Applied Algorithms

CSCI-B505 / INFO-I500

Lecture 4.

Review of Basic Data Structures - 2

M. Oguzhan Kulekci

- The skip-list ***probabilistic*** data structure
- Stacks
- Queues

The Problem

Search and Update on Sorted List of Items

- Given a **SORTED** list of items, support search(), insert(), and delete() operations *efficiently*.

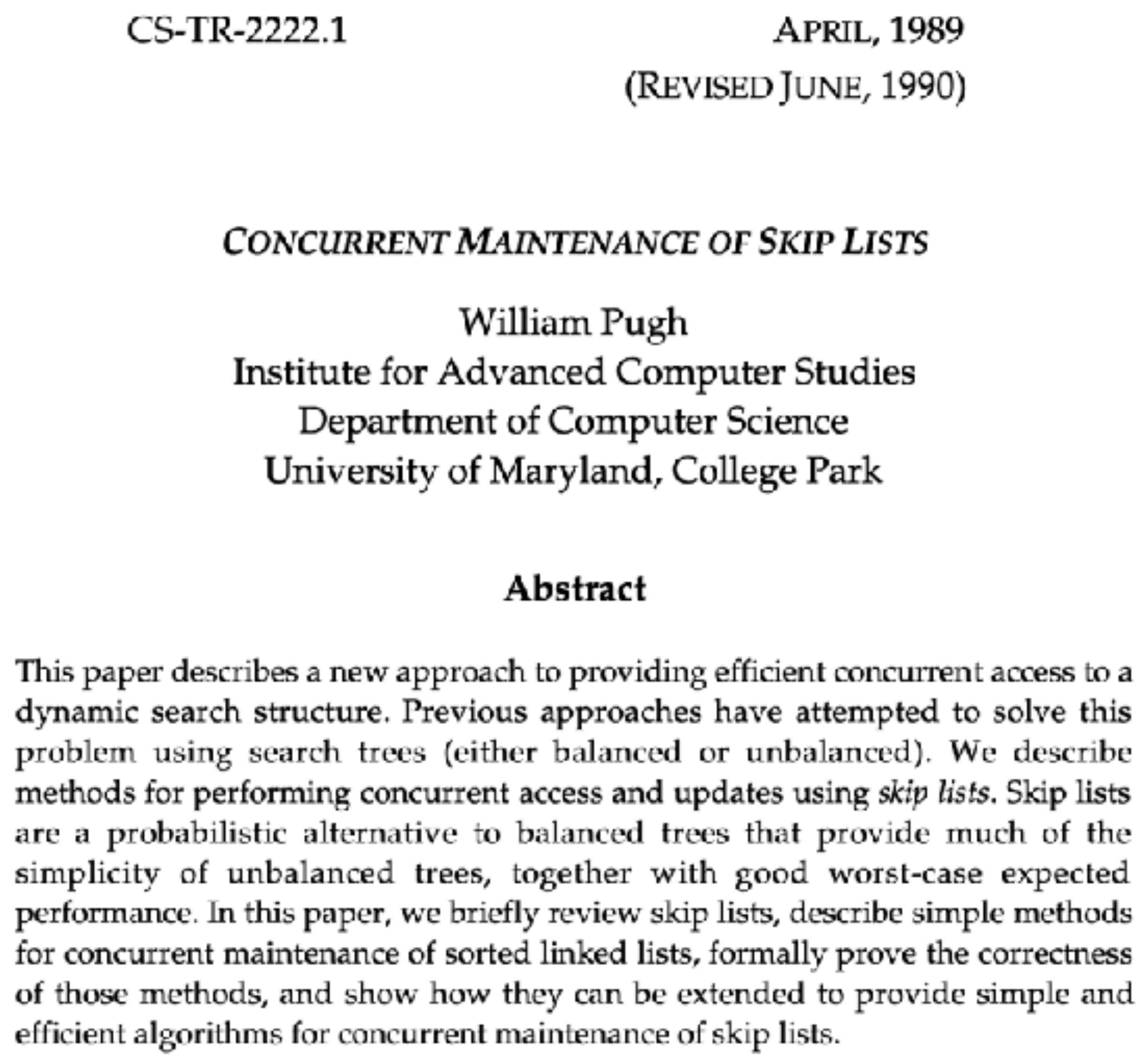
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	8	18	24	25	34	36	41	44	49	51	57	65	68	74	92

`insert(42) :`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	8	18	24	25	34	36	41	**42**	44	49	51	57	65	68	74	92

`delete(25) :`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	8	18	24	34	36	41	44	49	51	57	65	68	74	92



CS-TR-2222.1 APRIL, 1989

(REVISED JUNE, 1990)

CONCURRENT MAINTENANCE OF SKIP LISTS

William Pugh
Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland, College Park

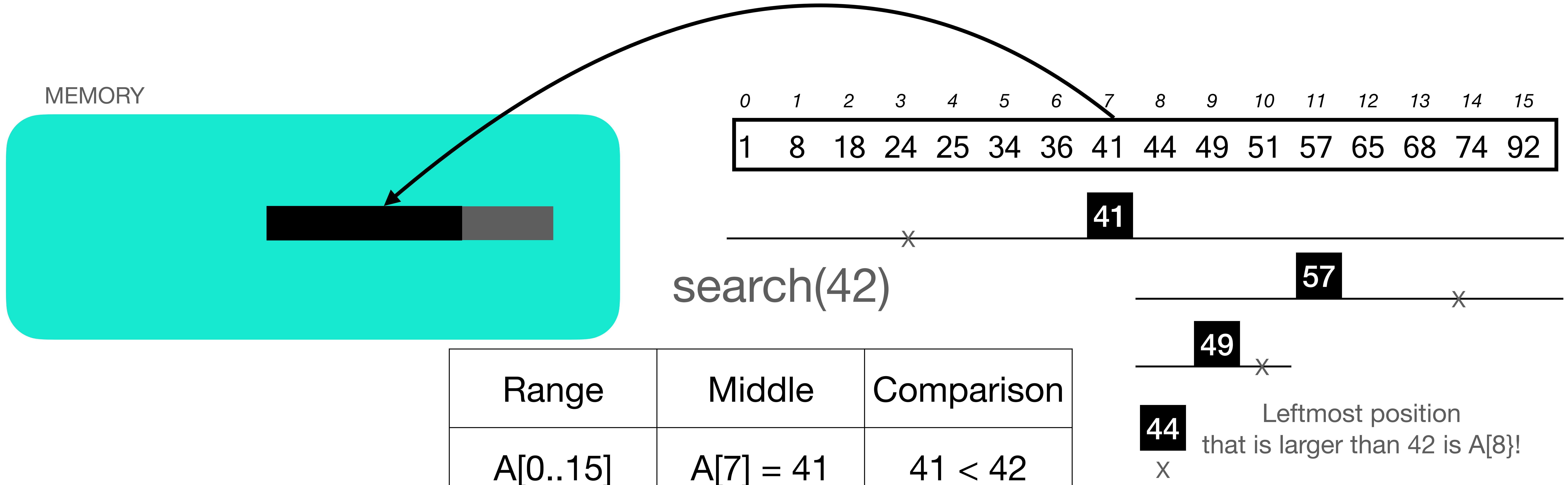
Abstract

This paper describes a new approach to providing efficient concurrent access to a dynamic search structure. Previous approaches have attempted to solve this problem using search trees (either balanced or unbalanced). We describe methods for performing concurrent access and updates using *skip lists*. Skip lists are a probabilistic alternative to balanced trees that provide much of the simplicity of unbalanced trees, together with good worst-case expected performance. In this paper, we briefly review skip lists, describe simple methods for concurrent maintenance of sorted linked lists, formally prove the correctness of those methods, and show how they can be extended to provide simple and efficient algorithms for concurrent maintenance of skip lists.

What is the significance from a practical view ?

If we use an array …

Search is efficient ! O(log n)

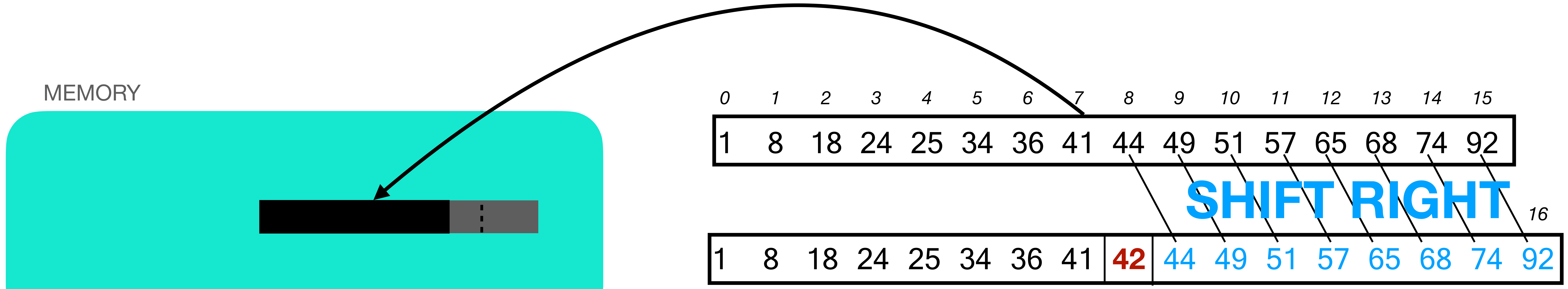


Range	Middle	Comparison
A[0..15]	A[7] = 41	41 < 42
A[8..15]	A[11] = 57	57 > 42
A[8..10]	A[9] = 49	49 > 42
A[8..8]	A[8] = 44	44 > 42

O(log n) - time binary search

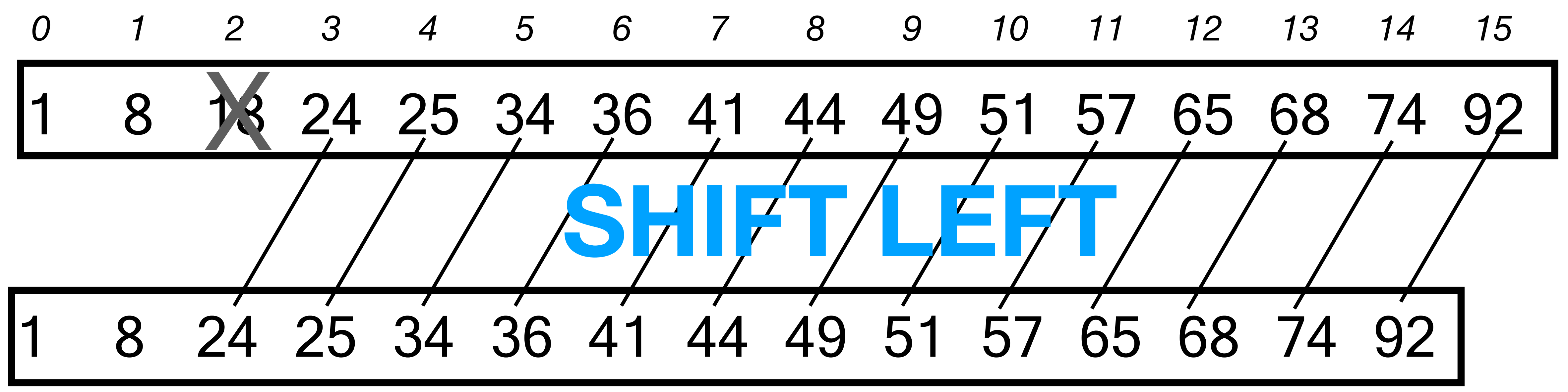
If we use an array …

But, updates are messy ! O(n)



insert(42) : First find the right position via search(42), and then, perform insertion

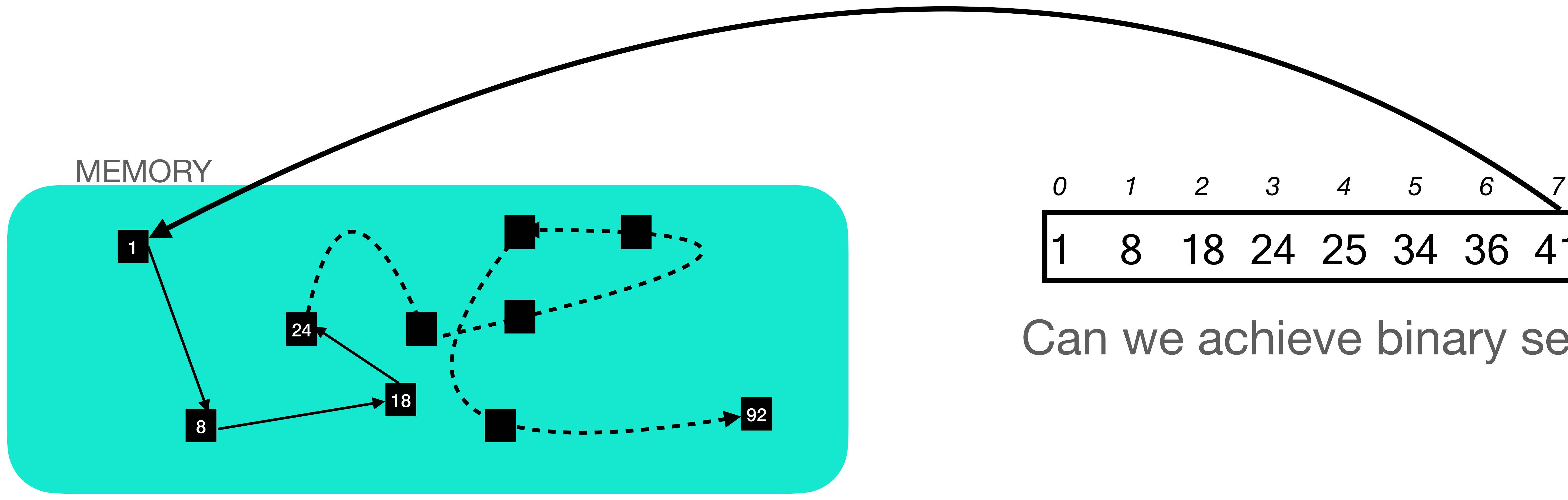
In the !**worst case!** n shift operations …



delete(18)

If we use a linked list …

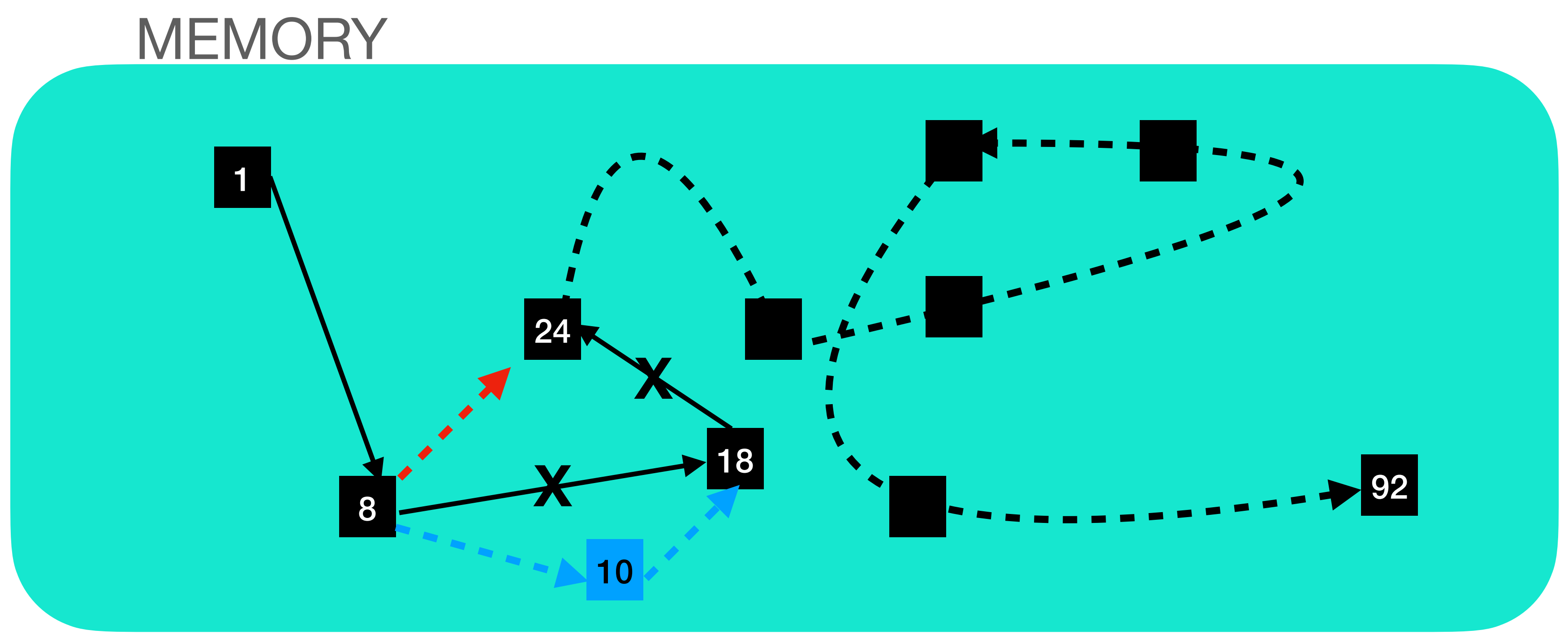
Updates are handled better, but search ?



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	8	18	24	25	34	36	41	44	49	51	57	65	68	74	92

Can we achieve binary search on linked list ? Why ?

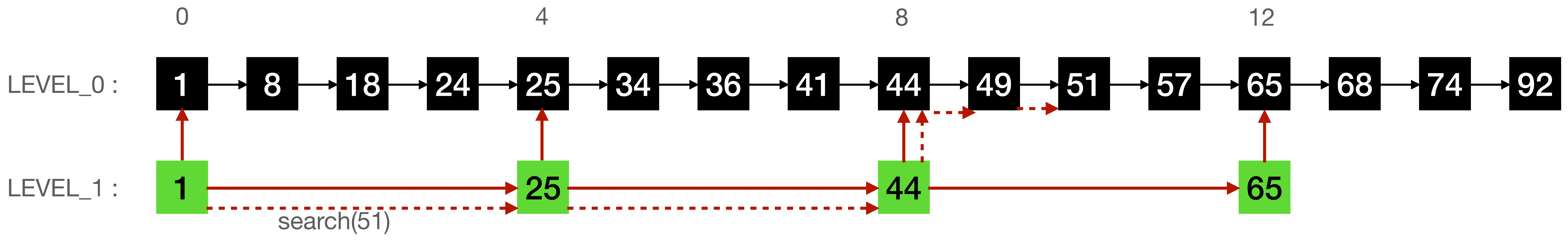
- Linear-time search ☹️
- Constant-time update **(when position is given!)** 👍

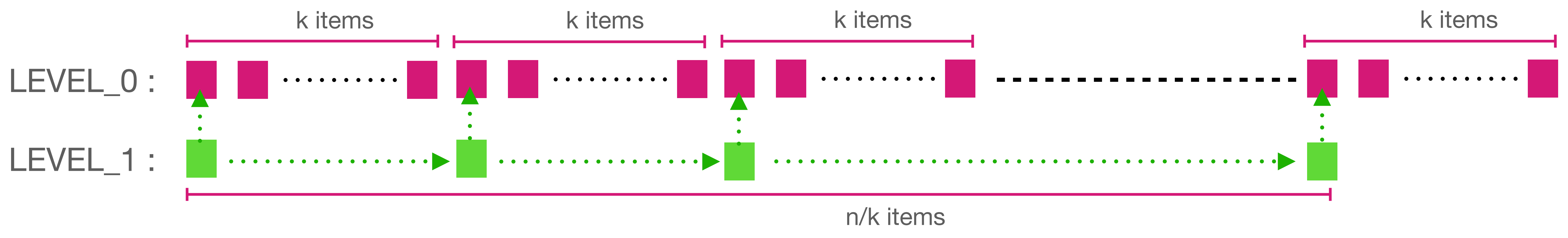


How about insertion or deletion ? insert(10), delete(18) !

How can we improve search on linked list ?

What if we introduce a second level …





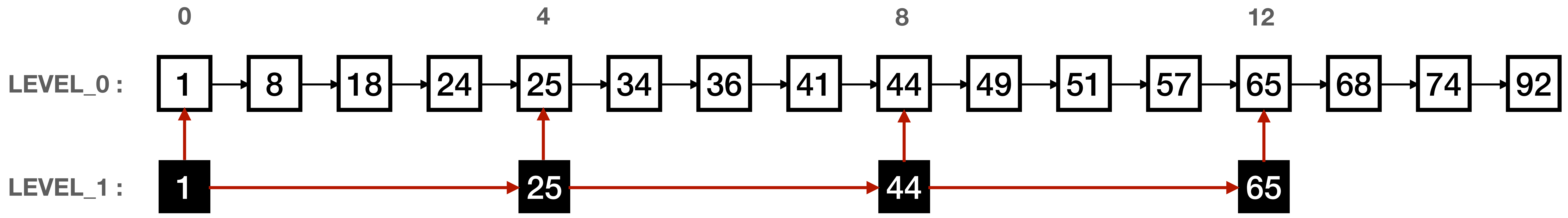
Sampling frequency is k in level_0.
During a search() operation:
1) At most how many nodes are visited in level_1?
2) At most how many nodes are visited in level_0?

Search requires **at most** (n/k + k) comparisons!
If k = $\sqrt{n}$, then O($\sqrt{n}$)-time.

Time-memory trade off: $O(\sqrt{n})$-time is achieved with the cost of using $O(\sqrt{n})$ more space !

Does increasing levels help to improve theoretical/practical performance?

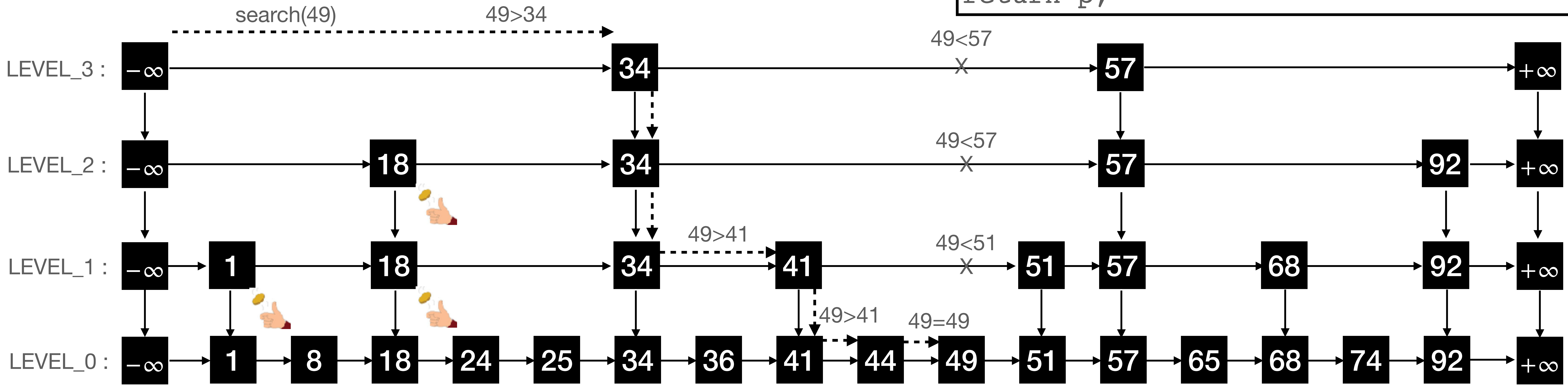
A different sampling strategy is required to reach O(log n) !


0
4
8
12
LEVEL_0 :
1
8
18
24
25
34
36
41
44
49
51
57
65
68
74
92
LEVEL_1 :
1
25
44
65

Skip List

Randomized sampling …

```
search(k):
p = head position of the top list;
while (p.next() < k) p = p.next();
while (p.down() ≠ NULL)
      p = p.down();
      while (p.next() < k) p = p.next();
return p;
```



Insert(k): Find position with search(k), then perform insertion into the ground level. Keep inserting on higher levels according to coin toss. insert(70) ?

Delete(k): Find position with search(k), then remove the corresponding nodes from all lists. delete(18) ?

Space complexity: $n\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} \cdots + \frac{1}{2^h}\right) < 2n$, $O(n)$**—space**

Skip List

How many levels ?

- We are carrying an element from level i to $(i+1)$ with probability $(1/2)$.
- The probability that a node appears at level k, which means it appears on all previous levels $0,1,2\ldots,(k-1)$as well, is $(1/2)^k$.
- The expected number of nodes at level k is $n \cdot (1/2)^k$
- If we solve for $n/2^k = 1$, then $k = \log n$.
- The probability P_k that there will be at least one node at level k is $P_k \leq n/2^k$
- Therefore, if $k = c \cdot \log n$ for some constant c value, then $P_k \leq 1/2^{c-1}$.

Skip List

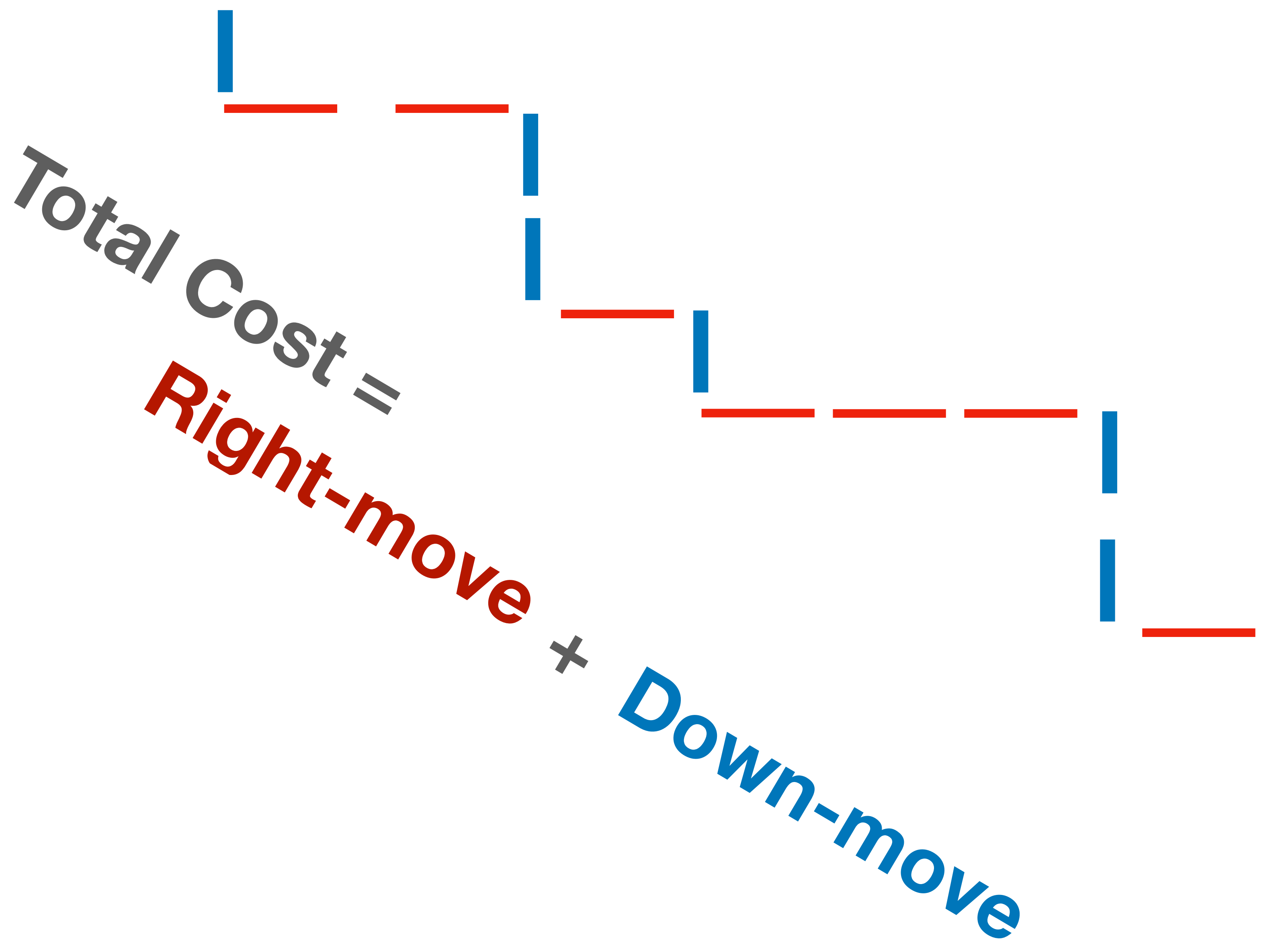
Time complexity

Assume we have $c \cdot \log n$ levels ?

Then, the number of downward steps is $h = O(\log n)$.

What is the expected number of consecutive nodes that appear at L_i but not in L_{i+1} ?

A node is carried upper level with 1/2 probability.
Thus, in any level , expected number of consecutive nodes not carried upper is 2, which is constant.



How is this question related with counting the number of right-moves ?

1. A node at level i can be visited if it is not sampled in level (i+1).
2. Therefore, the horizontal scan at any level is expected to be around the number of consecutive nodes that are sampled in a level, but not carried to the upper level.

```
search(k):
p = head position of the top list;
while (p.next() $\leq$ k) p = p.next();
while (p.down() $\neq$ NULL)
      p = p.down();
      while (p.next() $\leq$ k) p = p.next(); → $O(1)$
return p;
```

$O(\log n)$

Summary and Discussions…

Maintaining a sorted list of items via …

	Search	Update
Array	$O(\log n)$	$O(n)$
Linked List	$O(n)$	$O(1)*$
Skip List	$O(\log n)$	$O(\log n)$

$\}$ Worst-case

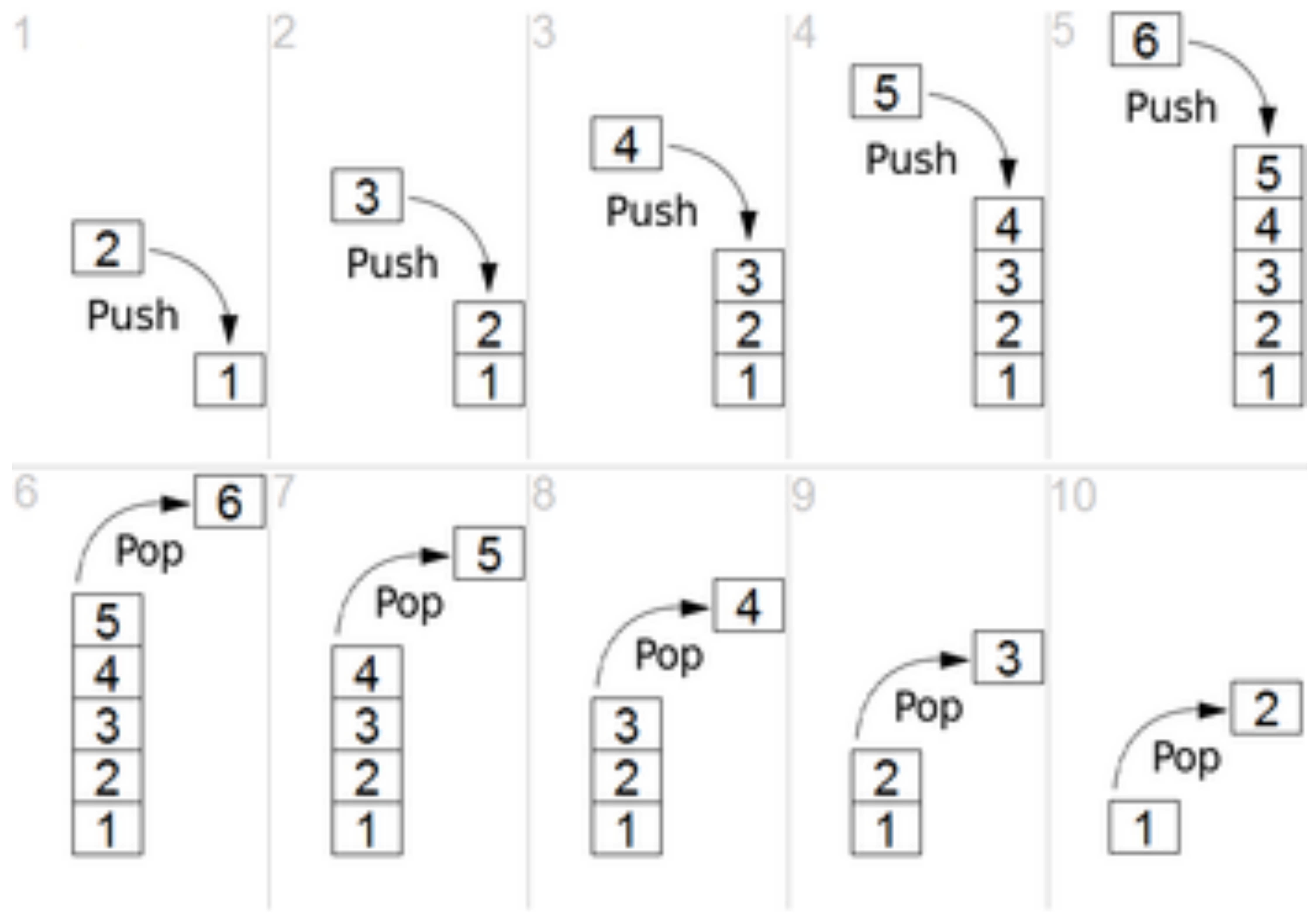
Average-case (probabilistic)

What happens to time- and space- complexities if the coin is not fare, e.g., an item is carried to upper level with probability p, rather than 0.5 ? What do you expect to observe in practice regarding to this change ?

Choosing the right data structure …

What is data structure ? How to decide among alternatives …

Stacks



STACK: Last-In-First-Out

- ***Abstract Data Type*** ?
- push(), pop(), top(),empty(), length()
- Array-based or linked-list based implementations ?

Stacks

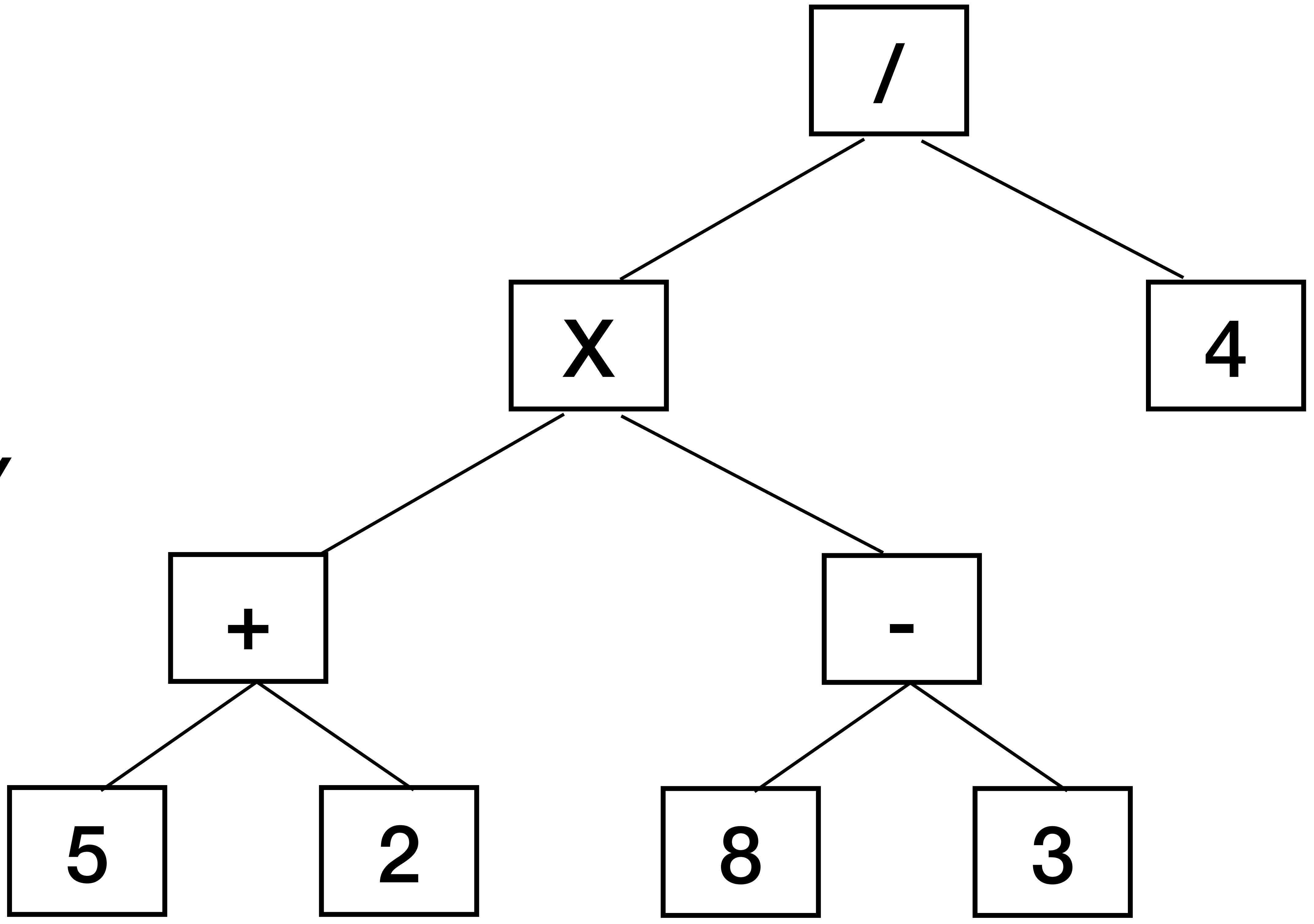
- *Parentheses checking {((5+2) ∗ [(8–3)) / 4] }*

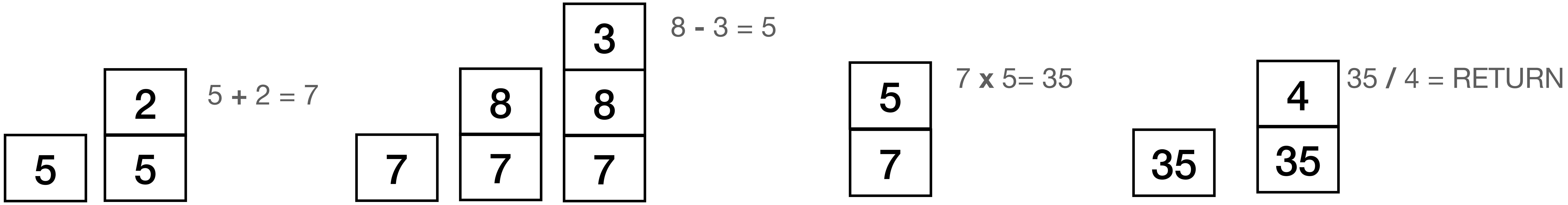
{
(
Matches
)
[
X
DOES NOT Match

Notice that this works for any type of tag matching, e.g., matching the xml tags in a document.

Stacks

Postfix equation evaluation ***5 2 + 8 3 – x 4 /***





Stacks

In a group of n people, a celebrity is a person who is known by everyone and does not know anyone. How can you find a celebrity in a given party of n people.

- Assume $a_1, a_2, \ldots, a_n$ denote the people in the party.
- In a pair of $\langle a_i, a_j \rangle$, does a_i know a_j ?

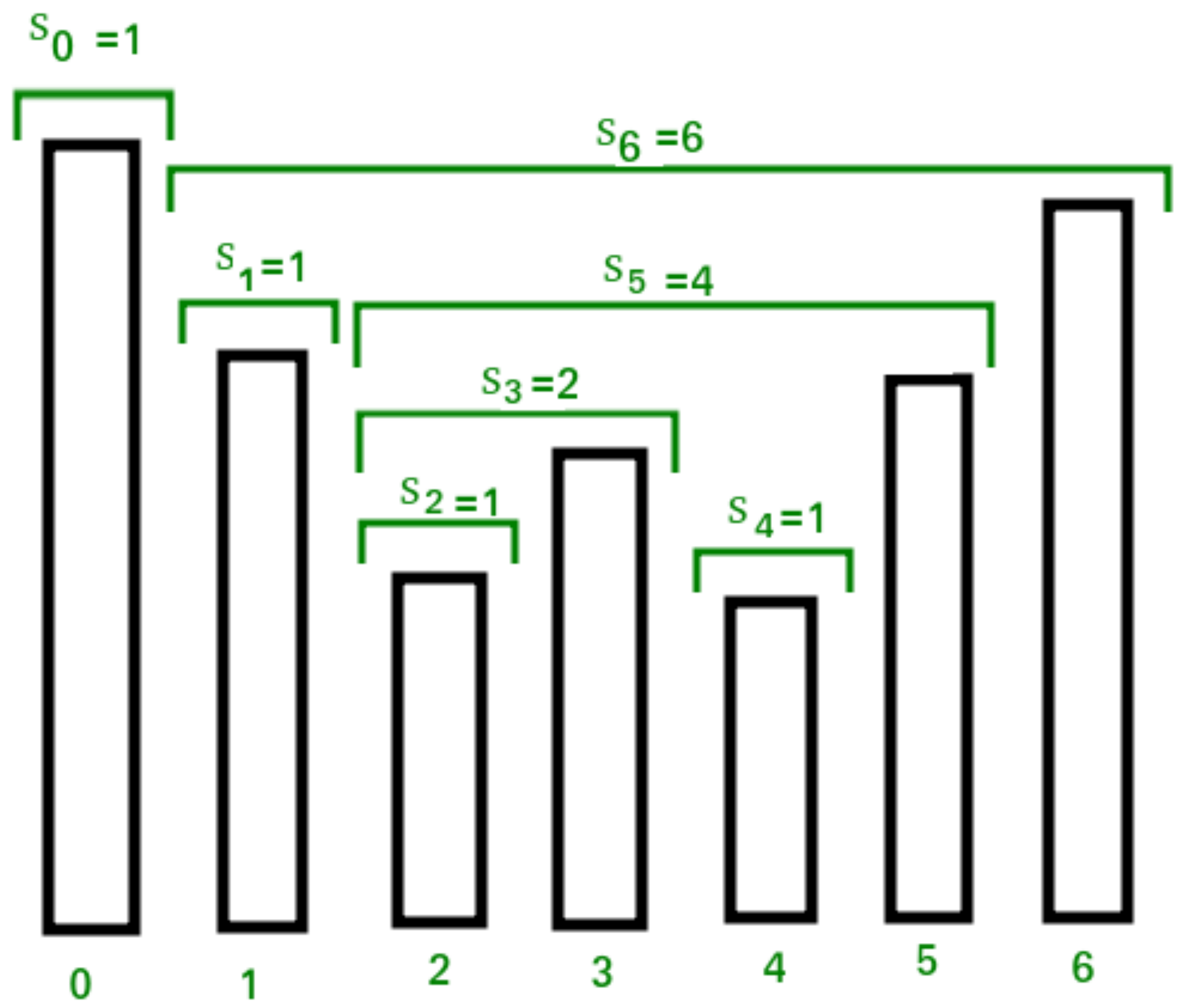
There are $n(n-1)$ such pairs, so is it $O(n^2)$ time ?

- If yes, a_i cannot be the celebrity ?
- If no, a_j cannot be the celebrity ?

Stacks

Stock Span Problem: Today's price is the highest of the last k days. What is k per each day ?

SPAN:	1	1	1	2	1	4	6
PRICE:	100	80	60	70	60	75	85

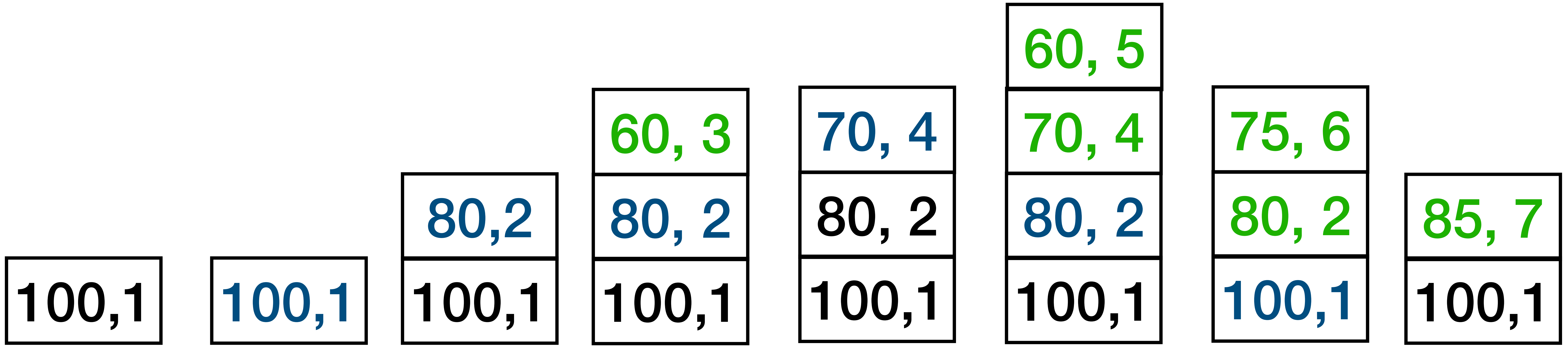


Naive solution is $O(n^2)$. We can make $O(n)$ with the stack

```
void calculateSpan(int price[], int n, int S[]) {
  S[0] = 1;
  for (int i = 1; i < n; i++) {
    S[i] = 1;
    for (int j = i - 1;
      (j >= 0) &&
      (price[i] >= price[j]); j--)
      S[i]++;
  }
}
```

Stacks

Stock Span Problem: Today's price is the highest of the last k days.
What is k per each day ?



i :	2	3	4	5	6	7
PRICE[I] :	80	60	70	60	75	85
SPAN[i] :	1	1	2	1	4	6

Stacks

Stock Span Problem: Today's price is the highest of the last k days.
What is k per each day ?

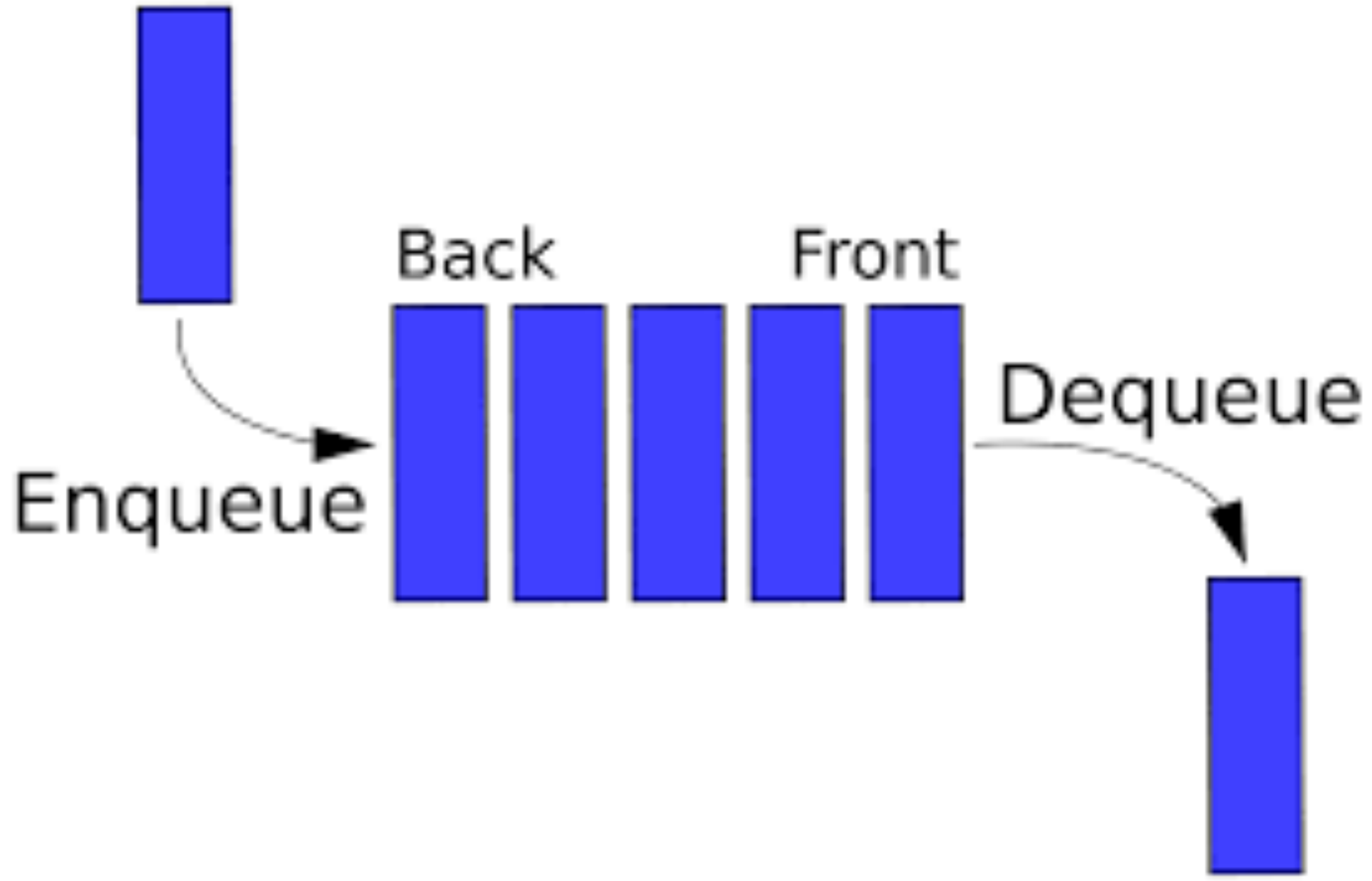
```
void calculateSpan(int price[], int n, int S[]) {
  stack < int > st;
  st.push(0);
  S[0] = 1;
  for (int i = 1; i < n; i++) {
    while (!st.empty() && price[st.top()] < price[i])
      st.pop();

    S[i] = (st.empty()) ? (i + 1) : (i - st.top());
    st.push(i);
  }
}
```

Why this is $O(n)$?

Queue

FIRST-IN-FIRST-OUT



Enqueue, dequeue, isEmpty, front, back, length….

Implementation with arrays, linked-list ?

Queue

Interleave first half with the second: 1.2.3.4.5.6.7.8 => 1.5.2.6.3.7.4.8

Push the first half into a stack
4 3 2 1
5 6 7 8
Enqueue from the stack
5 6 7 8 4 3 2 1
Dequeue and Enqueue the first half
4 3 2 1 5 6 7 8
Push the first half into a stack
1 2 3 4
5 6 7 8
5 6 7 8 1
6 7 8 1 5
2 3 4
Interleave the elements from stack and queue
6 7 8 1 5 2
7 8 1 5 2 6
3 4
7 8 1 5 2 6 3
8 1 5 2 6 3 7
4
8 1 5 2 6 3 7 4
1 5 2 6 3 7 4 8

Reading assignments

- We studied a probabilistic data structure, the skip-list, to polish our knowledge on linked lists and discussed the average case analysis.
- We have reviewed **stack** and **queue** data structures.
- Please read related sections from the textbooks to improve understanding.
- We will continue with the review of **trees** on next lecture…