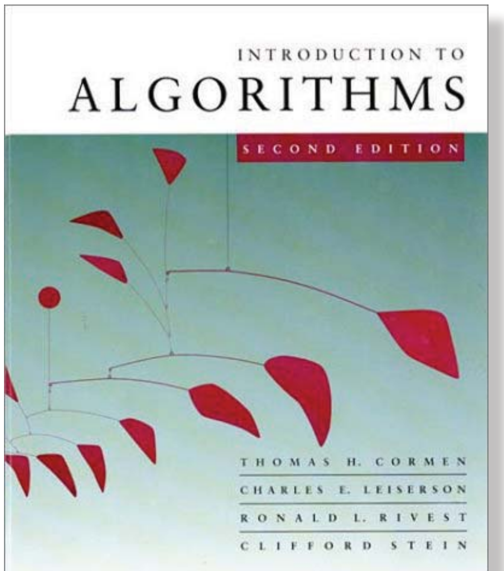
Design and Analysis of Algorithms

6.046J/18.401J



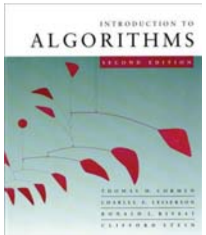
LECTURE 7

Skip Lists

- Data structure
- Randomized insertion
- With high probability (w.h.p.) bound

Skip lists

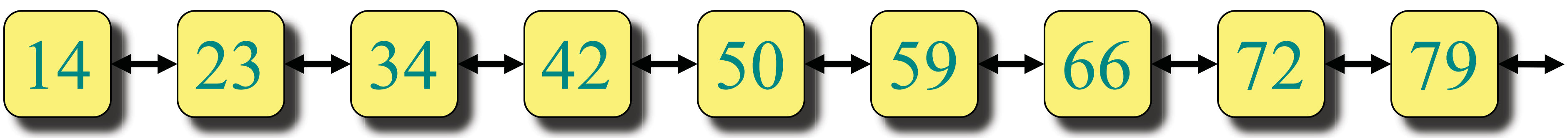
- Simple randomized dynamic search structure
  - Invented by William Pugh in 1989
  - Easy to implement
- Maintains a dynamic set of n elements in $O(\lg n)$ time per operation in expectation and *with high probability*
  - Strong guarantee on tail of distribution of $T(n)$
  - $O(\lg n)$ "almost always"



One linked list

Start from simplest data structure:
(sorted) linked list

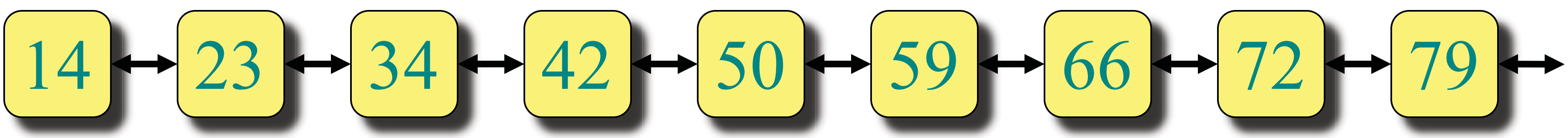
- Searches take $\Theta(n)$ time in worst case
- How can we speed up searches?



Two linked lists

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

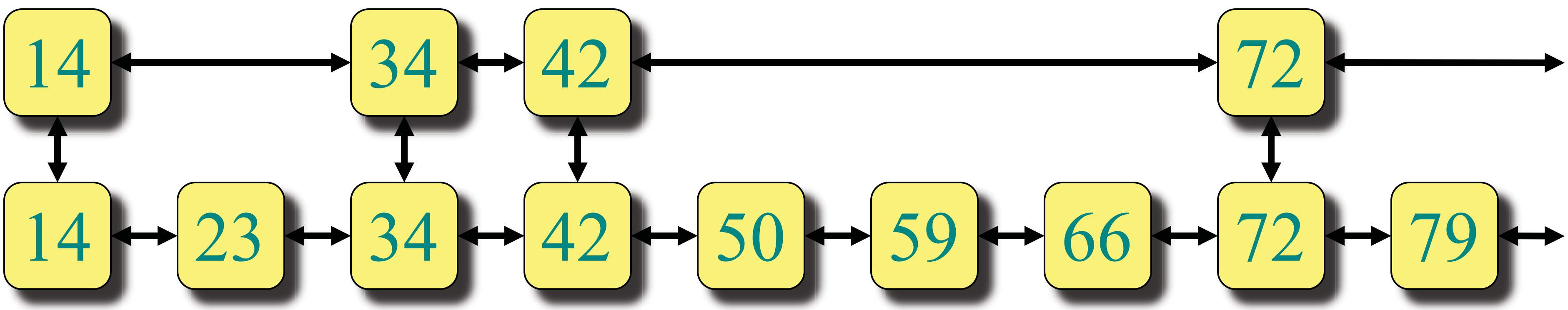
- Each element can appear in one or both lists
- How can we speed up searches?

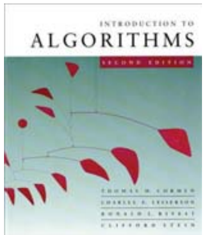


Two linked lists as a subway

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

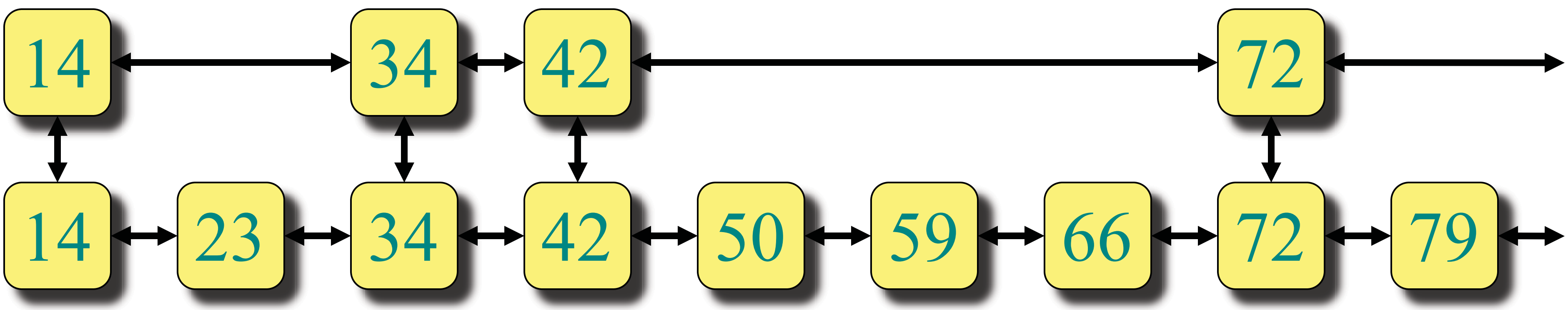




Searching in two linked lists

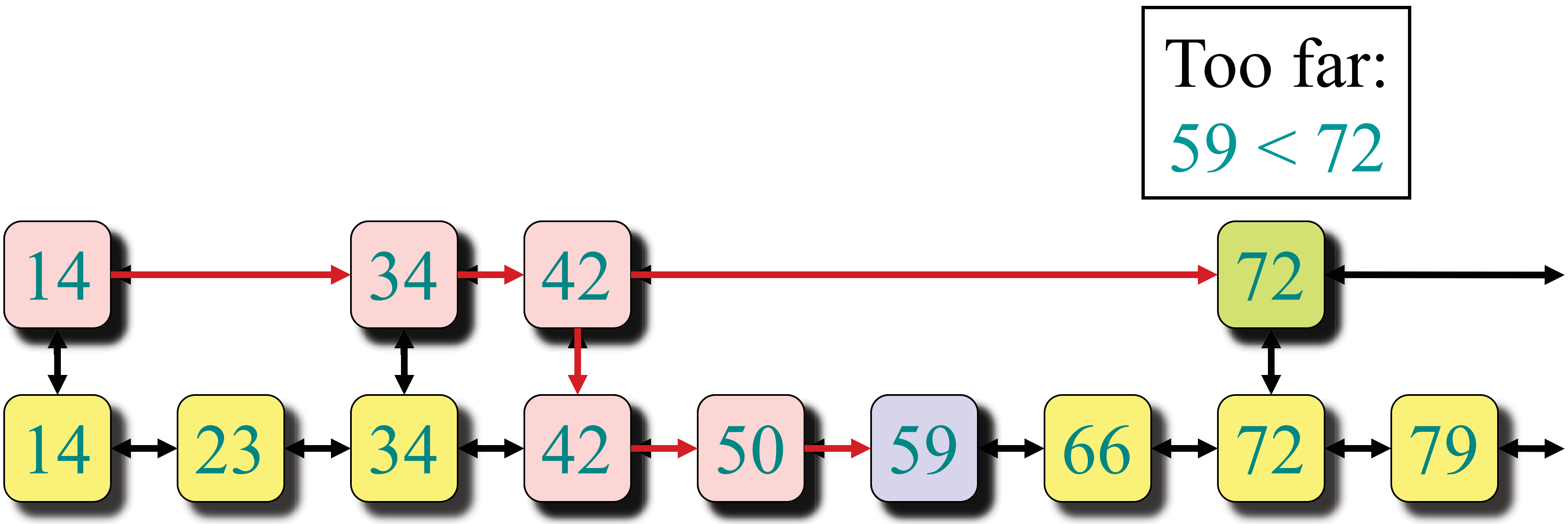
SEARCH(x):

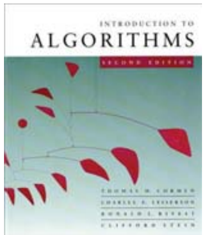
- Walk right in top linked list (L_1) until going right would go too far
- Walk down to bottom linked list (L_2)
- Walk right in L_2 until element found (or not)



Searching in two linked lists

EXAMPLE: SEARCH(59)

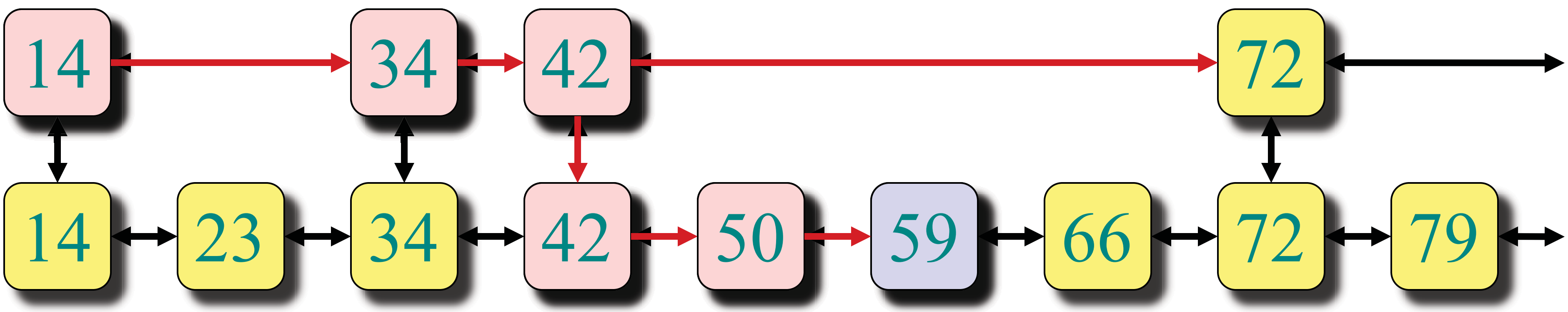


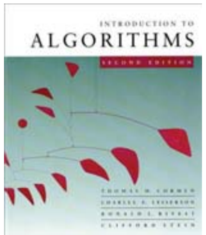


Design of two linked lists

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?

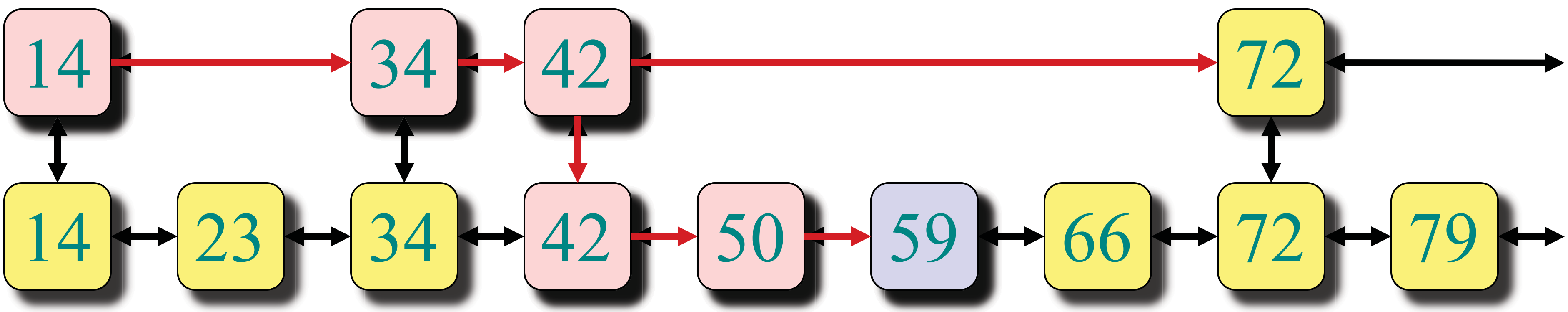


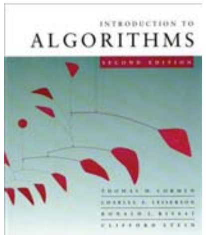


Analysis of two linked lists

ANALYSIS:

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



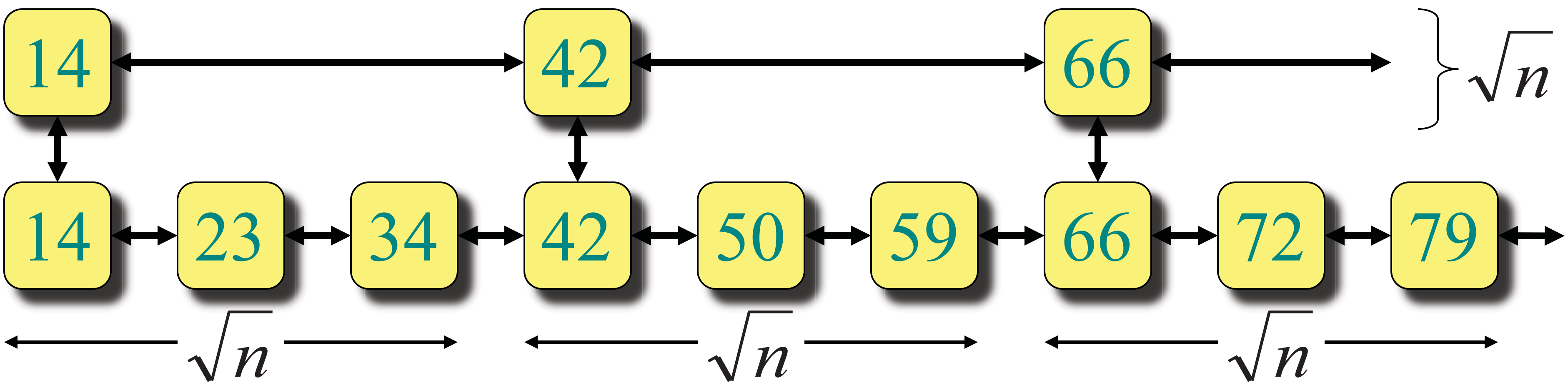


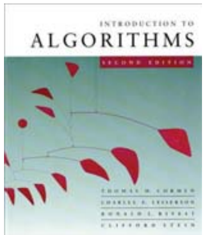
Analysis of two linked lists

ANALYSIS:

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

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

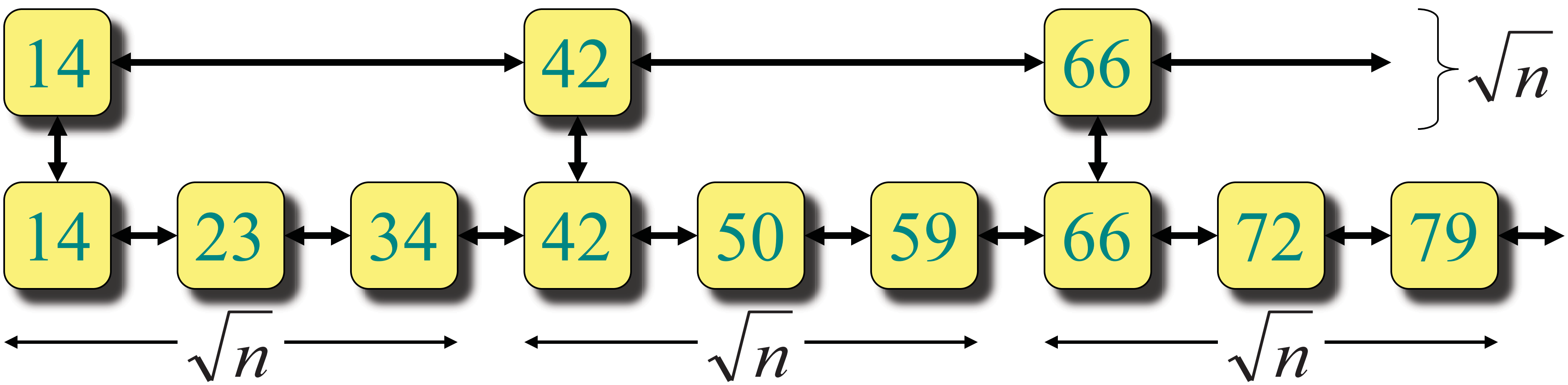


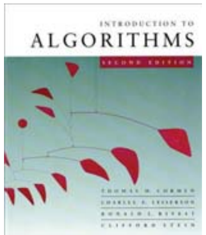


More linked lists

What if we had more sorted linked lists?

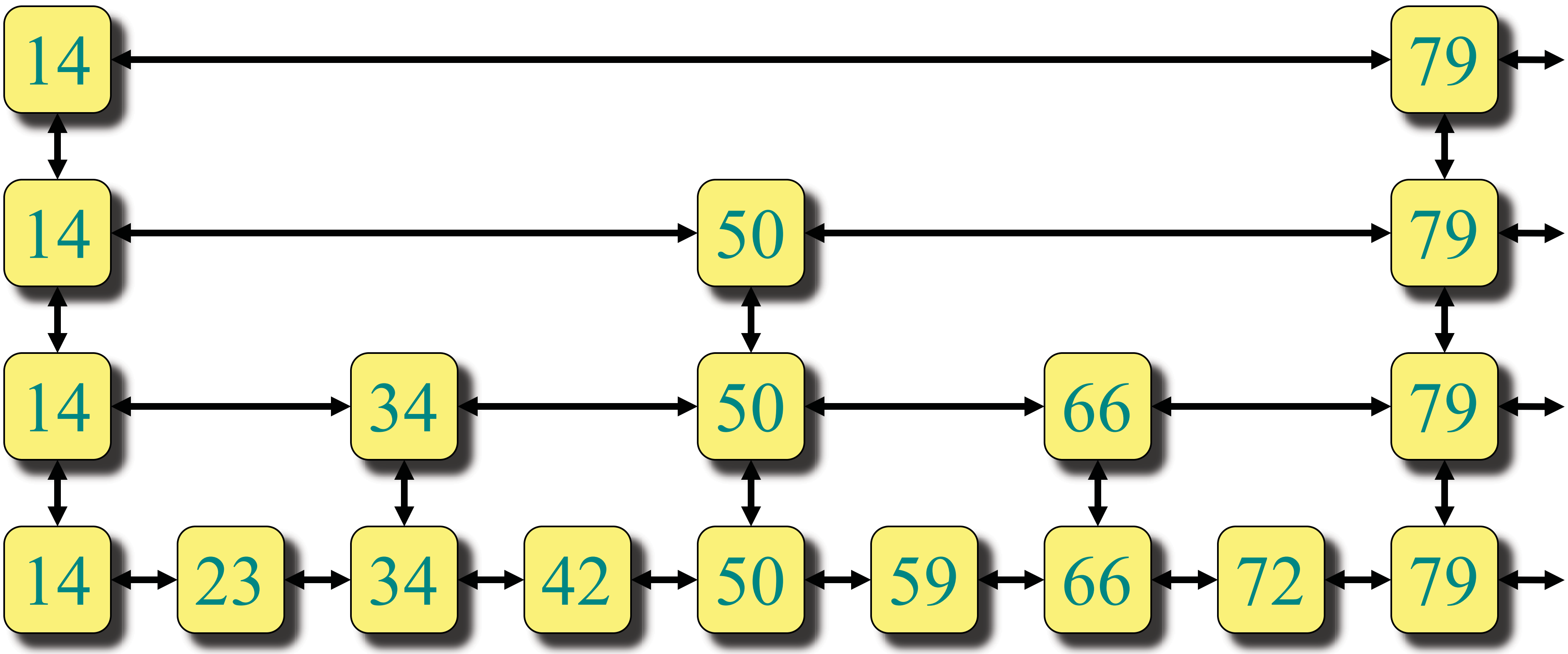
- 2 sorted lists $\Rightarrow 2 \cdot \sqrt{n}$
- 3 sorted lists $\Rightarrow 3 \cdot \sqrt[3]{n}$
- k sorted lists $\Rightarrow k \cdot \sqrt[k]{n}$
- $\lg n$ sorted lists $\Rightarrow \lg n \cdot \sqrt[\lg n]{n} = 2 \lg n$

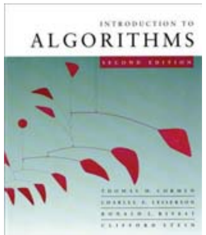




lg n linked lists

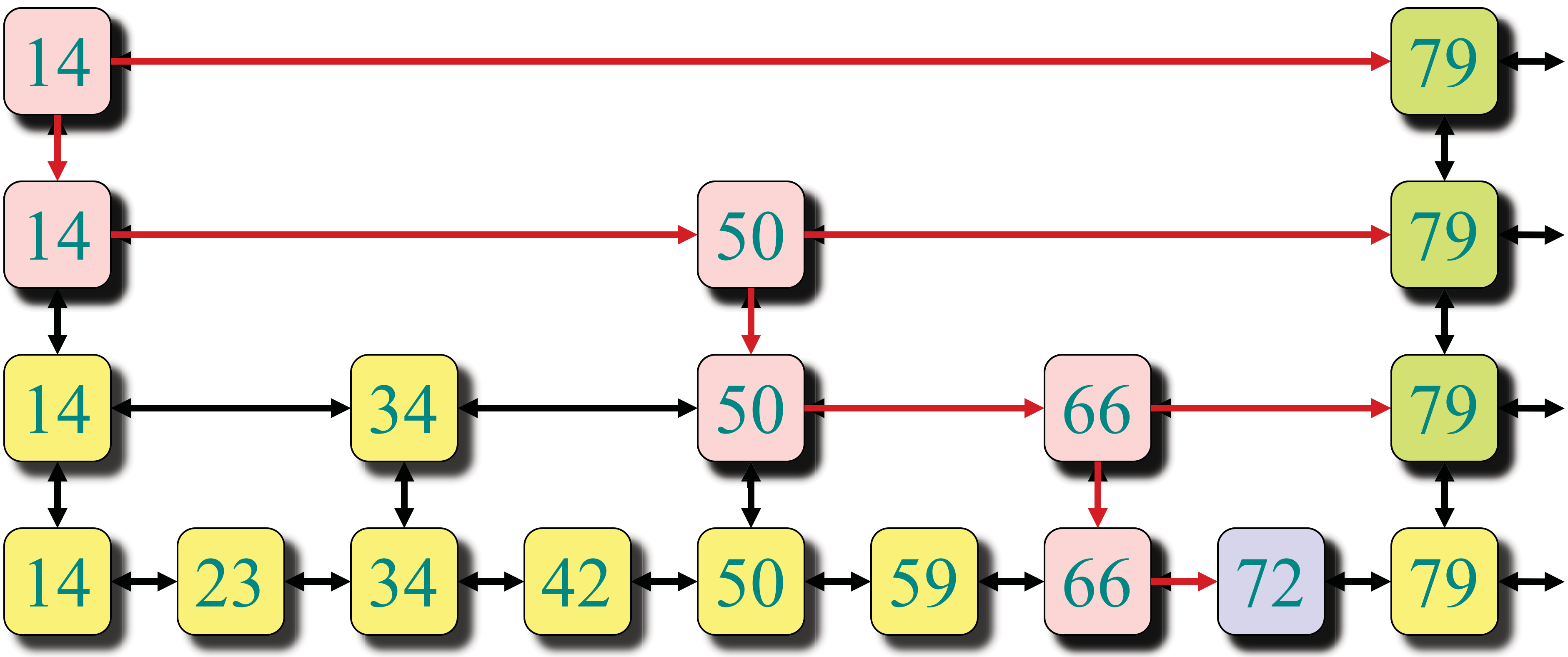
lg n sorted linked lists are like a binary tree (in fact, level-linked B^+-tree)

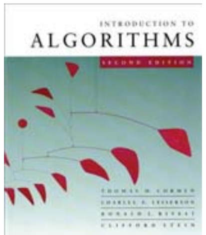




Searching in lg n linked lists

EXAMPLE: SEARCH(72)

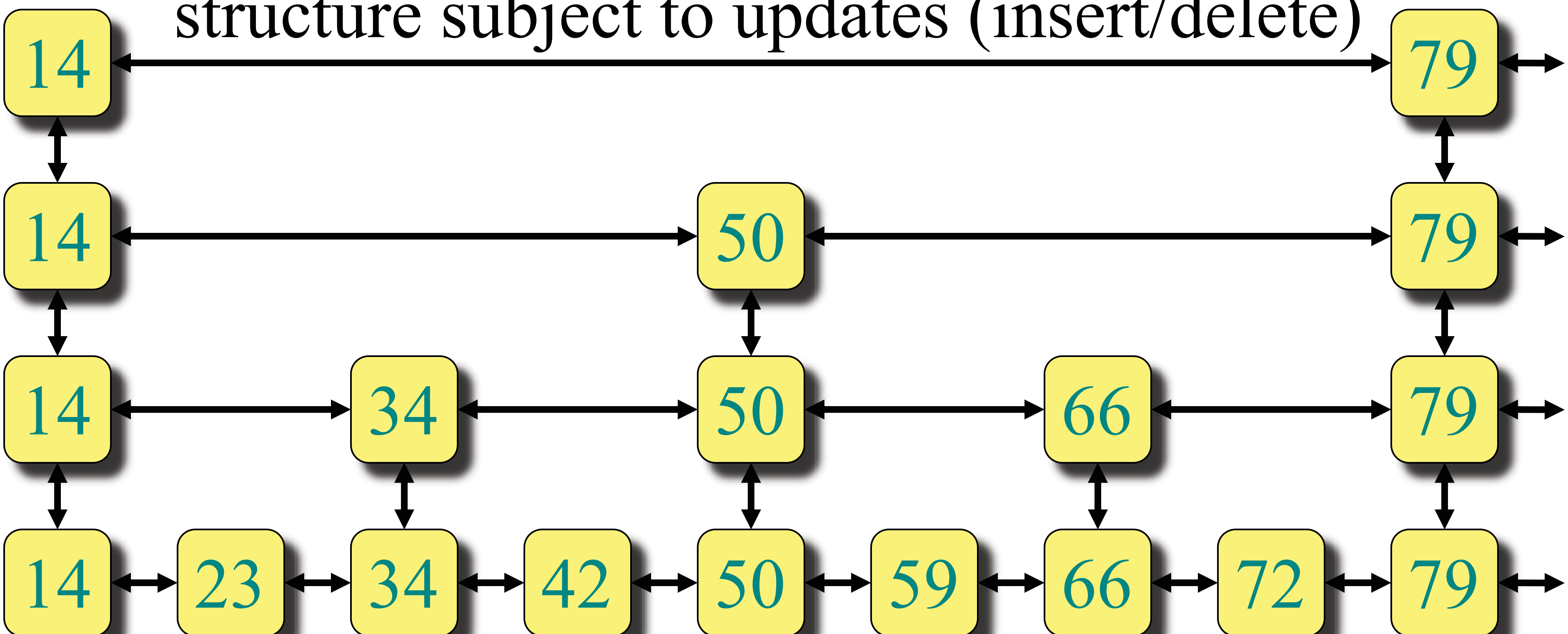


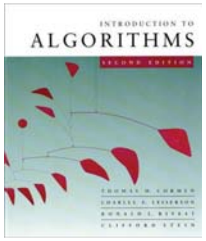


Skip lists

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)

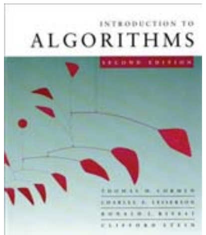
To insert an element x into a skip list:

- 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?



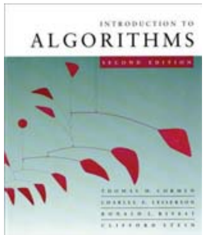
INSERT(x)

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 $= p = 1/2$
- 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?

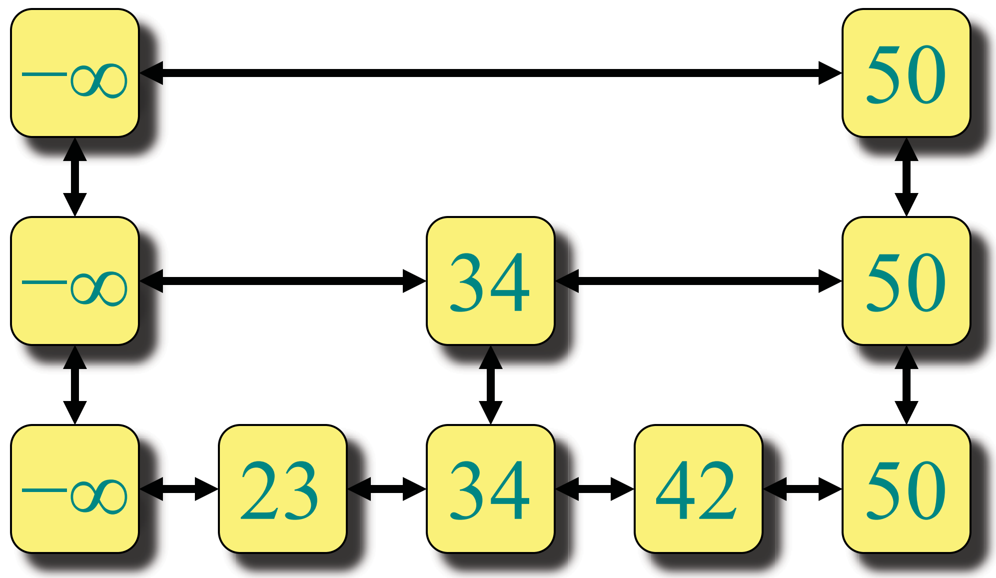


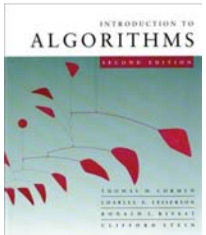
Example of skip list

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

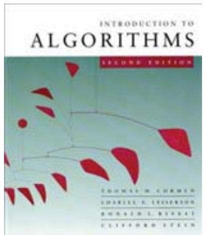
Skip lists

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
- **Expected Time for Search:** $O(\lg n)$

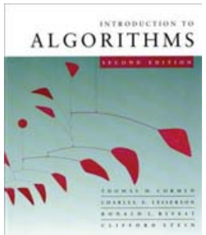


Expected Time for SEARCH

- Search for target begins with head element in top list
- Proceed horizontally until current element greater than or equal to target
- If the current element is equal to the target, it has been found. If the current element is greater than the target, go back to the previous element and drop down vertically to the next lower list and repeat the procedure.
- The expected number of steps in each linked list is seen to be $1/p$, by tracing the search path **backwards** from the target until reaching an element that appears in the next higher list.
- The total *expected* cost of a search is $O(\log_{1/p} n) \cdot (1/p)$ which is $O(\lg n)$ when p is a constant

With-high-probability theorem

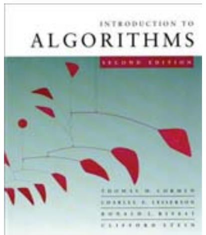
THEOREM: *With high probability*, every search in an n-element skip list costs $O(\lg n)$



With-high-probability theorem

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

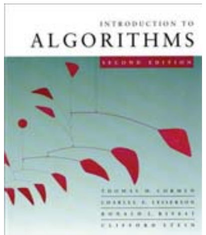
- **INFORMALLY:** Event E occurs ***with high probability*** (***w.h.p.***) if, for any $\alpha \geq 1$, there is an appropriate choice of constants for which E occurs with probability at least $1 - O(1/n^{\alpha})$
  - In fact, constant in $O(\lg n)$ depends on α
- **FORMALLY:** Parameterized event E_{α} occurs ***with high probability*** if, for any $\alpha \geq 1$, there is an appropriate choice of constants for which E_{α} 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 α 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 $E_1, E_2, \ldots, E_k$,

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

Application to with-high-probability events:

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

Analysis Warmup

LEMMA: n-element skip list has $O(\lg n)$ expected number of levels

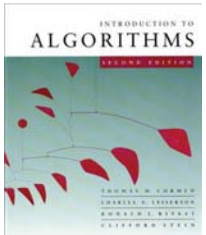
PROOF:

- Probability that x has been promoted once is p
- Probability that x has been promoted k times is pk f
- Expected number of promotions is
- Sigma I = 0 \infty i. p^I = O(log 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)

$$= n \cdot (1/2^{c \lg n})$$
$$= n \cdot (1/n^c)$$
$$= 1/n^{c-1}$$

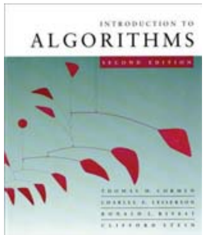


Analysis Warmup

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 n^{α} for $\alpha = c - 1$.
- We can make α arbitrarily large by choosing the constant c in the $O(\lg n)$ bound accordingly. $\square$

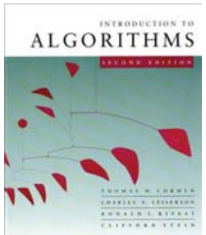


Proof of theorem

THEOREM: Every search in an n-element skip list costs $O(\lg n)$ expected time

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 $-\infty$)



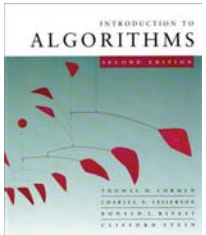
Proof of theorem

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:

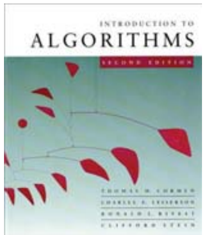
- Search makes "up" and "left" moves until it reaches the root (or $-\infty$)
- Expected number of "up" moves < num. of levels $\leq O(\lg n)$ *(Lemma)*
- $\Rightarrow$ w.h.p., number of moves is at most the number of times we need to flip a coin to get $c \lg n$ HEADS



Chernoff Bounds

THEOREM (CHERNOFF): Let Y be a random variable representing the total number of heads (tails) in a series of m independent coin flips, where each flip has a probability p of coming up heads (tails). Then, for all $r > 0$,

$$Pr[Y \geq E[Y] + r] \leq e^{-2r^2/m}$$



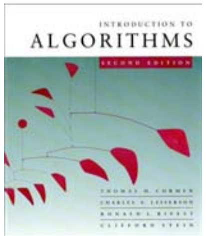
Lemma

LEMMA: For any c there is a constant d such that w.h.p. the number of heads in flipping $d \lg n$ fair coins is at least $c \lg n$.

PROOF: Let Y be the number of tails when flipping a fair coin $d \lg n$ times. $p = ½$.

$m = d \lg n$, so $E[Y] = ½\, m = ½\, d \lg n$

We want to bound the probability of $\leq c \lg n$ heads = probability of $\geq d \lg n - c \lg n$ tails.



Lemma Proof (contd.)

$$Pr[Y \geq (d - c)\lg n] =$$

$$Pr[Y \geq E[Y] + (½\, d - c)\lg n]$$

Choose $d = 3c \Rightarrow r = 3c\lg n$

By Chernoff, probability of $\leq c\lg n$ heads is

$$\leq e^{-2r^2/m} = e^{-2(3c\lg n)^2/8c\lg n} = e^{-9/4c\lg n}$$

$$\leq e^{-c\lg n}$$

$$\leq 2^{-c\lg n} \quad (e > 2)$$

$$= 1/n^c$$

Proof of theorem (finally!)

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

event A: number of levels $\leq c \lg n$ w.h.p.

event B: number of moves until $c \lg n$ "up" moves $\leq d \lg n$ w.h.p.

A and B are not independent!

Want to show A & B occurs w.h.p. to prove theorem

$$
\begin{aligned}
Pr(\overline{A \,\&\, B}) = Pr(\overline{A} + \overline{B}) &\leq Pr(\overline{A}) + Pr(\overline{B}) \quad \textit{(union bound)} \\
&\leq 1/n^{c-1} + 1/n^{c} \\
&= O(1/n^{c-1})
\end{aligned}
$$