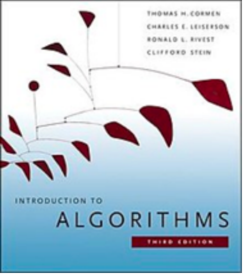
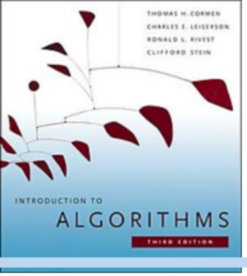


Introduction to Algorithms
12. Skip Lists

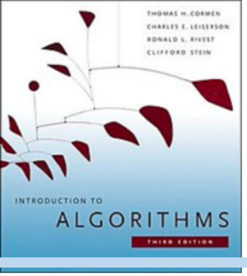
Hyungsoo Jung

Scalable Computing Systems Laboratory
Hanyang University



Skip lists

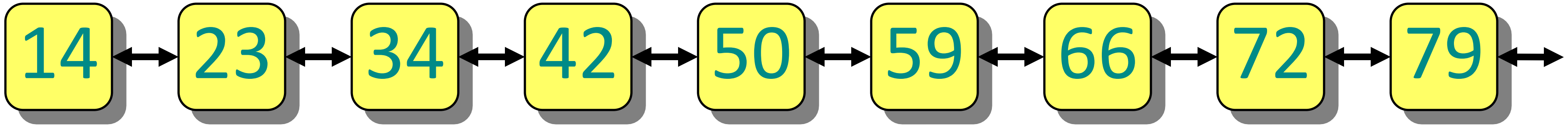
- Simple randomized dynamic search structure
  - Invented by William Pugh in 1989
  - Easy to implement
- Maintains a dynamic set of n elements in $O(\lg n)$ time per operation in expectation and *with high probability*
  - Strong guarantee on tail of distribution of $T(n)$
  - $O(\lg n)$ "almost always"

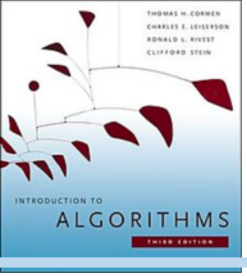


One linked list

Start from simplest data structure:
(sorted) linked list

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



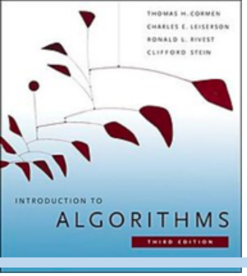


Two linked lists

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?

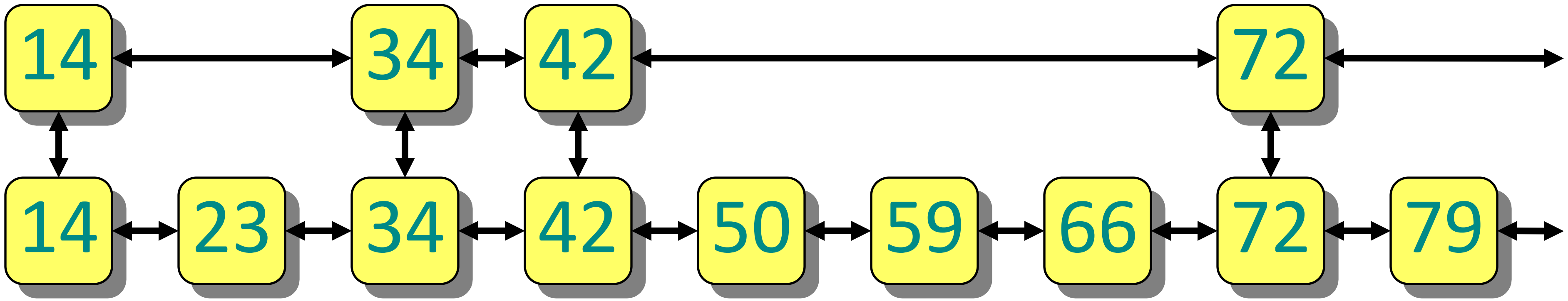
14 ↔ 23 ↔ 34 ↔ 42 ↔ 50 ↔ 59 ↔ 66 ↔ 72 ↔ 79 ↔

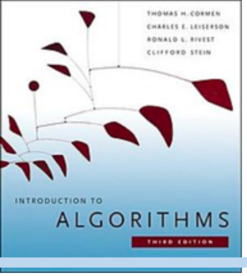


Two linked lists as a subway

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

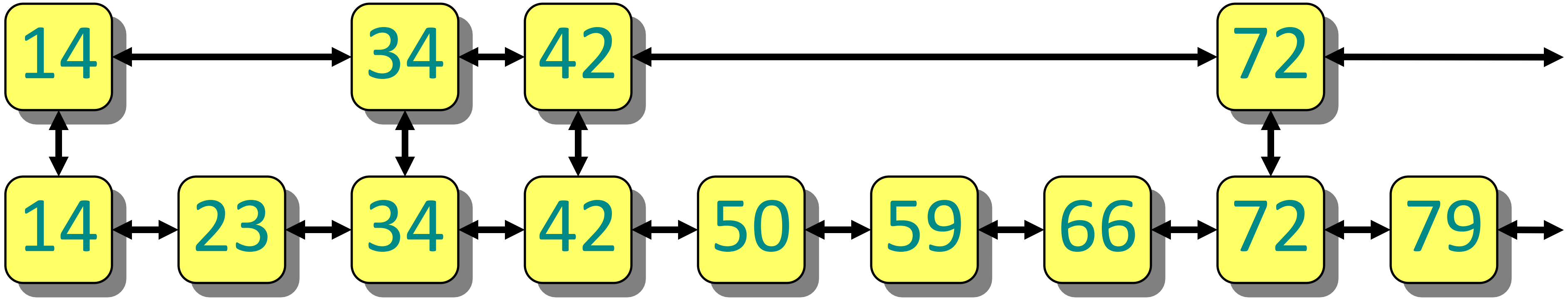


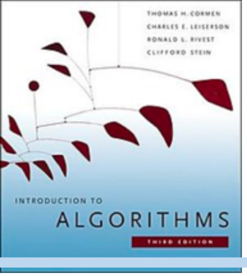


Searching in two linked lists

SEARCH(x):

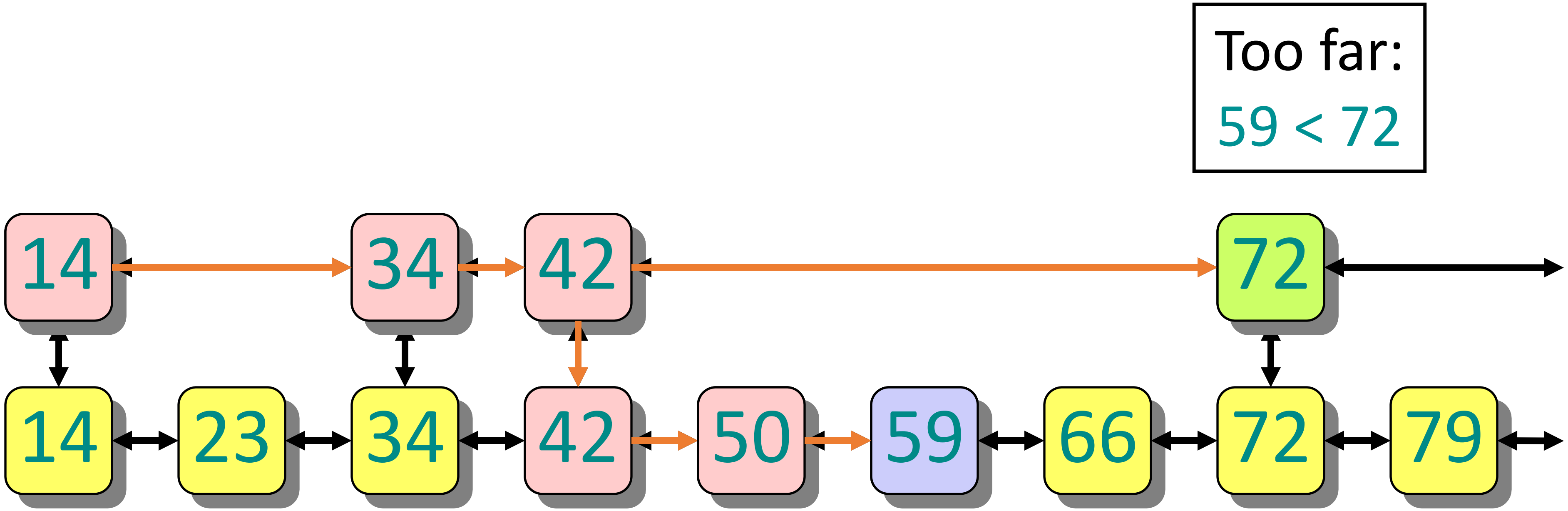
- Walk right in top linked list (L_1) until going right would go too far
- Walk down to bottom linked list (L_2)
- Walk right in L_2 until element found (or not)





Searching in two linked lists

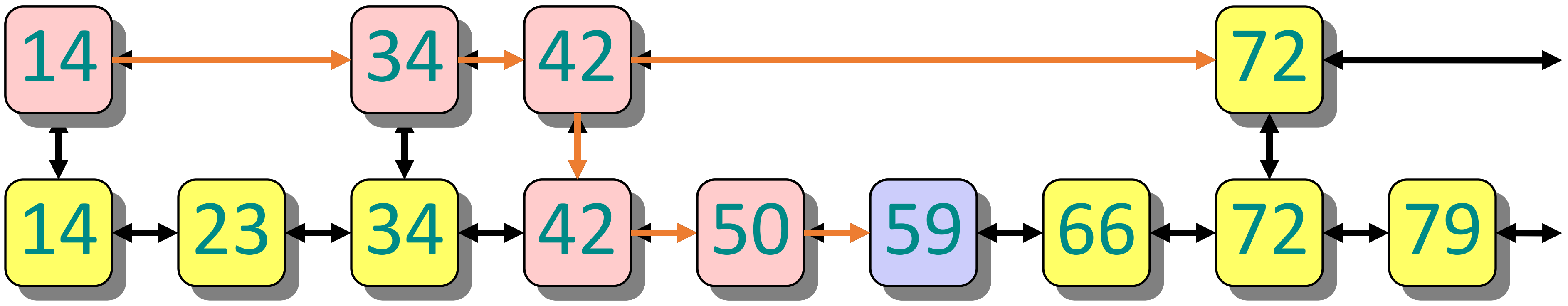
EXAMPLE: SEARCH(59)

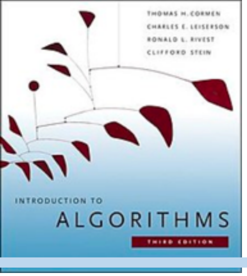


Design of two linked lists

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?

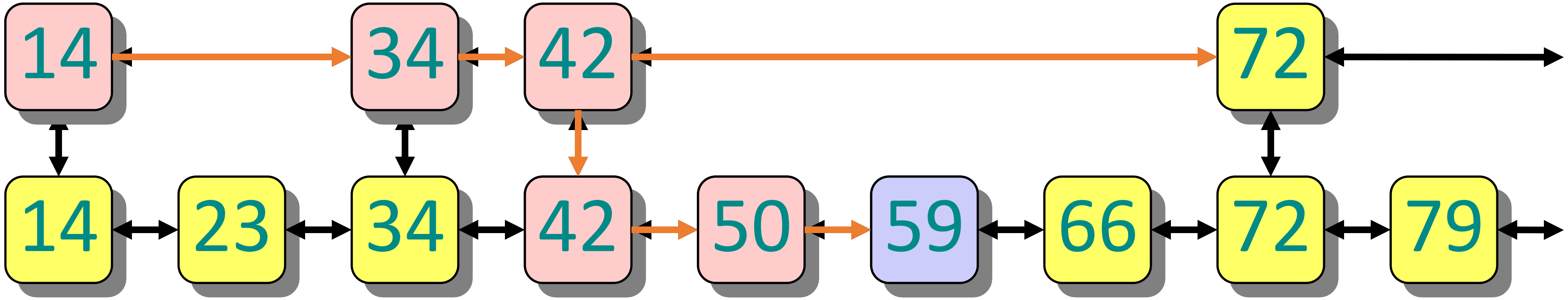


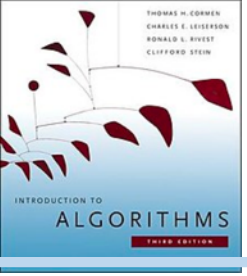


Analysis of two linked lists

ANALYSIS:

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



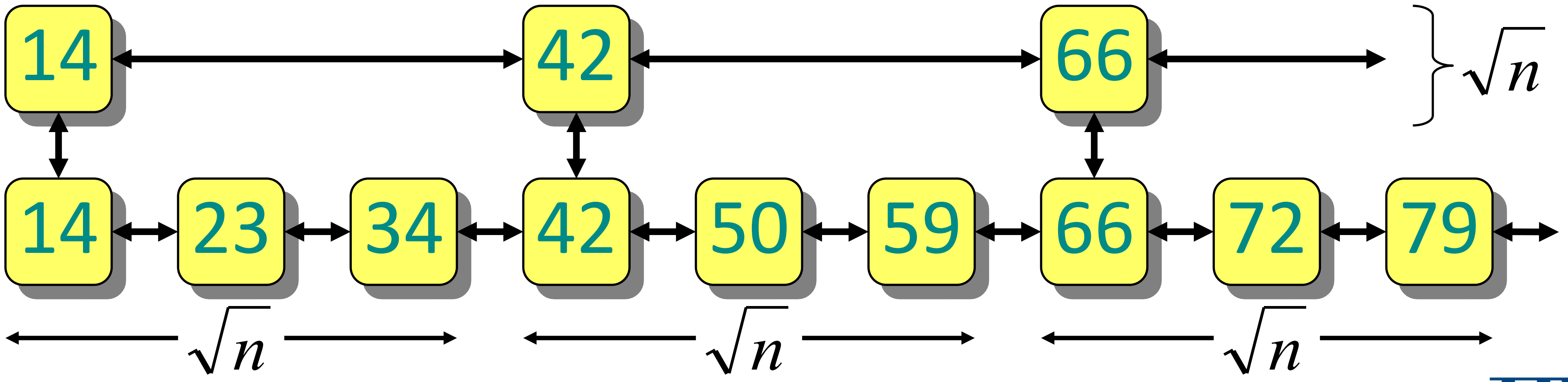


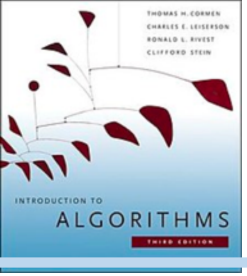
Analysis of two linked lists

ANALYSIS:

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

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

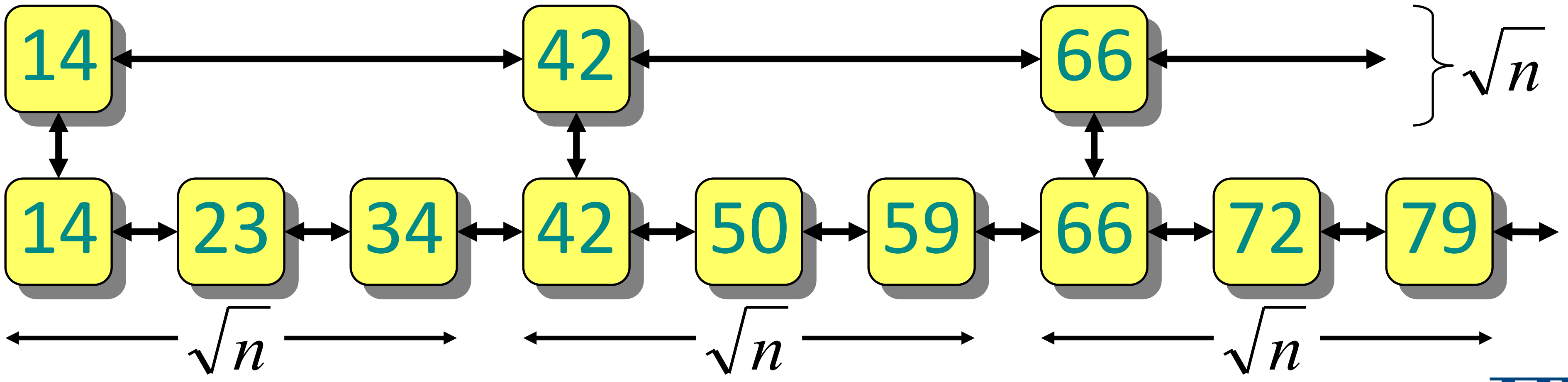


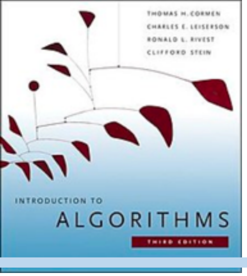


More linked lists

What if we had more sorted linked lists?

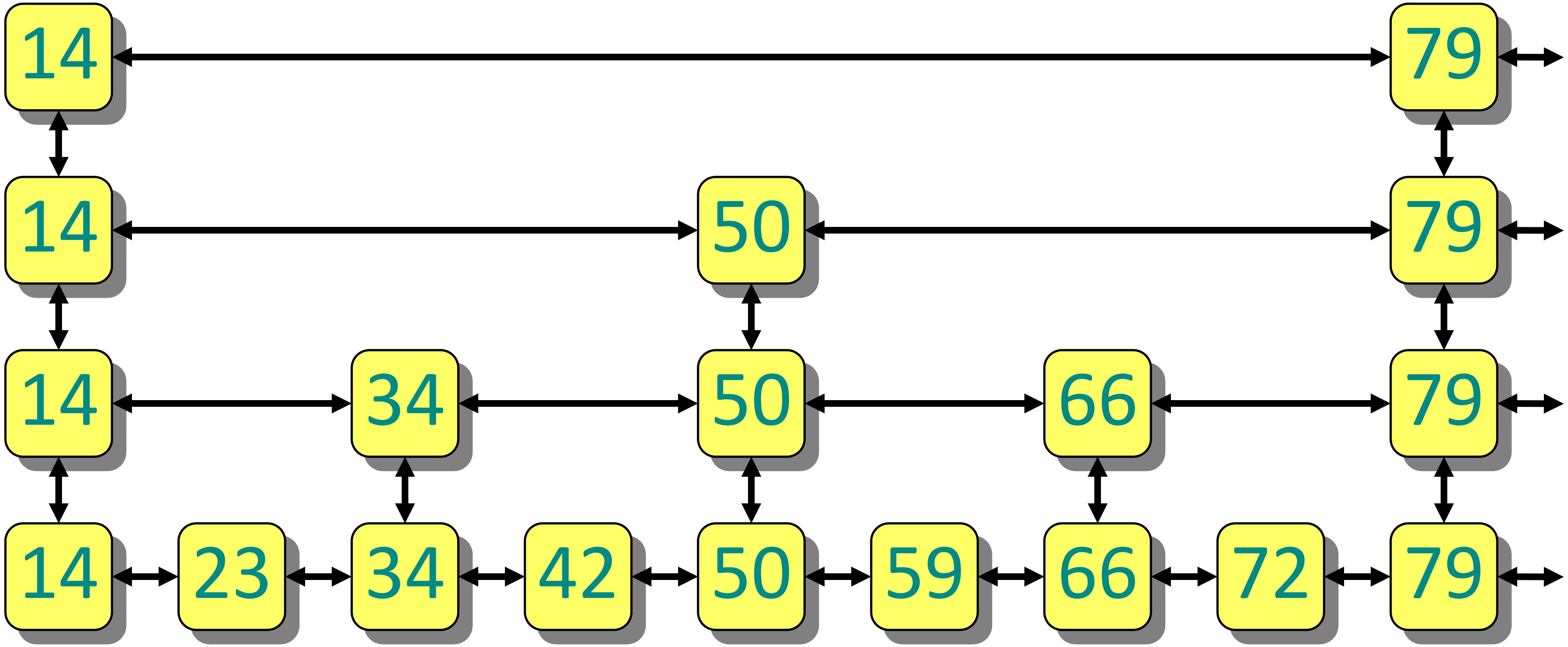
- 2 sorted lists $\Rightarrow$ $2 \cdot \sqrt{n}$
- 3 sorted lists $\Rightarrow$ $3 \cdot \sqrt[3]{n}$
- k sorted lists $\Rightarrow$ $k \cdot \sqrt[k]{n}$
- lg n sorted lists $\Rightarrow$ $\lg n \cdot \sqrt[\lg n]{n} = 2 \lg n$





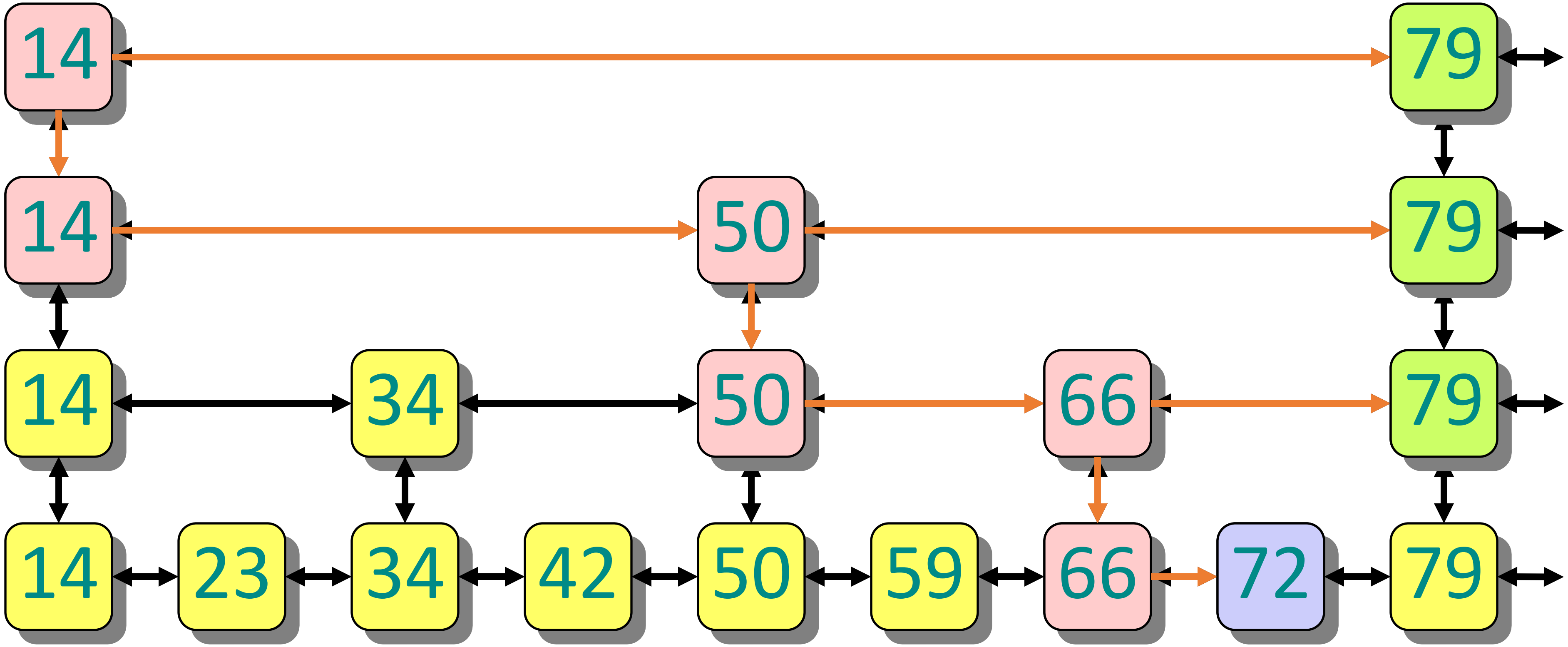
lg *n* linked lists

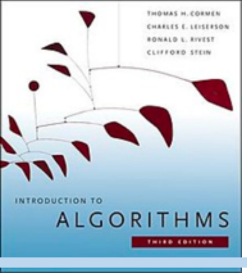
lg *n* sorted linked lists are like a binary tree
(in fact, level-linked B+-tree; see Problem Set 5)



Searching in lg *n* linked lists

EXAMPLE: SEARCH(72)

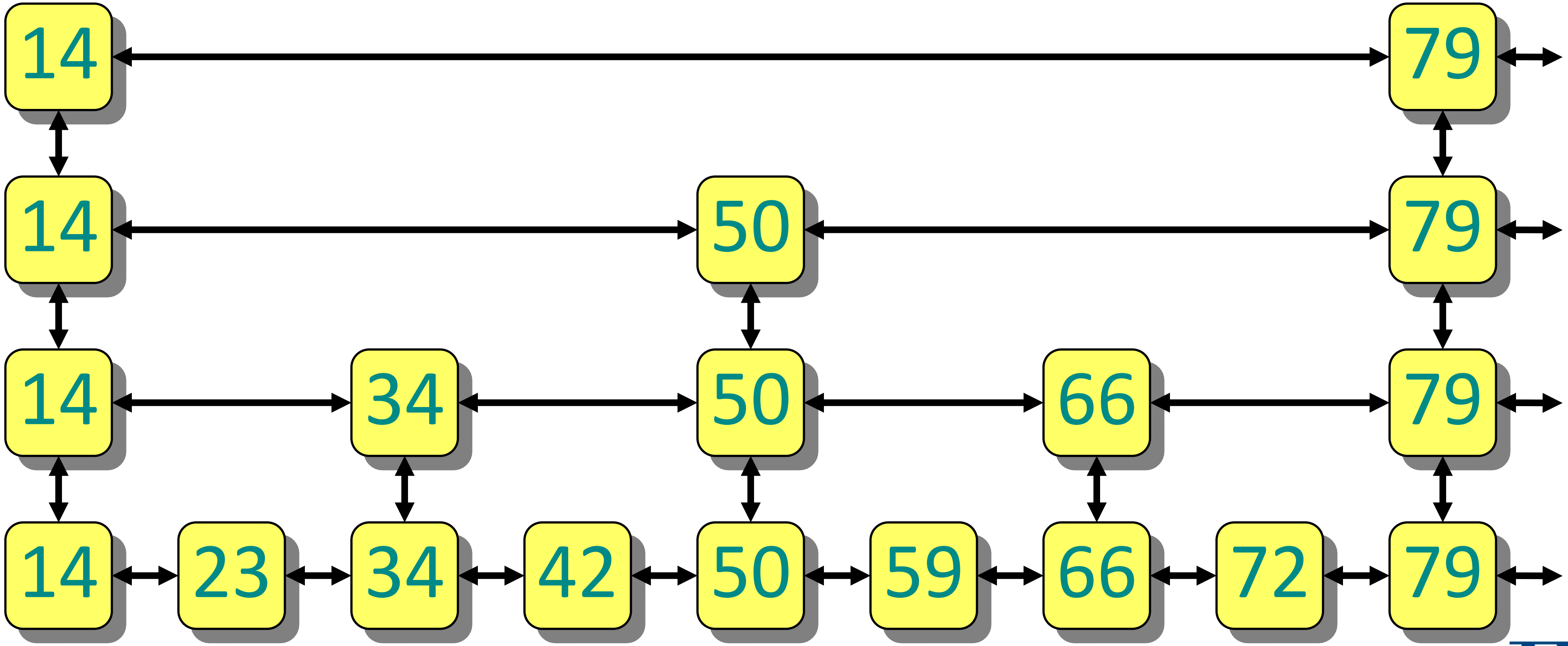




Skip lists

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*)

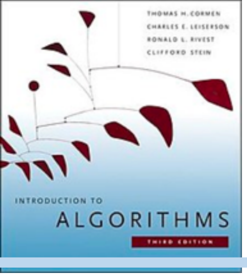
To insert an element *x* into a skip list:

- 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*?



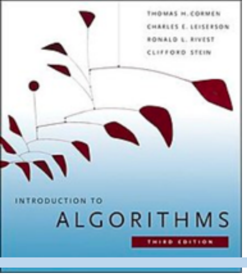
INSERT(x)

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?

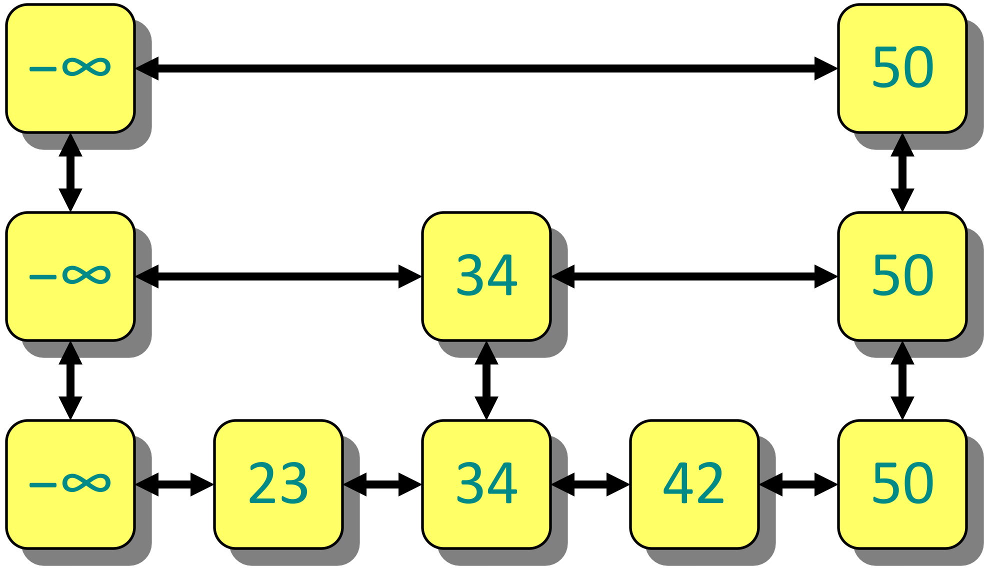


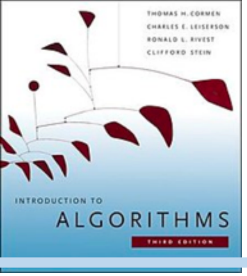
Example of skip list

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

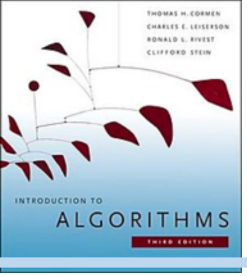




Skip lists

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



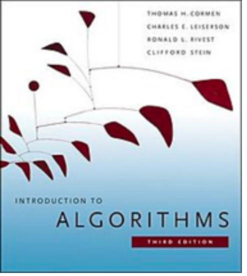
Skip lists

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


THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN
INTRODUCTION TO
ALGORITHMS
THIRD EDITION

Thank You