{ HEADS}\} \leq 1/n^\alpha$  for  $\alpha = [9 - \lg(10e)]c$
- **KEY PROPERTY:**  $\alpha \rightarrow \infty$  as  $10 \rightarrow \infty$ , for any  $c$
- So set  $10$ , i.e., constant in  $O(\lg n)$  bound, large enough to meet desired  $\alpha$  □

This completes the proof of the coin-flipping claim and the proof of the theorem.