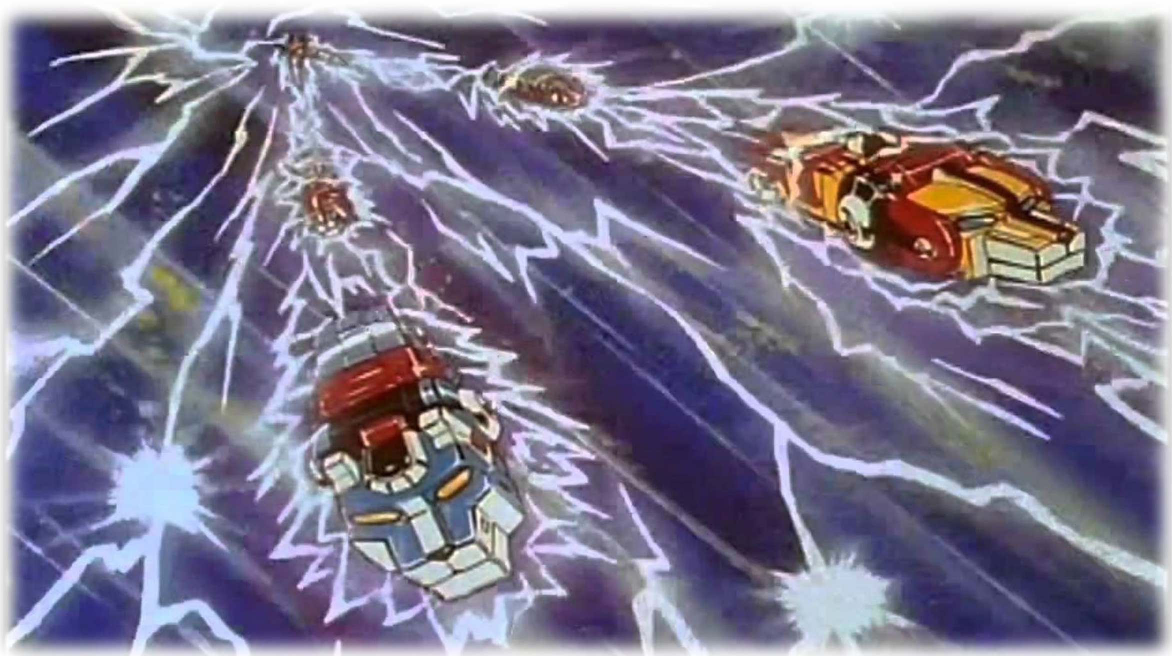
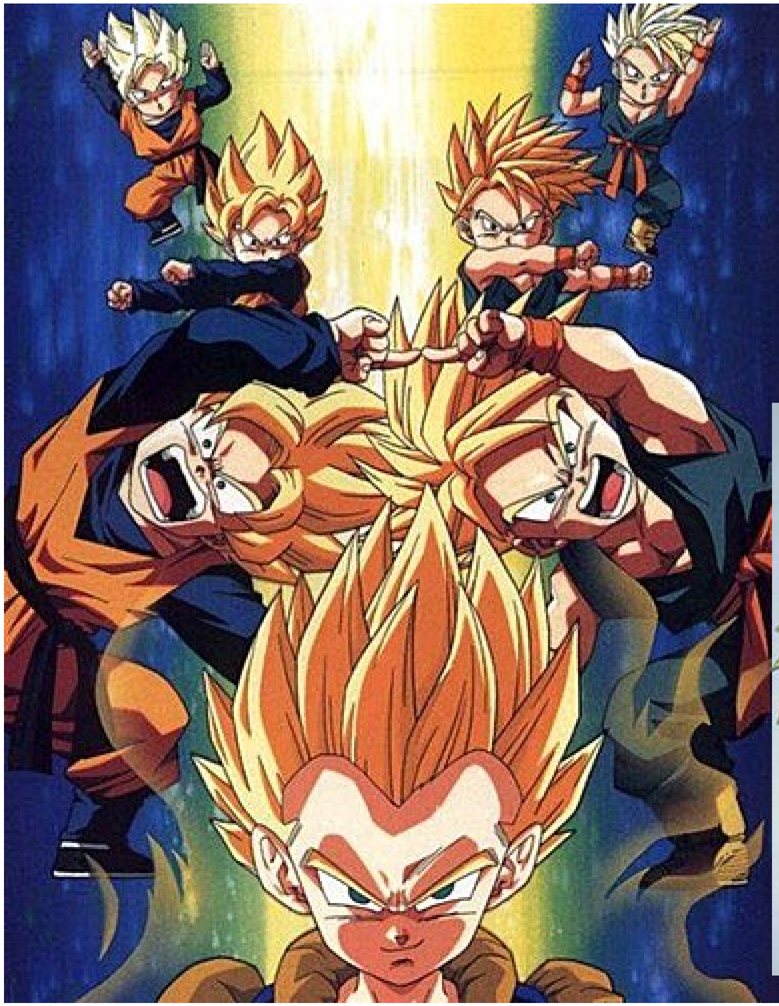
Data Structures with Multiple Organization




RED LION
GREEN LION
BLACK LION
YELLOW LION
BLUE LION
Let's Go Lion!
King of Hundred Beasts GoLion and Lion Force Voltron

When you got it wrong

Got it Right

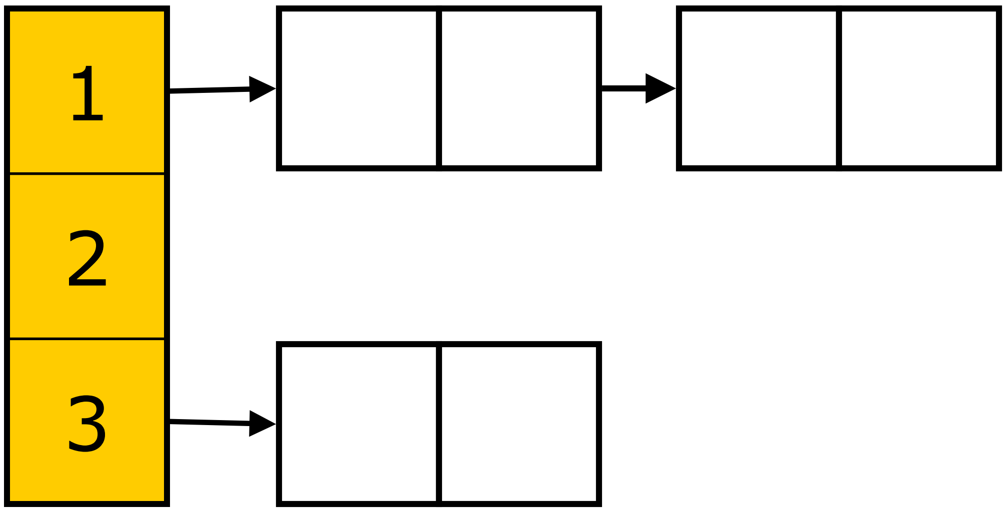


Basic Data Structures

- Array
- Linked List
- Trees

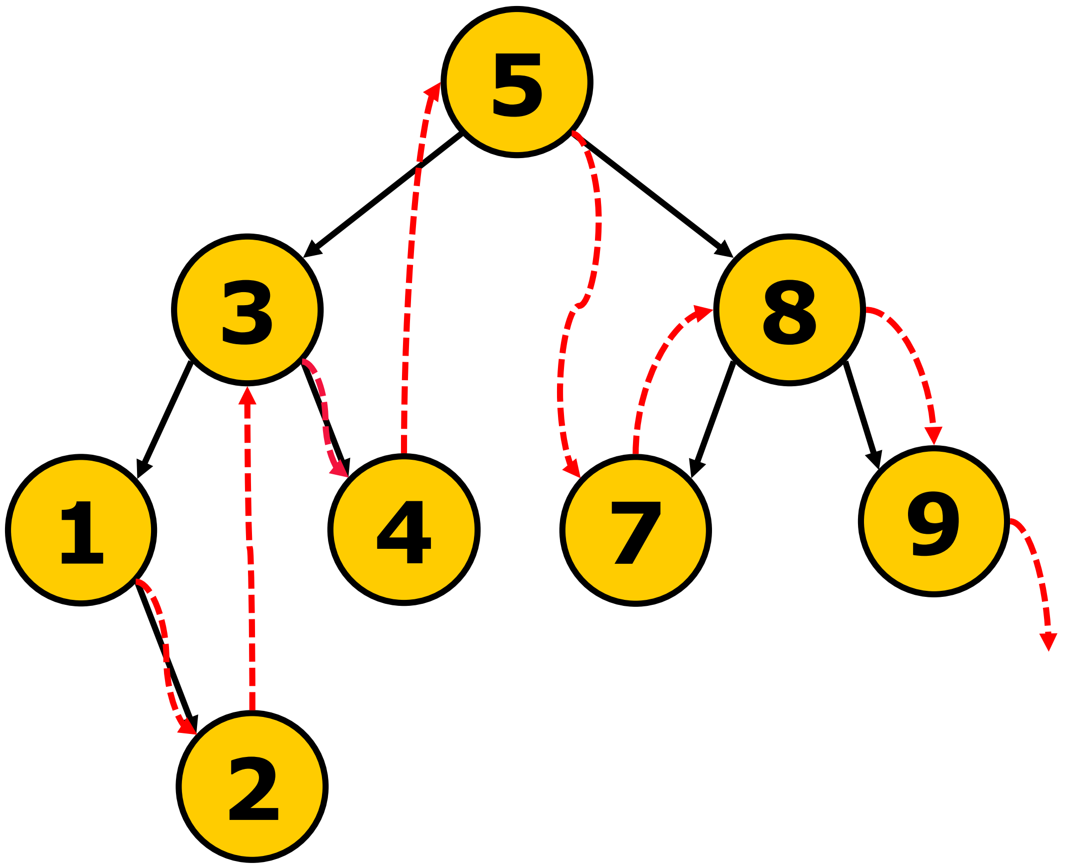
Mix-and-Match 1

- Array of Linked-List



Mix-and-Match 2

- Binary Search Tree + Linked-List



More Examples

- Need an ADT for
  - enqueue(item)
  - dequeue(item)
  - peek()
  - printInOrder()
    - Not “in-order” traversal
    - Just print them according to ascending or descending order

Use a Queue (Linked List)

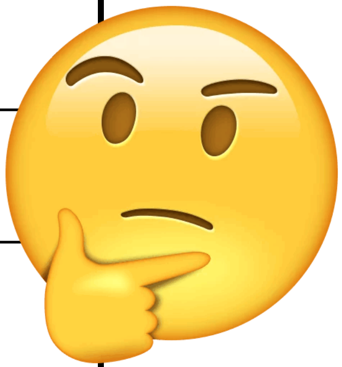
enqueue(item)	O(1)
dequeue()	O(1)
peek()	O(1)
printInOrder()	O(N log N)

🤔

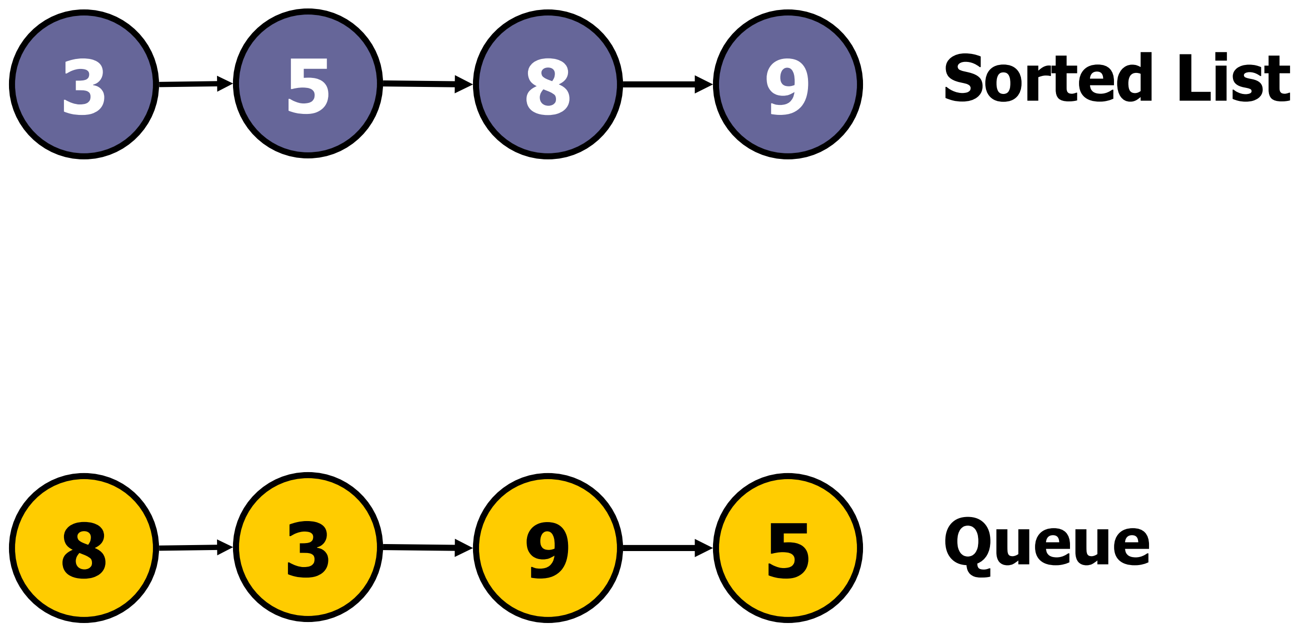
trivial because need to sort first

Use a Sorted Linked List

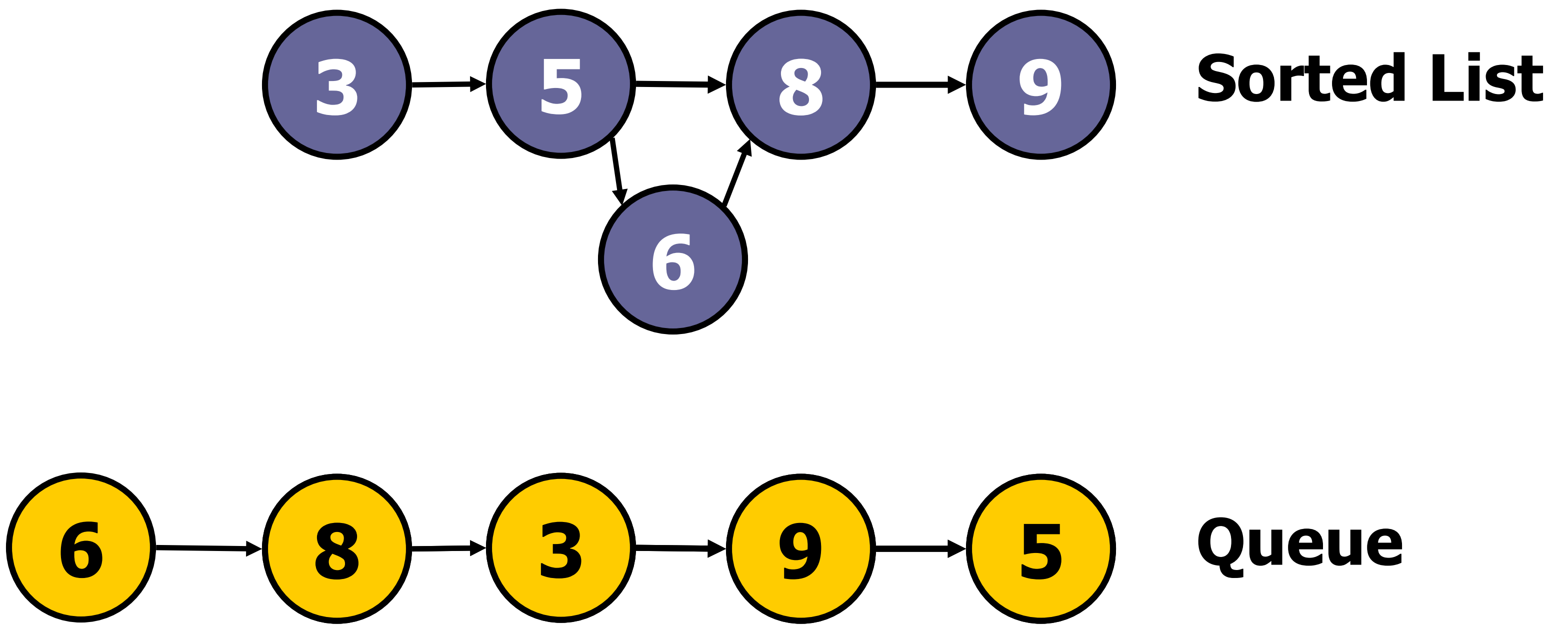
enqueue(item)	?
dequeue()	?
peek()	?
printInOrder()	O(N)



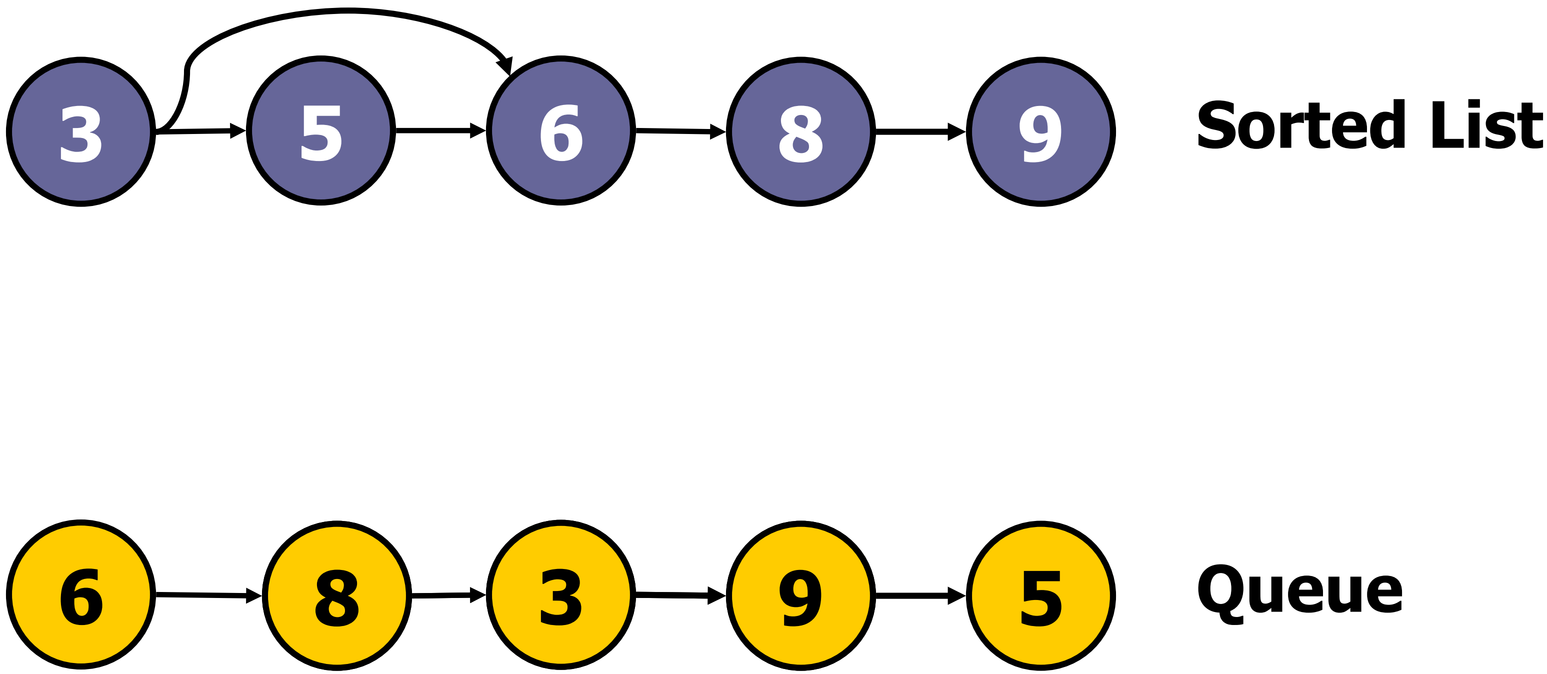
Use both



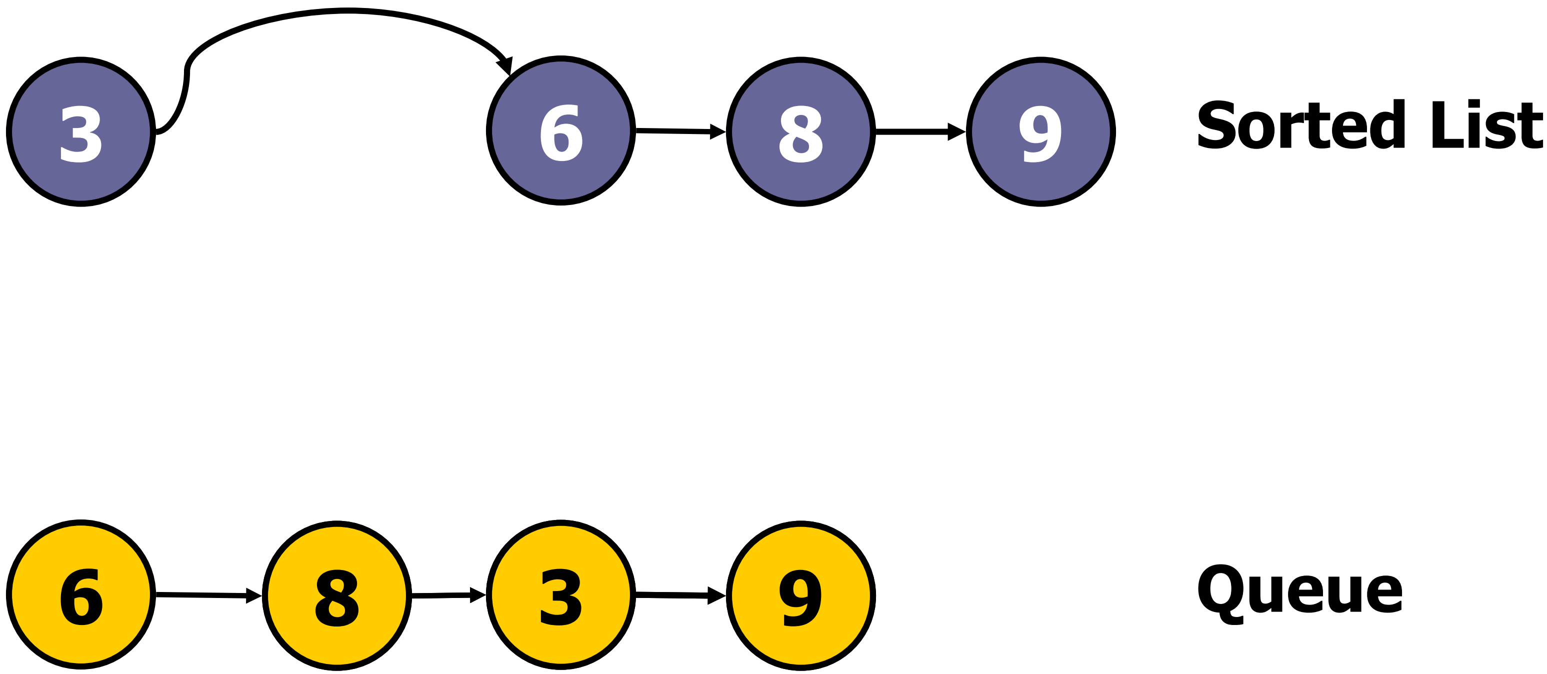
Enqueue(6)



Dequeue()

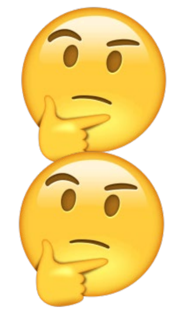


Dequeue()

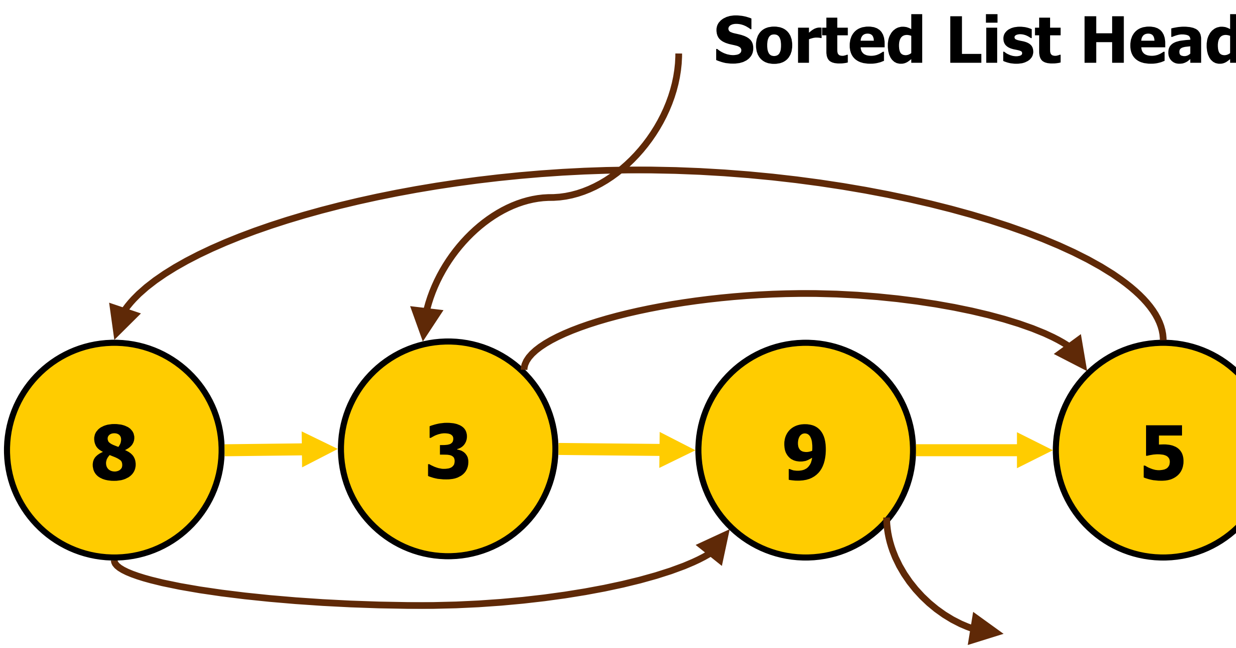


Use Queue and Linked List

enqueue(item)	O(N)
dequeue()	O(N)
peek()	O(1)
printInOrder()	O(N)



Improvement



Queue Pointer

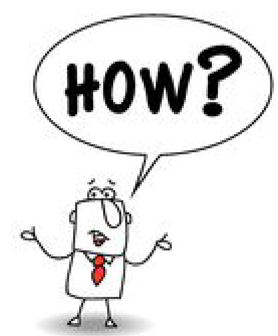
Sorted List Pointer

Combine Queue and Linked List

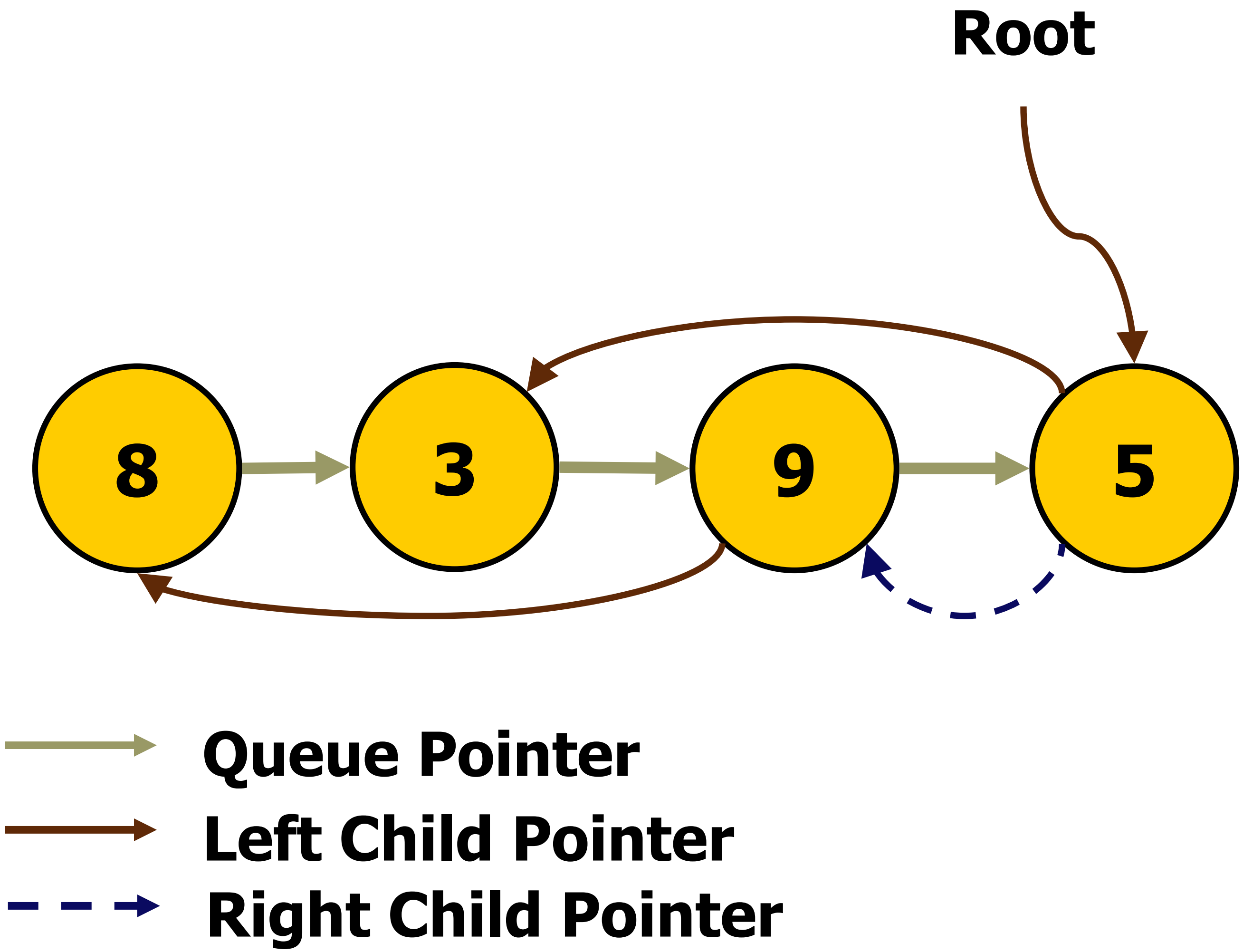
enqueue(item)	O(N)
dequeue()	**O(1)**
peek()	O(1)
printInOrder()	O(N)

Combine Queue and BST

enqueue(item)	**O(log N)**
dequeue()	O(1)
peek()	O(1)
printInOrder()	O(N)



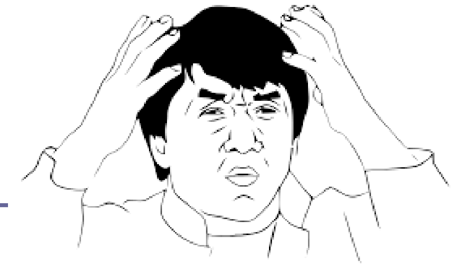
More Improvement

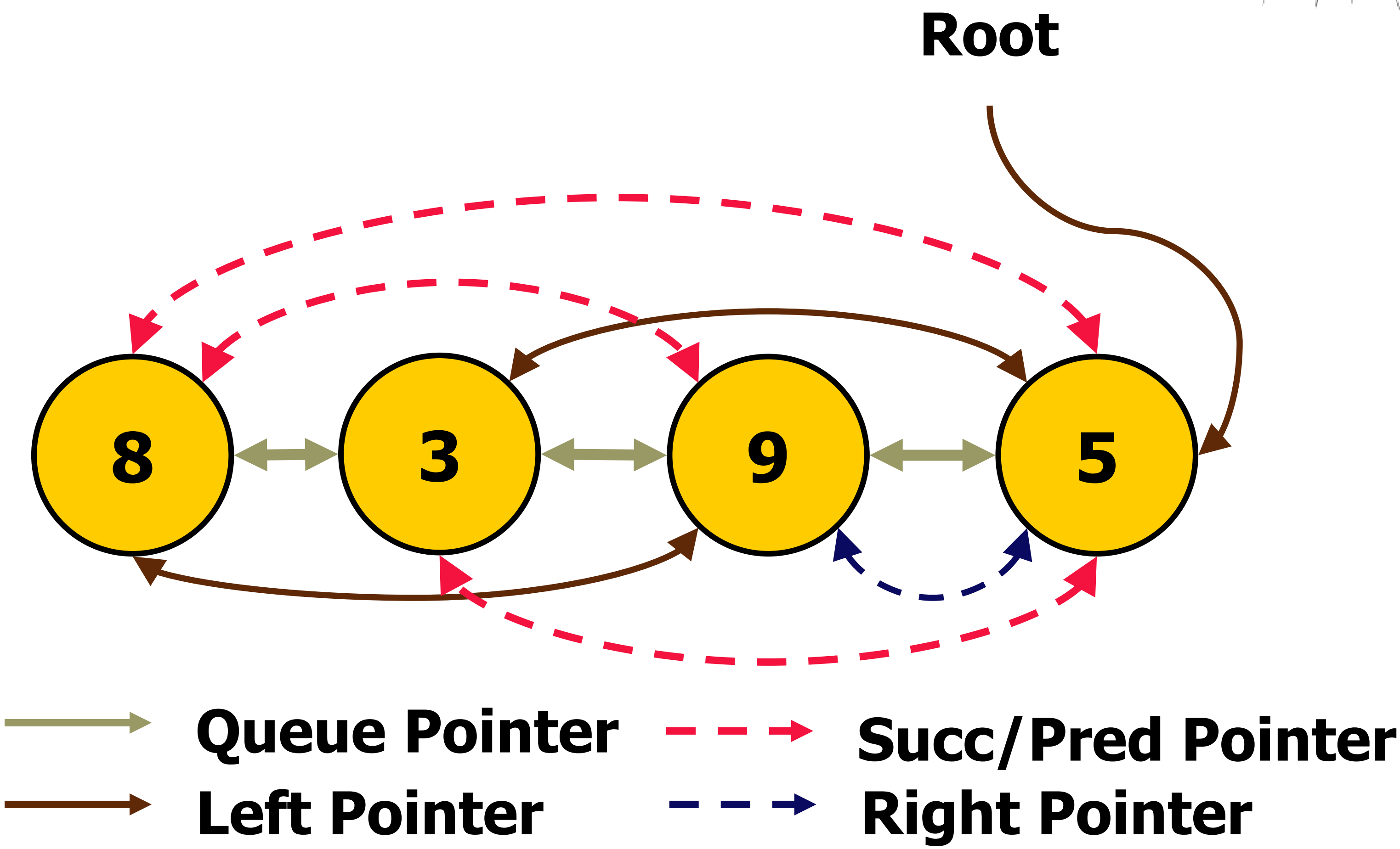


Combine Queue and BST

enqueue(item)	O(log N)
dequeue()	**O(1)**
peek()	O(1)
printInOrder()	O(N)

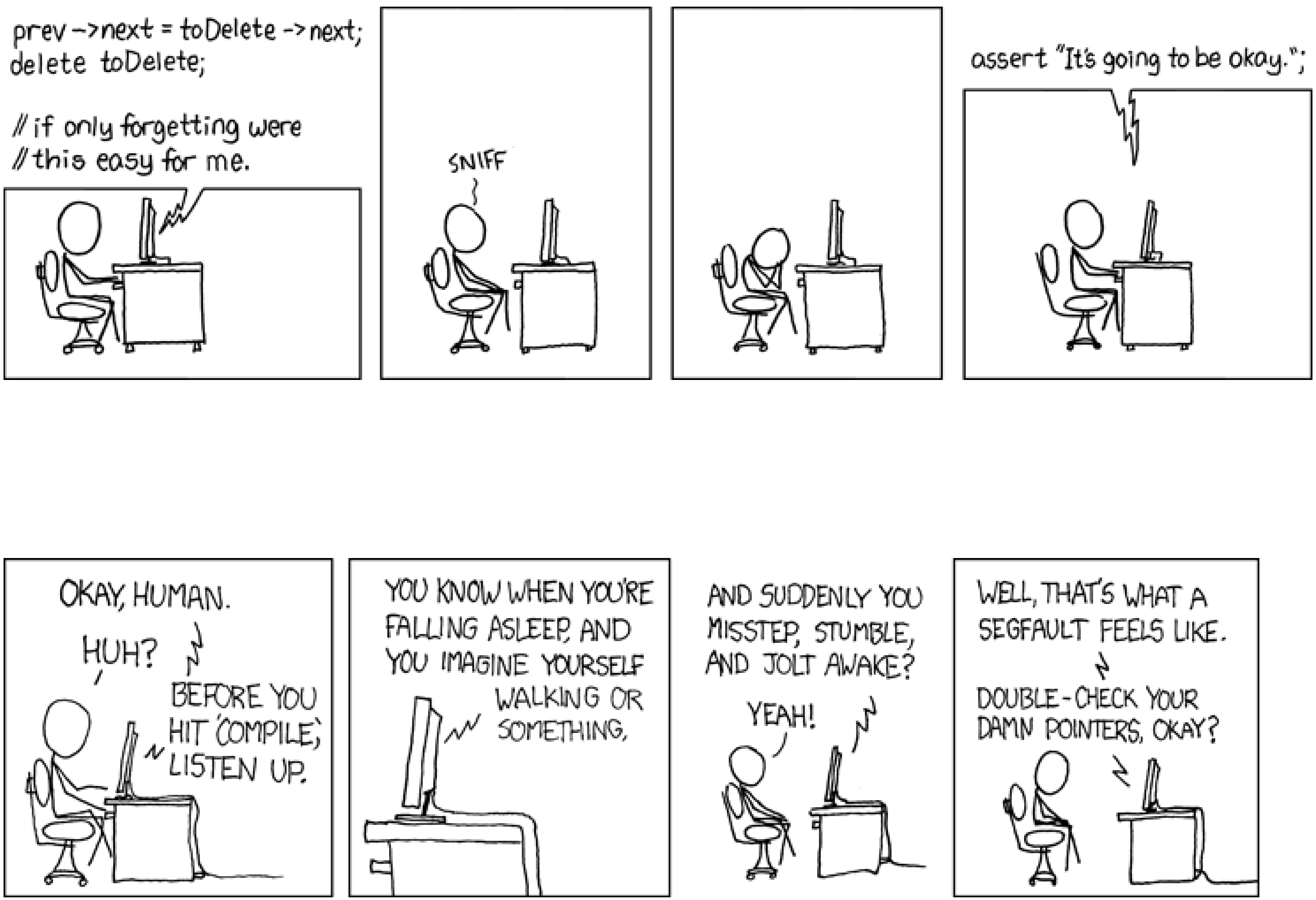
More Improvement





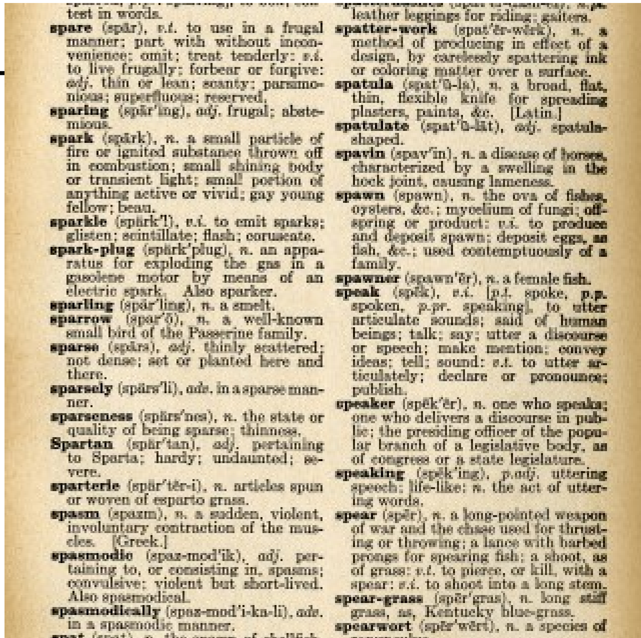
Mange the Pointers Well

Some Pointer Jokes (xkcd)



Every node is…

- a node in a Linked List, as well as,
- a node in a Tree, as well as,
- a node in a Queue, as well as….
- Graph…



test in words.

spare (spâr), *v.t.* to use in a frugal manner; part with without inconvenience; omit; treat tenderly: *v.i.* to live frugally; forbear or forgive: *adj.* thin or lean; scanty; parsimonious; superfluous; reserved.

sparing (spâr'ing), *adj.* frugal; abstemious.

spark (spärk), *n.* a small particle of fire or ignited substance thrown off in combustion; small shining body or transient light; small portion of anything active or vivid; gay young fellow; beau.

sparkle (spärk'l), *v.i.* to emit sparks; glisten; scintillate; flash; coruscate.

spark-plug (spärk'plug), *n.* an apparatus for exploding the gas in a gasolene motor by means of an electric spark. Also sparker.

sparling (spär'ling), *n.* a smelt.

sparrow (spar'ō), *n.* a well-known small bird of the Passerine family.

sparse (spärs), *adj.* thinly scattered; not dense; set or planted here and there.

sparsely (spärs'li), *adv.* in a sparse manner.

sparseness (spärs'nes), *n.* the state or quality of being sparse; thinness.

Spartan (spär'tan), *adj.* pertaining to Sparta; hardy; undaunted; severe.

sparterie (spär'tēr-i), *n.* articles spun or woven of esparto grass.

spasm (spazm), *n.* a sudden, violent, involuntary contraction of the muscles. [Greek.]

spasmodic (spaz-mod'ik), *adj.* pertaining to, or consisting in, spasms; convulsive; violent but short-lived. Also spasmodical.

spasmodically (spaz-mod'i-ka-li), *adv.* in a spasmodic manner.

spat (spat), *n.* the spawn of shellfish

leather leggings for riding; gaiters.

spatter-work (spat'ēr-wērk), *n.* a method of producing in effect of a design, by carelessly spattering ink or coloring matter over a surface.

spatula (spat'ū-la), *n.* a broad, flat, thin, flexible knife for spreading plasters, paints, &c. [Latin.]

spatulate (spat'ū-lāt), *adj.* spatula-shaped.

spavin (spav'in), *n.* a disease of horses, characterized by a swelling in the hock joint, causing lameness.

spawn (spawn), *n.* the ova of fishes, oysters, &c.; mycelium of fungi; offspring or product: *v.i.* to produce and deposit spawn; deposit eggs, as fish, &c.; used contemptuously of a family.

spawner (spawn'ēr), *n.* a female fish.

speak (spēk), *v.i.* [*p.t.* spoke, *p.p.* spoken, *p.pr.* speaking], to utter articulate sounds; said of human beings; talk; say; utter a discourse or speech; make mention; convey ideas; tell; sound: *v.t.* to utter articulately; declare or pronounce; publish.

speaker (spēk'ēr), *n.* one who speaks; one who delivers a discourse in public; the presiding officer of the popular branch of a legislative body, as of congress or a state legislature.

speaking (spēk'ing), *p.adj.* uttering speech; life-like: *n.* the act of uttering words.

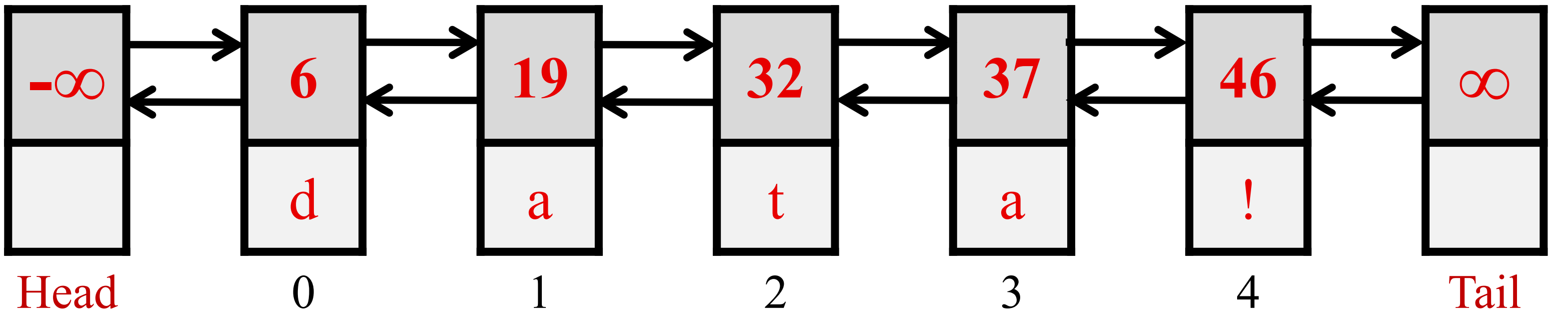
spear (spēr), *n.* a long-pointed weapon of war and the chase used for thrusting or throwing; a lance with barbed prongs for spearing fish; a shoot, as of grass: *v.t.* to pierce, or kill, with a spear: *v.i.* to shoot into a long stem.

spear-grass (spēr'gras), *n.* long stiff grass, as, Kentucky blue-grass.

spearwort (spēr'wērt), *n.* a species of

Implementing a dictionary, again…

Store keys in a sorted linked list:

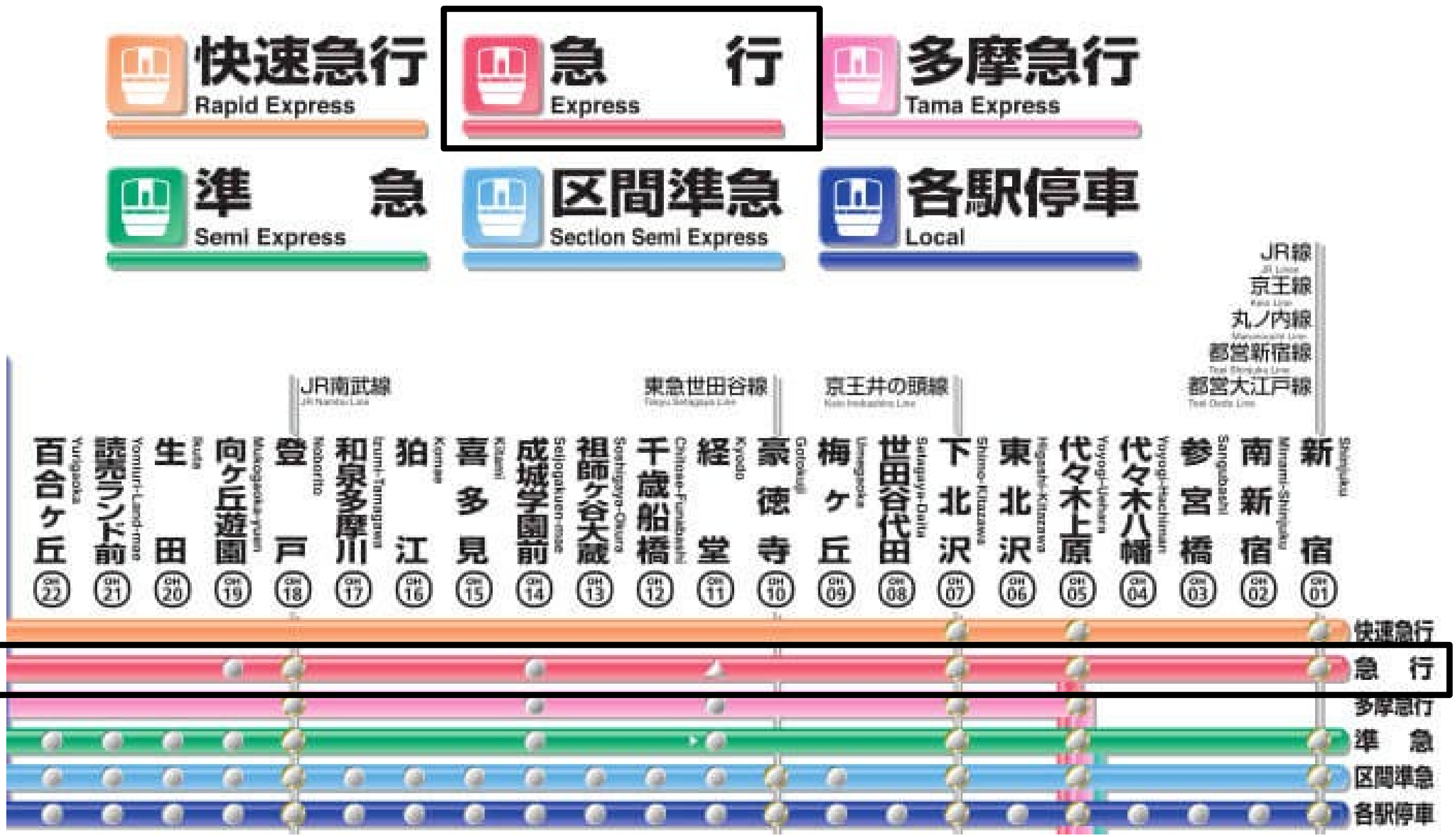


Time:

- **Search: O(n)**
- **Insert: O(n)**

Japan Rail System?

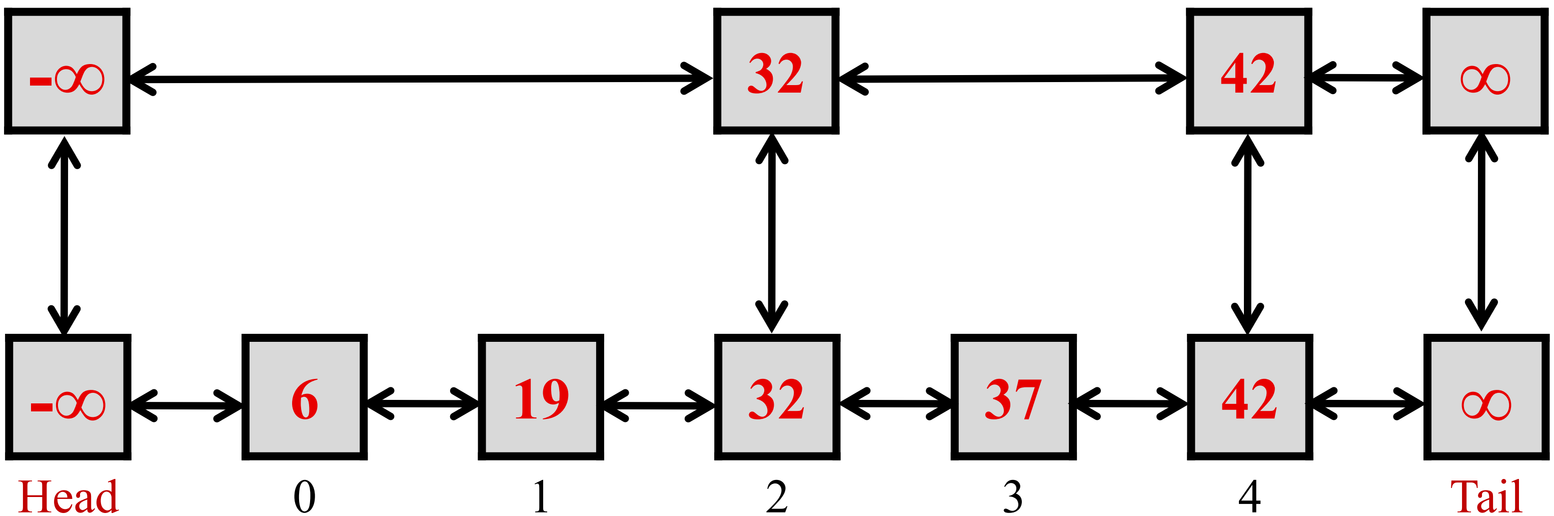
Japan Rail System?



What if…

What if we use two lists?

- Express train
- Local train



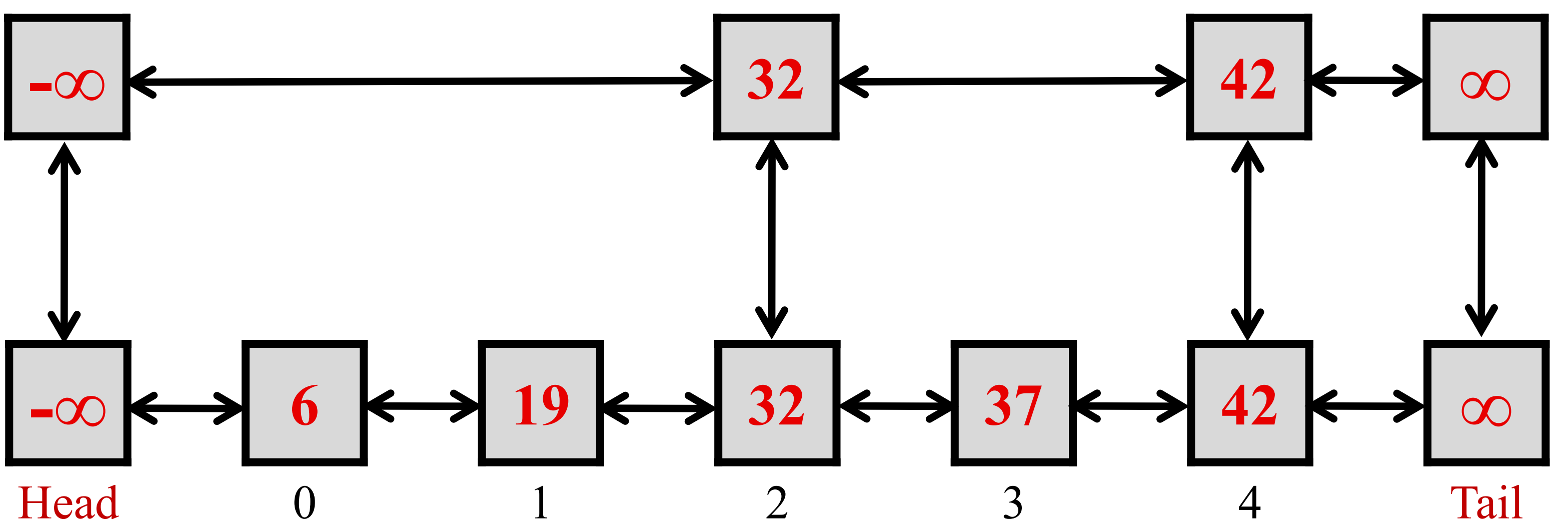
`search(37)` takes only 3 steps!

What if…

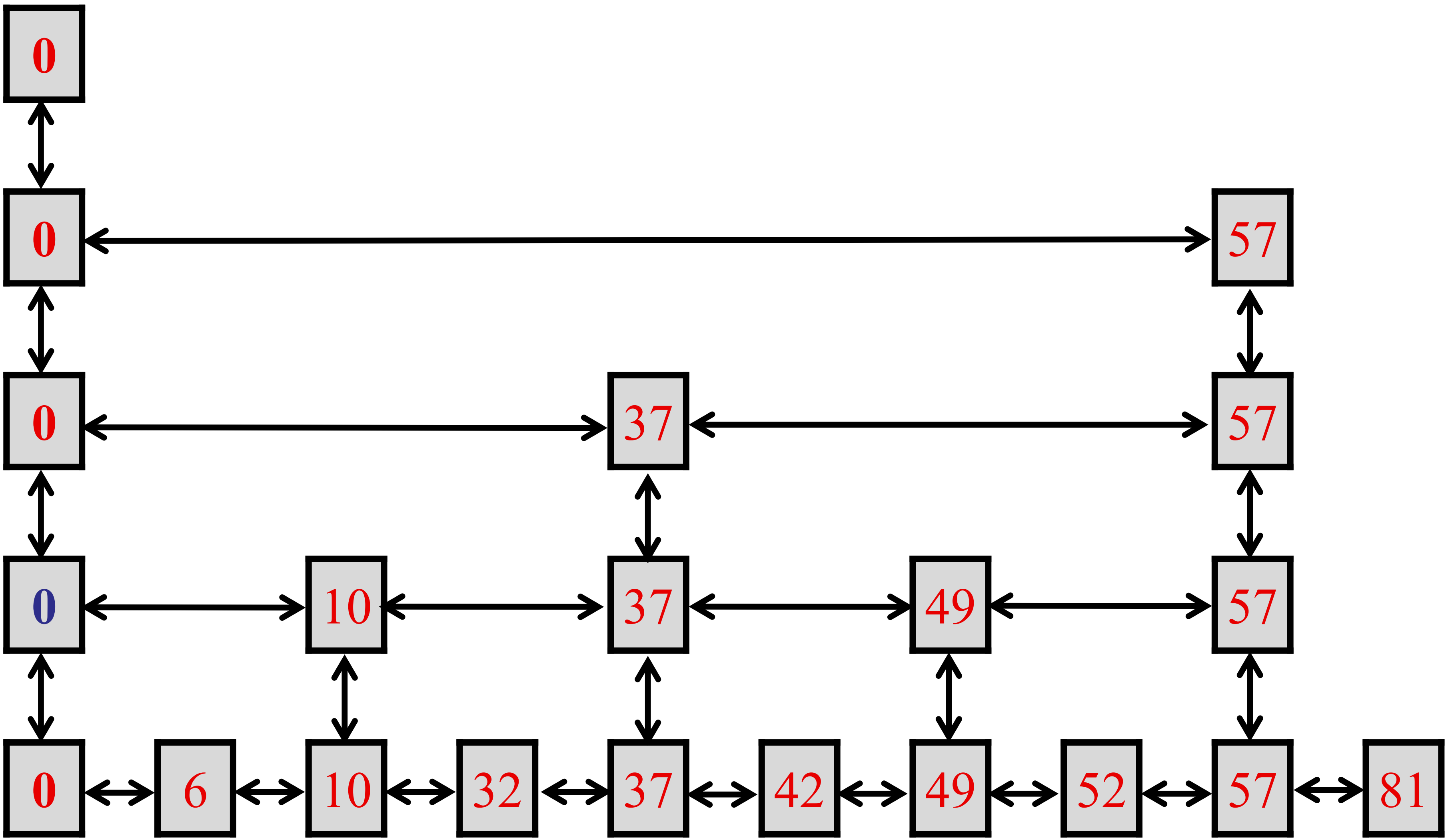
Calculation:

- If the "express" list skips 5 elements per "stop", then search takes at most:

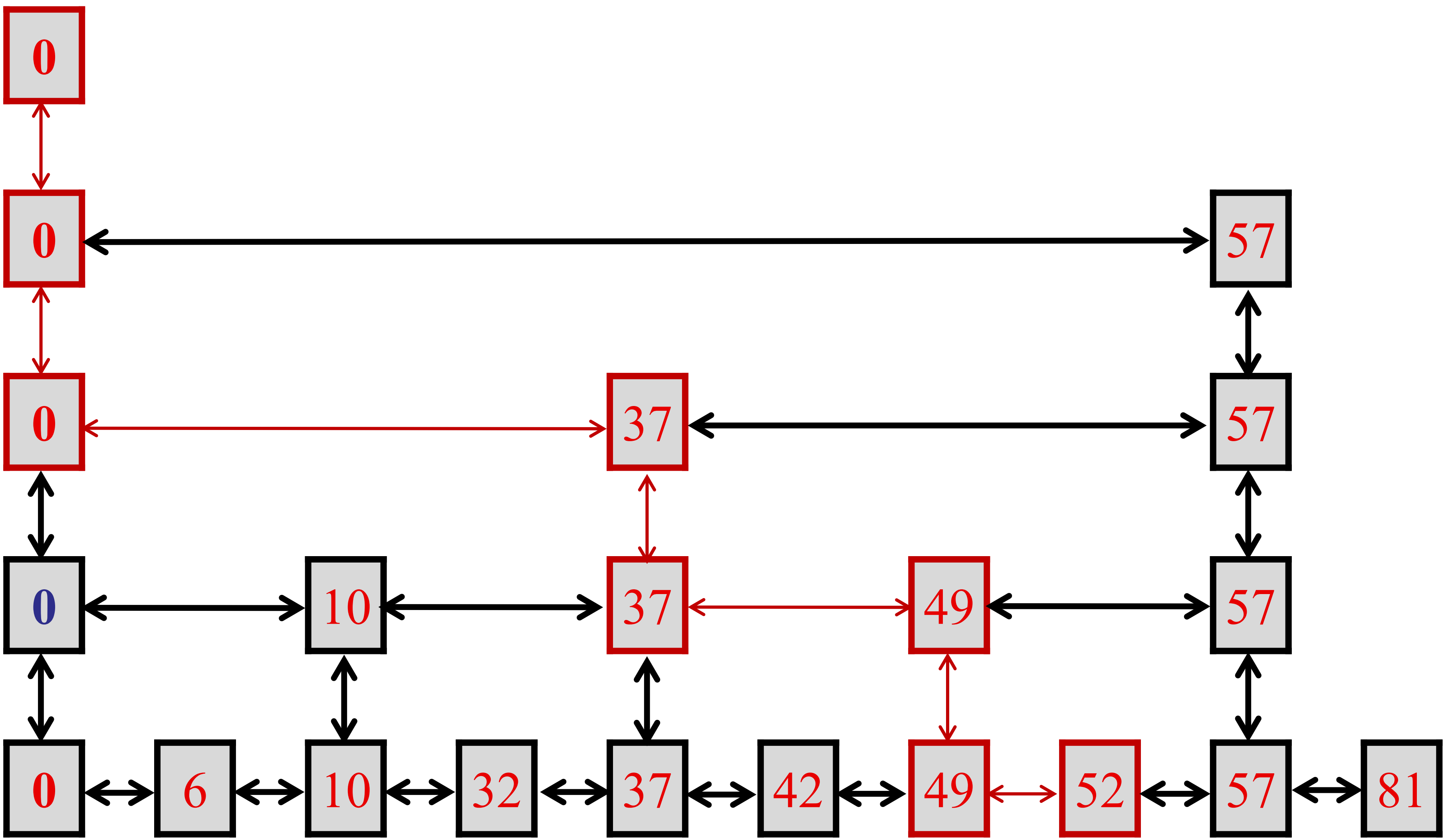
$$n/5 + 5 \text{ steps}$$



Another way to think about it…



Example: `search(52)`

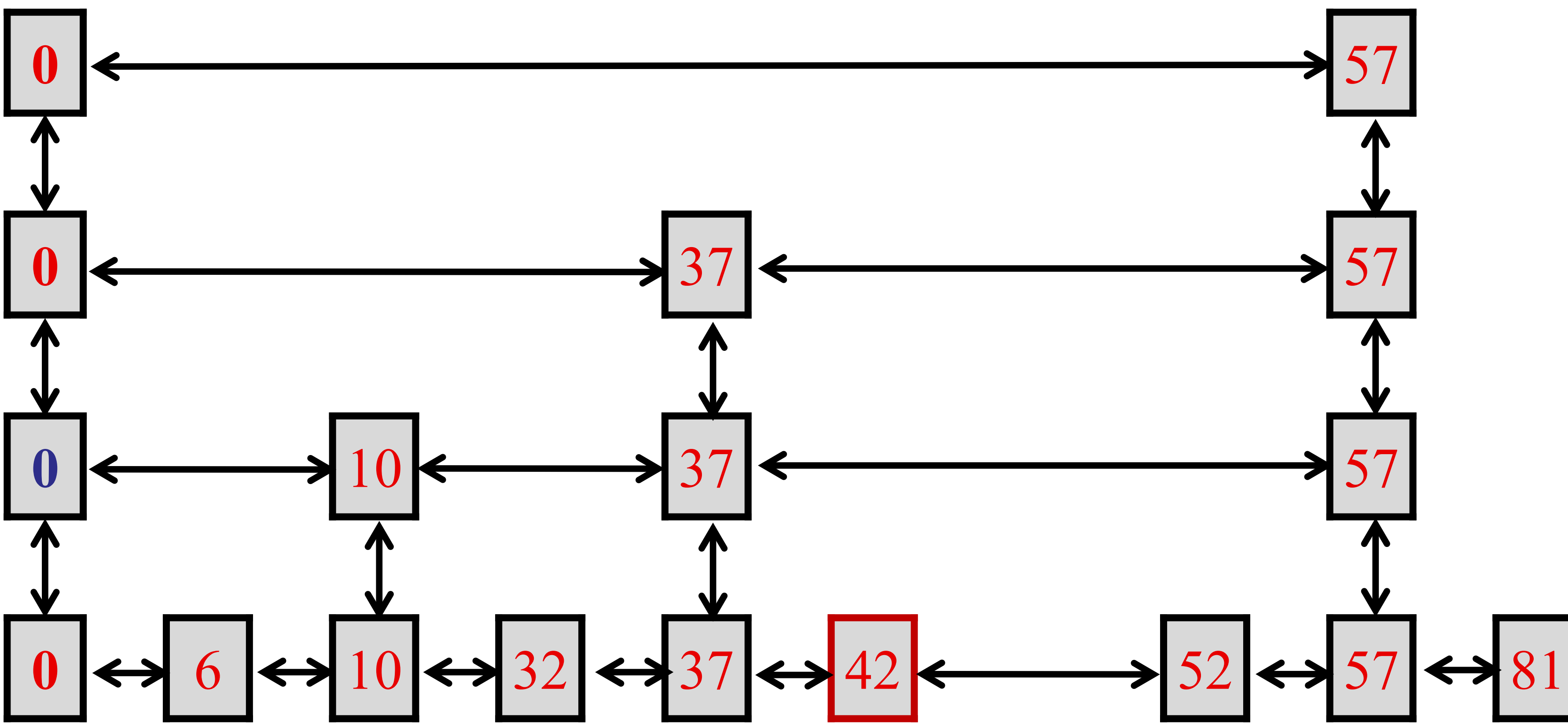


Insertions

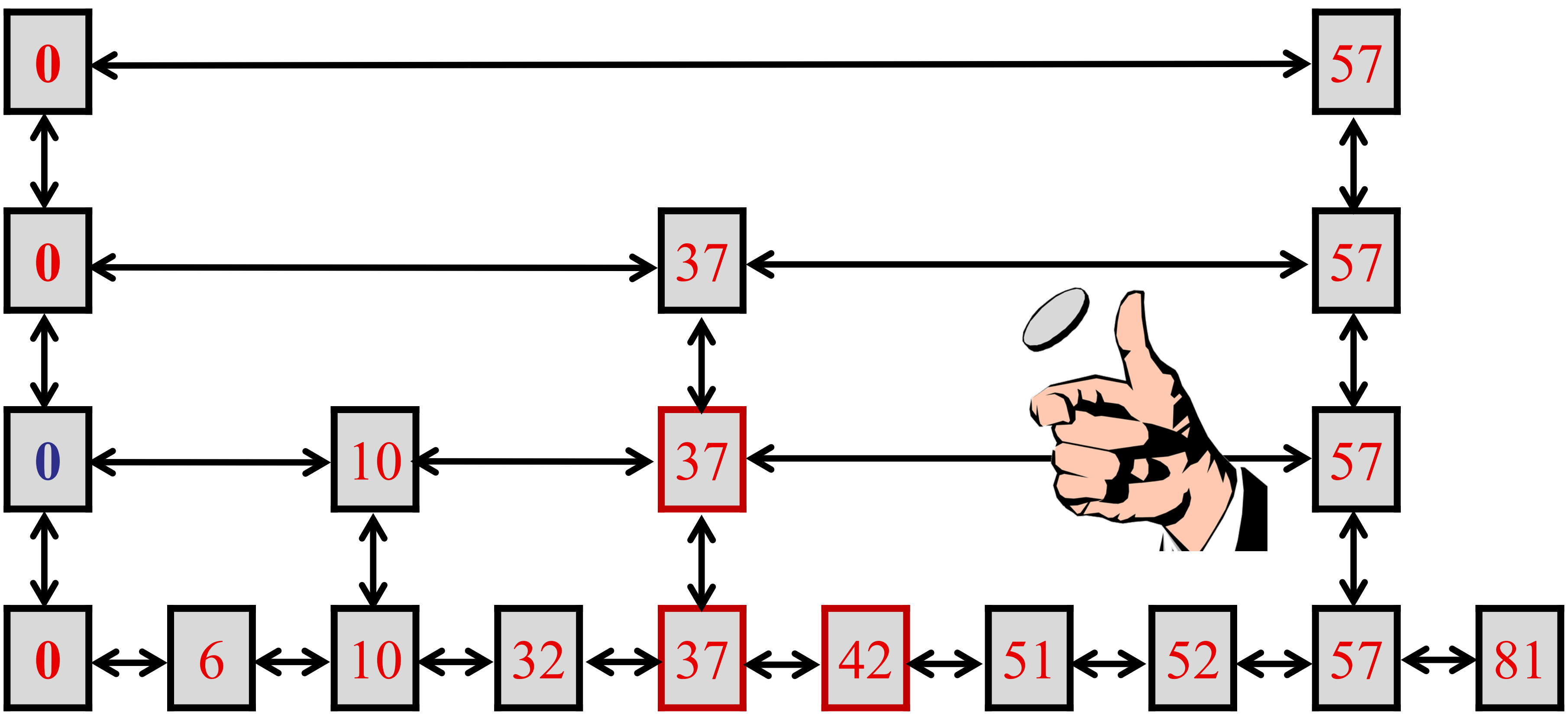
Key idea: flip a coin

```
k = 0;
while (!done) {
        Insert element into level k list.
        Flip a fair coin:
                with probability ½: done = true;
                with probability ½: k = k+1;
}
```

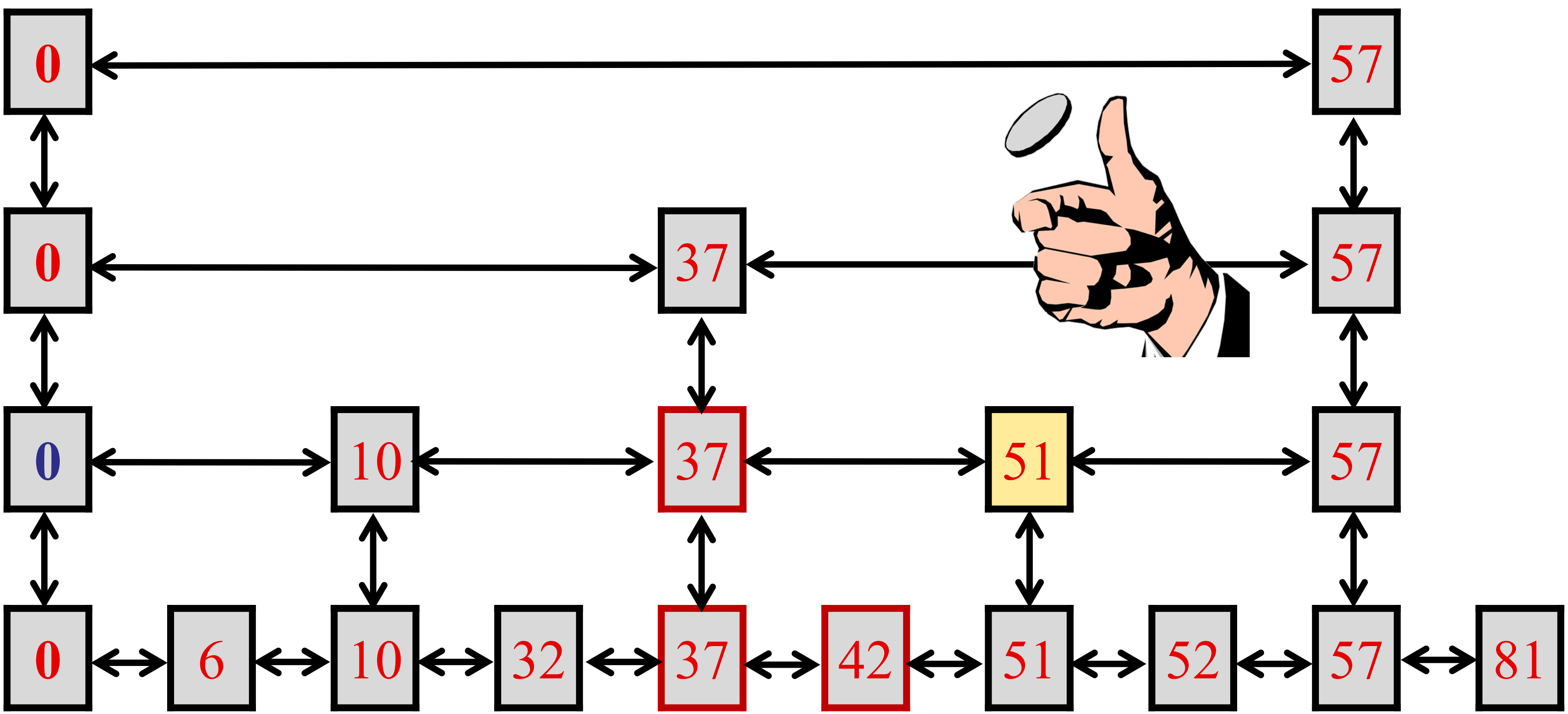
Example: `insert(51)`



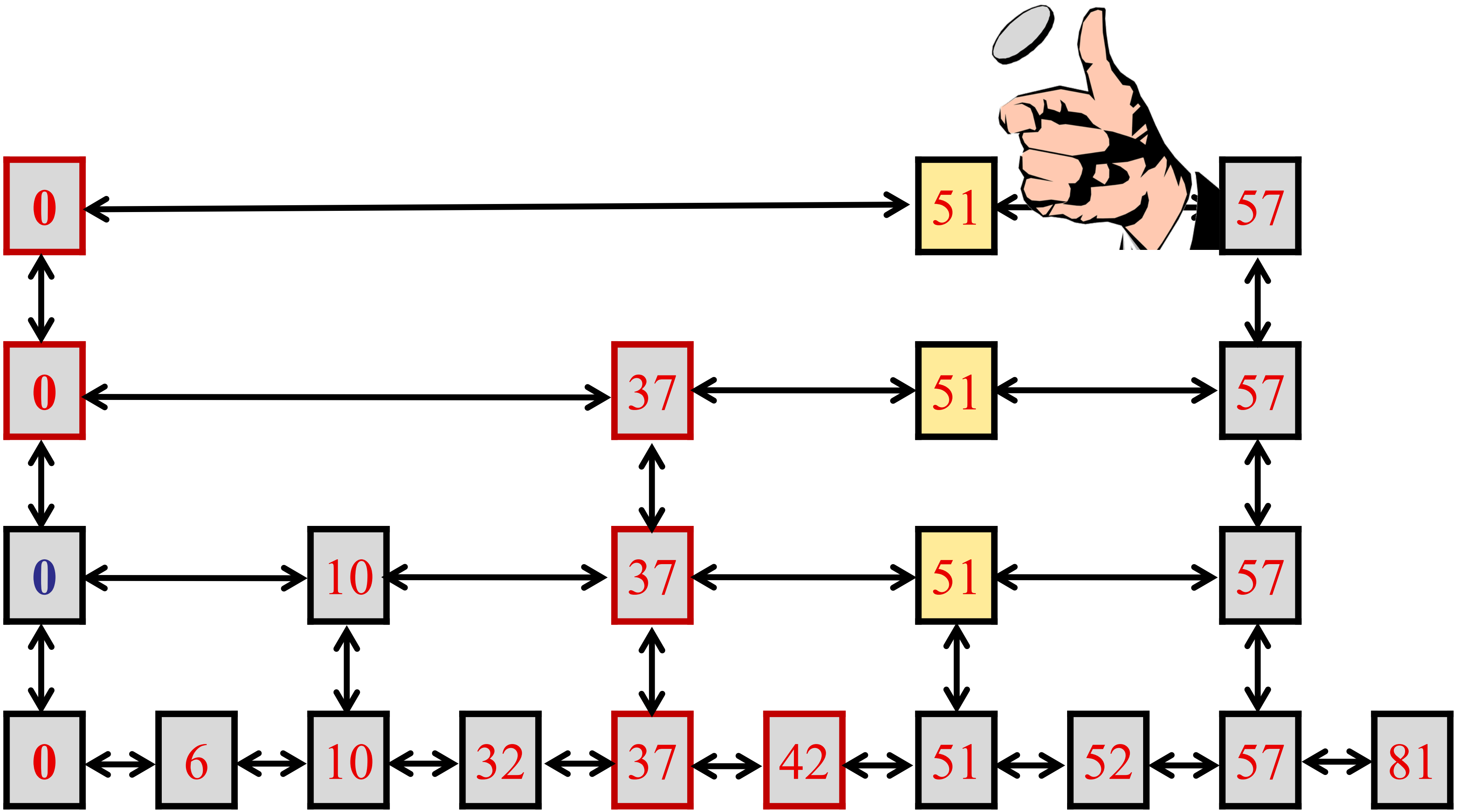
Example: `insert(51)`



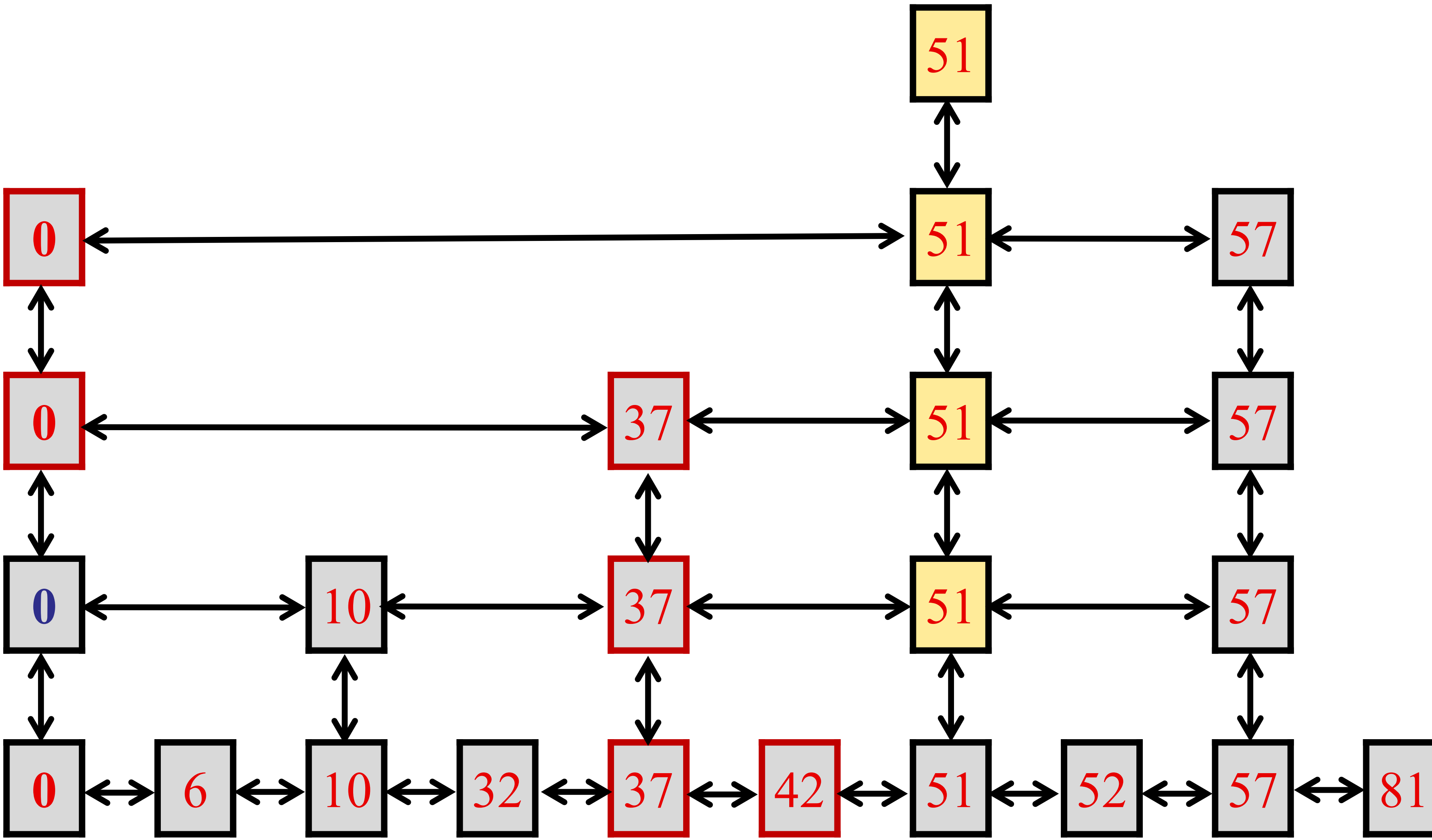
Example: `insert(51)`



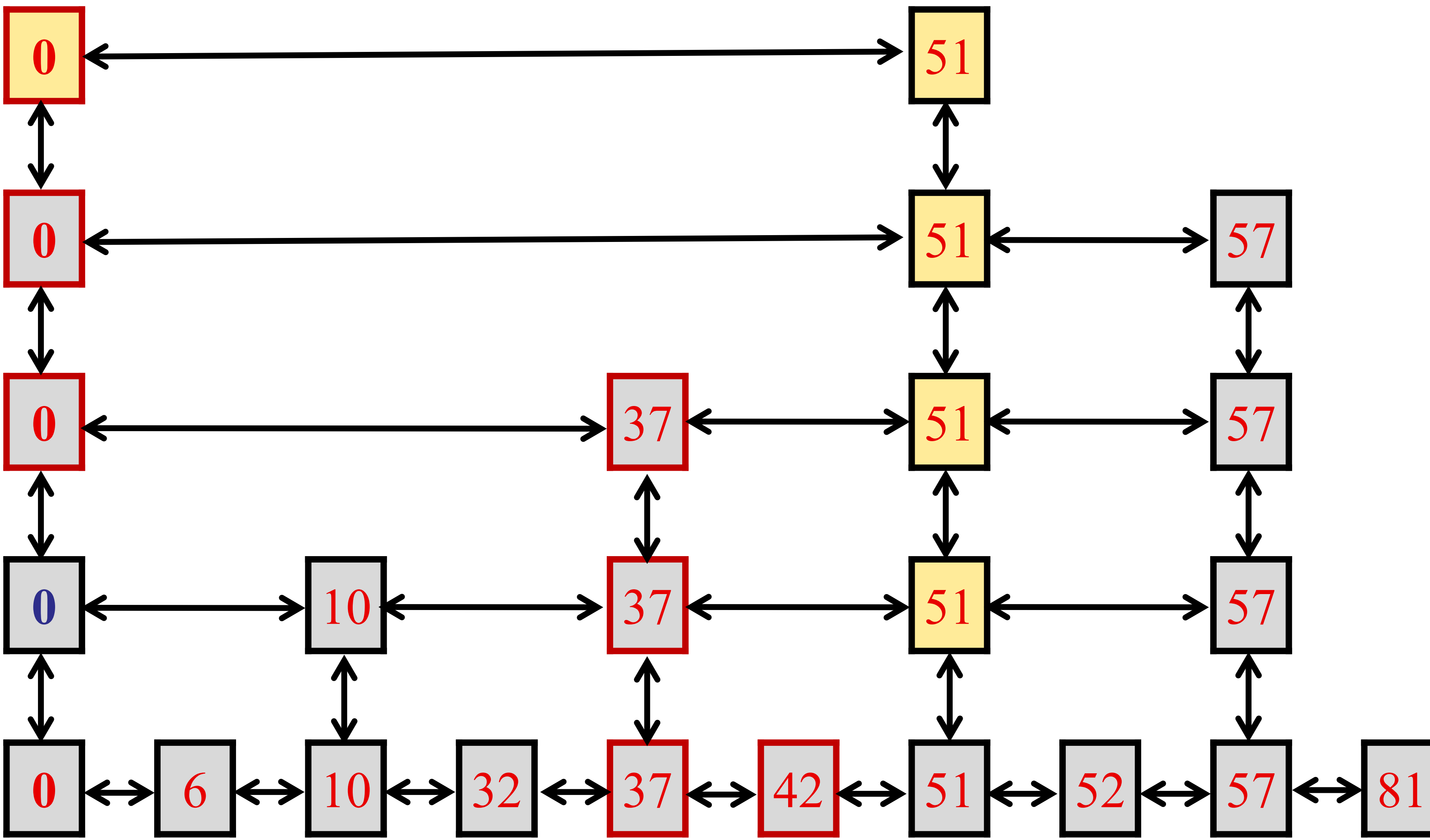
Example: `insert(51)`



Example: `insert(51)`



Example: `insert(51)`



SkipList Analysis

Claim: *In expectation*, after O(log n) coin flips, you get c log n heads.

Conclusion: Each search takes O(log n) steps in expectation.

Mix-and-Match

- Overlay/Merge/Contain multiple data structures
  - Possibly to get the best out of all

AVL can do what a Heap does (superset of Heap)
- Extract min/max, inc/dec keys in O(log n)

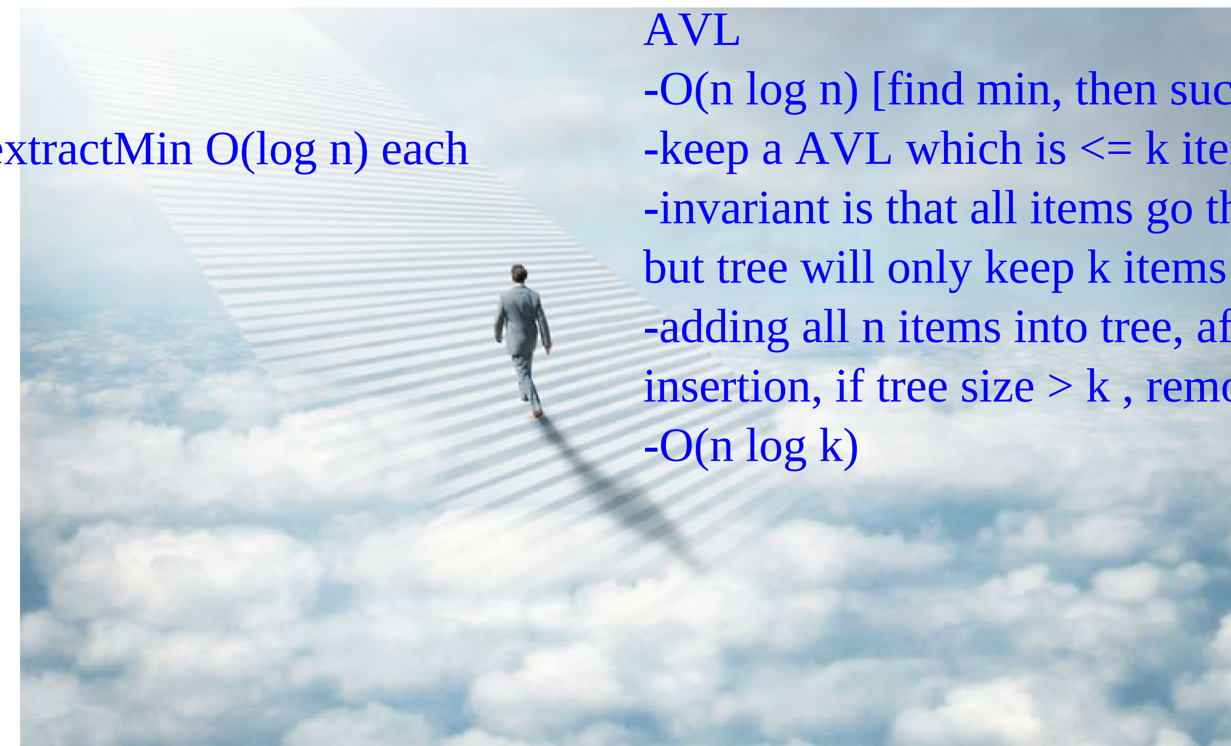
What did we learn?

+ Union Find

- C++, OOP
- Linked List, Stack, Queue
- Big O notations
- Sorting: BSIMQ
- AVL Trees, augmented trees
- Hashing
- Heaps
- Graphs: SSSP, TopoSort, MST
- Comp Geom

kSelect problem (given n items, we want to extract the smallest k items)

Life after CS2040C



Heap
-Heapify: O(n), extractMin O(log n) each
-O(n + k log n)

AVL
-O(n log n) [find min, then successor() k times]
-keep a AVL which is <= k items
-invariant is that all items go through the tree, but tree will only keep k items
-adding all n items into tree, after each insertion, if tree size > k , remove max
-O(n log k)

Operating System

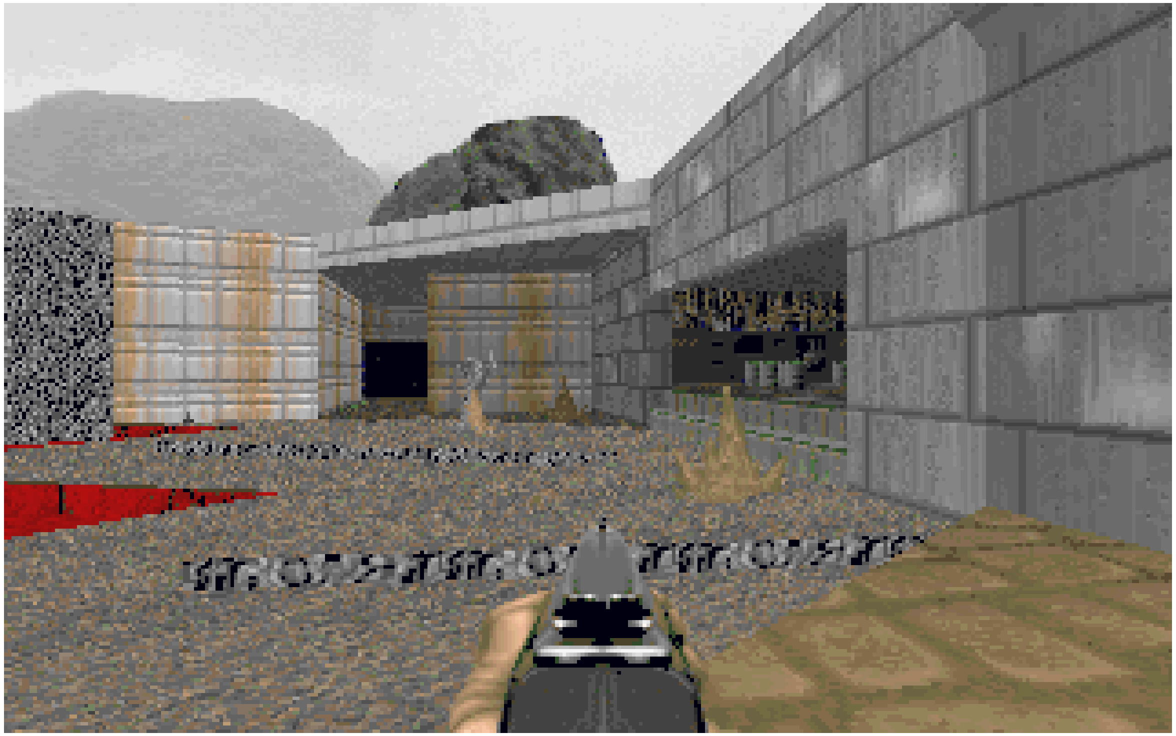
- how to map filename to location on disk?
  - tables, linked list
- how to manage processes?
  - priority queues

Computer Graphics

- Algorithms
  - Draw with occlusions (sorting)
  - Ray tracing (recursive!)

Computer Graphics

- Data structure: Binary Space Partition Tree



Compiler

- How to keep track of variable names, method names, class names?
  - Hash table
- How can computers “understand” programs?
  - Expression tree
  - Syntax tree

Artificial Intelligence

- how does computer play chess?
  - BFS on a tree/graph
- how to understand human language?
  - semantic network (a graph!)
- LISP/ML/Scheme/Prolog
  - plenty of lists and recursions!

Final Exam

Scope

- Everything
  - Except NP-hard problems

CS2040C

- Give an introduction to data structures and algorithms for **constructing efficient computer programs.**
- Emphasis is on **data abstraction** issues (through ADTs) in the code development.
- Emphasis on **efficient implementations** of chosen data structures and algorithms.

Objectives

- Include **stacks**, **queues**, **trees** (including BST, heap and AVL trees), **hash tables**, and **graphs**; together with their algorithms (tree and graph traversals, minimum spanning trees).
- Simple algorithmic paradigms, such as **search** algorithms and **divide-and-conquer** algorithms will be introduced.
- Elementary **analysis of algorithmic complexities** will also be taught.