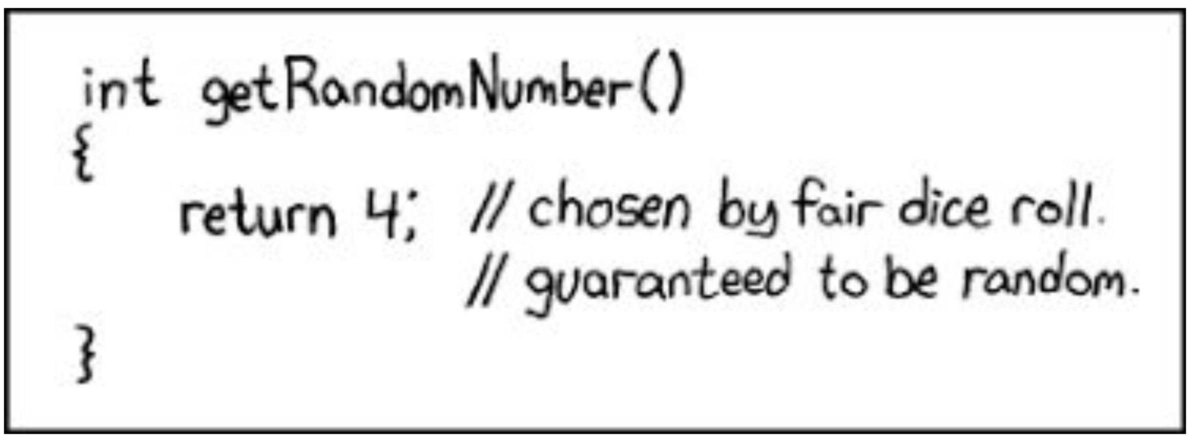
More Randomized Algorithms



When is Randomization Necessary?

my "bonus" lecture next week

- In the areas of online algorithms / cryptography / games / etc. when the input is hidden, we can often **prove** randomization is necessary.

- In the "standard" setting (input given upfront), whether or not randomization is necessary is a **big open problem**. Most researchers seem to believe *every* randomized algorithm can be derandomized.

- When randomization isn't necessary, it's still useful for getting simpler and/or faster-in-practice algorithms, and it provides inspiration for designing deterministic algorithms.

Tools from last time

An ***indicator*** random variable X has 2 possible outcomes: 0 and 1.

Expected value of an indicator r.v.: **E[X] = Pr[X=1].**

Linearity of Expectation: For any (not necessarily independent!) random variables N_i:

$$\mathbb{E}\Big[\sum_i N_i\Big] = \sum_i \mathbb{E}[N_i].$$

Markov's Inequality: If X is a *non-negative* random variable and $a > 0$, then $\Pr[X \geq a] \leq \mathbb{E}[X]/a$.

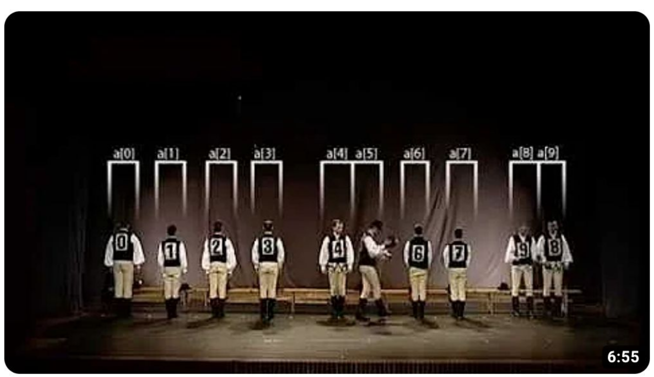
QuickSort

QuickSort is a commonly used randomized sorting algorithm.

1. Pick an array element as the **“pivot”**.
2. Compare pivot to each element to partition list into **two parts**: elements **less that pivot** and elements **greater than pivot**.
3. **Recurse** on both parts of list.

How do you choose the **pivot**?

We will analyze a common strategy: **<u>choose uniformly at random!</u>**



Quick sort with Hungarian, folk dance

153K views • 10 years ago

kamal yassin

https://www.youtube.com/watch?v=3San3uKKHgg

From EECS 281 on quicksort:

Time Analysis

- Cost of partitioning N elements: $O(N)$
- Worst case: pivot always leaves one side empty
  - $T(N) = N + T(N - 1) + T(0)$
  - $T(N) = N + T(N - 1) + c$ [since $T(0)$ is $O(1)$]
  - $T(N) \sim N^2/2 \Rightarrow O(N^2)$ [via substitution]
- Best case: pivot divides elements equally
  - $T(N) = N + T(N / 2) + T(N / 2)$
  - $T(N) = N + 2T(N / 2) = N + 2(N / 2) + 4(N / 4) + \ldots + O(1)$
  - $T(N) \sim N \log N \Rightarrow O(N \log N)$ [master theorem or substitution]
- Average case: $\boxed{\text{tricky}}$ We have the background for this now
  - Between $2N \log N$ and $\sim 1.39\, N \log N \Rightarrow O(N \log N)$

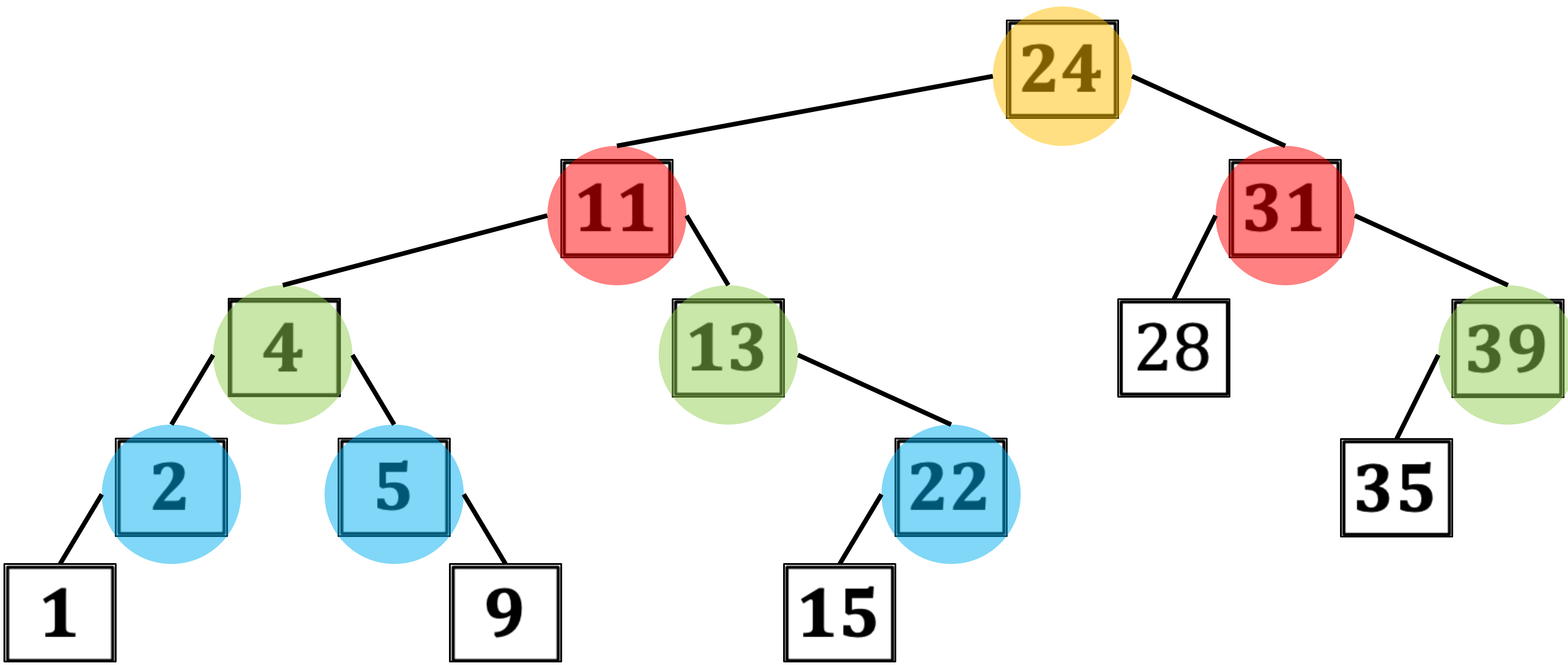
Quicksort

Our goal: Prove that *for any input*, the expected running time of QuickSort is O(n log n).

5	28	22	11	1	35	39	24	2	4	13	9	31	15
5	22	1	11	2	4	13	9	15	**24**	28	35	39	31
5	1	2	4	9	**11**	22	13	15	**24**	28	**31**	35	39
1	2	4	5	9	**11**	**13**	15	22	**24**	28	**31**	**35**	**39**
1	**2**	**4**	**5**	**9**	**11**	**13**	**15**	**22**	**24**	**28**	**31**	**35**	**39**

Analysis of Quicksort

Question: Which pairs of elements were compared during the execution of the algorithm?



Analysis of Quicksort

* **Idea:** Fix all the randomness at the beginning. Randomly assign **priorities** to each element, pick the one with smallest priority when choosing a pivot.

5	28	22	11	1	35	39	24	2	4	13	9	31	15
10	11	13	2	9	6	5	1	7	3	8	12	4	14

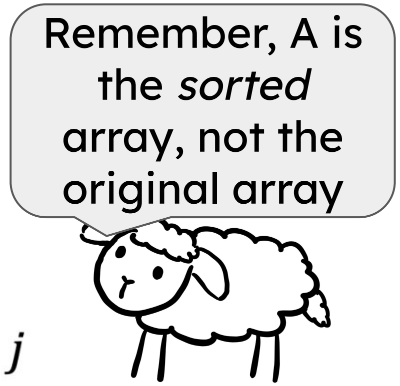
5	22	11	1	2	4	13	9	15	24	28	35	39	31
10	13	2	9	7	3	8	12	14		11	6	5	4

5	1	2	4	9	11	22	13	15	24	28	31	35	39
10	9	7	3	12		13	8	14				6	5

1	2	4	5	9	11	13	22	15	24	28	31	35	39
9	7		10	12			13	14					

1	2	4	5	9	11	13	15	22	24	28	31	35	39

Analysis of Quicksort



* **Example:** Sorted sequence A with priorities:

						i					j		
1	2	4	5	9	11	13	15	22	24	28	31	35	39
9	7	3	10	12	2	8	14	13	1	11	4	6	5

* **Question:** Can you tell if A[i] and A[j] were compared during the algorithm, just by looking at the priorities?

Analysis of Quicksort

Our goal: Prove that *for any input*, the expected running time of QuickSort is O(n log n).

We'll apply the method of <u>indicator random variables</u> + <u>linearity of expectation</u> from last time

Analysis of Quicksort

Let **X** be the number of comparisons made by QuickSort.

Goal: calculate E[X] (since running time of Quicksort = O(X))

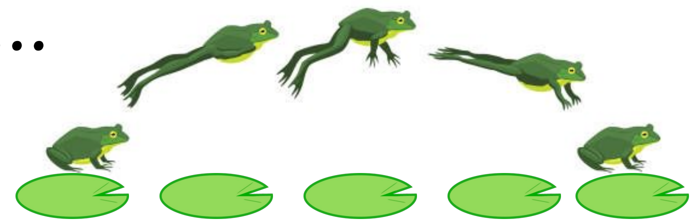
Let $\mathbf{X_{ij}}$ be an indicator r.v. for whether A[i] and A[j] are compared.

Observe that $\mathbf{X} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbf{X_{ij}}$.

Let's use linearity of expectation to calculate E[X].
(question from 2 slides ago will be useful)

Now for a randomized *data structure*…

Skip Lists



A ***dictionary*** is an abstract data type (ADT) that supports insert, find, and delete operations.

Simple implementations: linked list, array

no in-order traversal complicated bookkeeping

Faster implementations: hash tables (various types), balanced binary search trees (AVL, red-black, etc.)

Discovered by William Pugh, 1989

Skip lists are **simpler to implement** (no need for rotations, invariants, bookkeeping)

A **skip list** is like an embellished version of a **linked list**, with expected O(log n) search time (instead of O(n))

Usages of Skip Lists

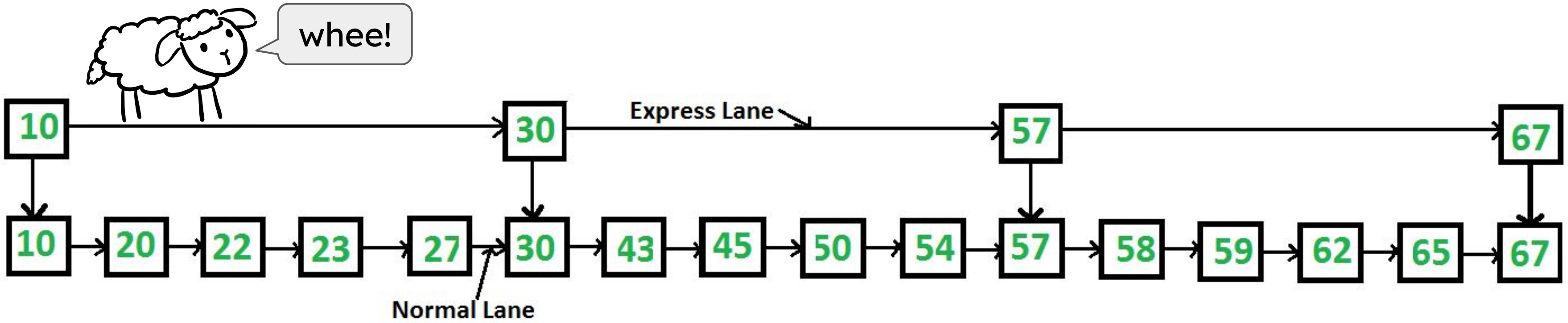
Usages [edit]

List of applications and frameworks that use skip lists:

- Apache Portable Runtime implements skip lists.[9]
- MemSQL uses lock-free skip lists as its prime indexing structure for its database technology.
- MuQSS, for the Linux kernel, is a cpu scheduler built on skip lists.[10][11]
- Cyrus IMAP server offers a "skiplist" backend DB implementation[12]
- Lucene uses skip lists to search delta-encoded posting lists in logarithmic time.[*citation needed*]
- The "QMap" key/value dictionary (up to Qt 4) template class of Qt is implemented with skip lists.[13]
- Redis, an ANSI-C open-source persistent key/value store for Posix systems, uses skip lists in its implementation of ordered sets.[14]
- Discord uses skip lists to handle storing and updating the list of members in a server.[15]
- RocksDB uses skip lists for its default Memtable implementation.[16]

Warm-up: 2-level Linked List

A linked list with an "express lane"

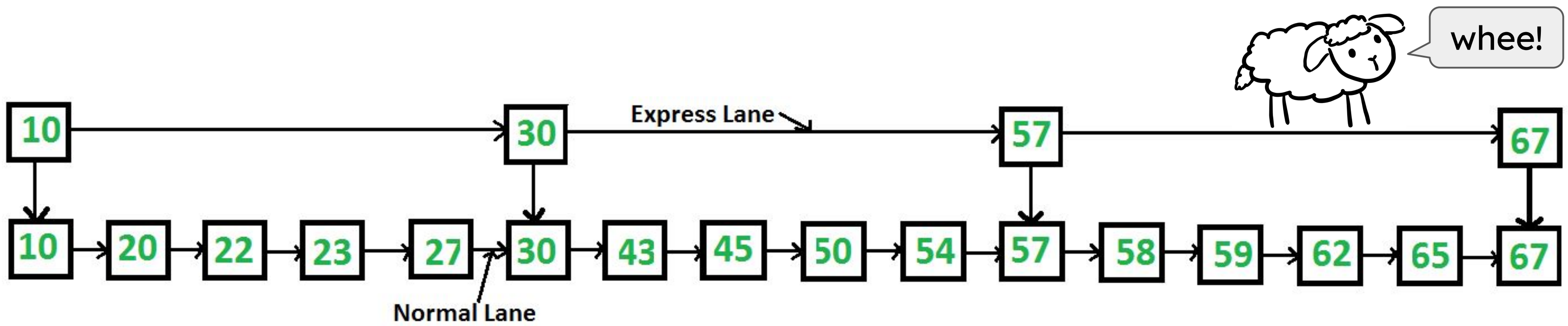


Worst-case running time for search?

How many elements should be in the express lane?

Warm-up: 2-level Linked List

A linked list with an "express lane"



Deterministically keeping express lane roughly evenly spaced amidst insertions/deletions would require some bookkeeping…

Instead let's promote elements to the express lane randomly!

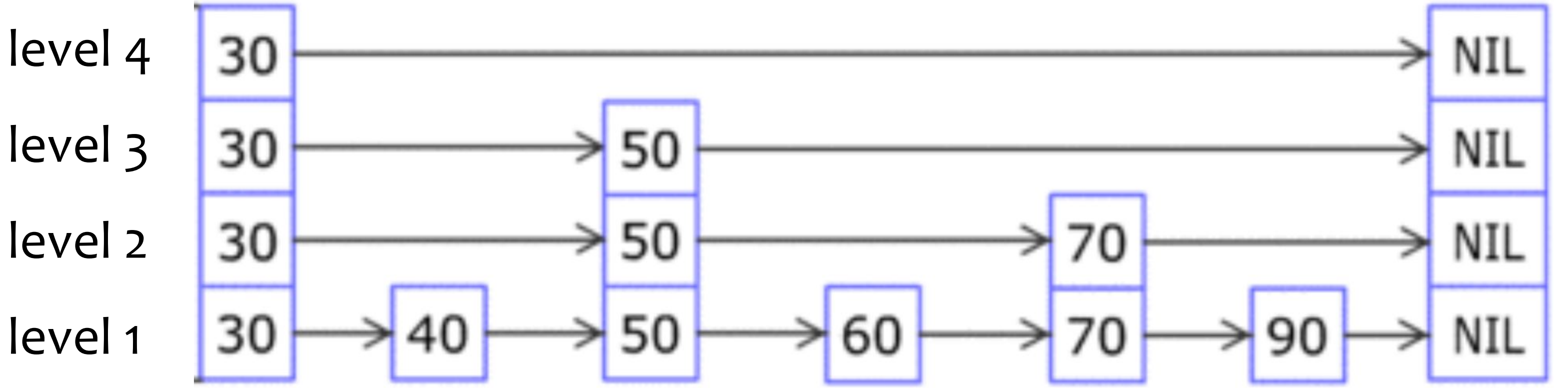
With what probability should we promote each element?

Idea of a Skip List: Add more express lanes!

(Roughly log n of them)

Put every element in **level 1**
Promote about ½ of the elements to **level 2**
Promote about ½ of the elements in **level 2** to **level 3**
Promote about ½ of the elements in **level 3** to **level 4**
⋮

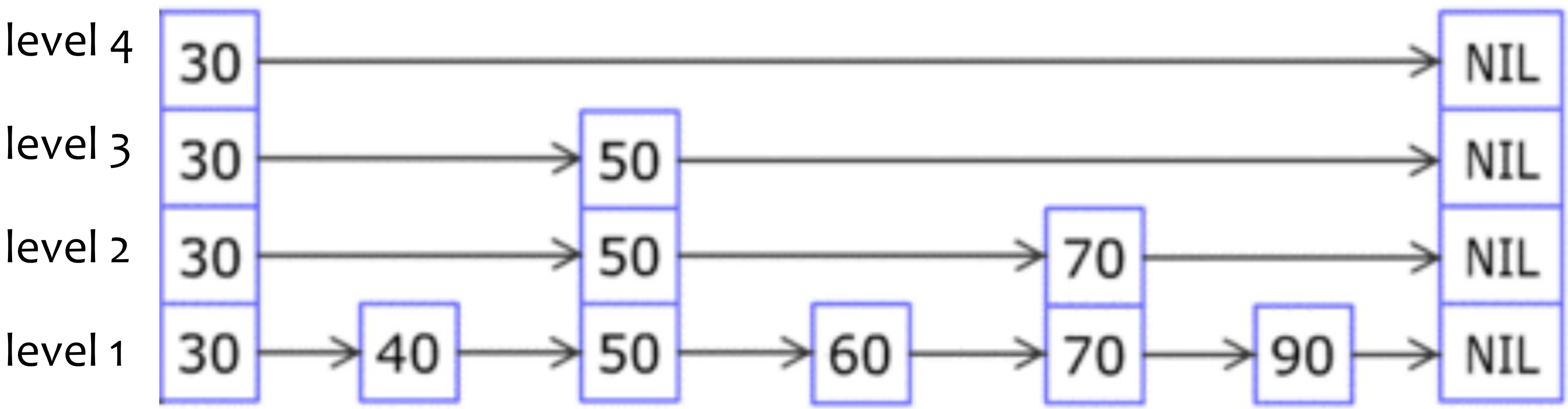
How to accomplish this: For each element, flip a coin and keep promoting until you get tails.



Skip Lists

How do we <u>insert</u> an element?

(<u>delete</u> and <u>find</u> are very similar to <u>insert</u>)



Skip Lists

We will prove: *For any fixed sequence of n operations* (insert, delete, find), the expected running time per operation is O(log n).

Note: Our analysis will rely on the fact that the choice of operations <u>cannot respond to</u> the random choices of the algorithm.
(Imagine the sequence of operations is fixed ahead of time.)

Properties of Skip Lists

Goal #1: Show that the expected number of levels is O(log n).

Suppose we have a skip list containing elements $\mathbf{x_1, x_2}$…

Question 1: In expectation, how many elements are on level **i**?
What tool should we use to answer this question?

Fix a level **i**. Let **X** be the number of elements on level **i**.

Properties of Skip Lists

Goal #1: Show that the expected number of levels is O(log n).

Fix a level **i**. Let **X** be the number of elements on level **i**.

Question 2: What is the probability level **i** has at least one element?
What tool should we use to answer this question?

Properties of Skip Lists

Goal #1: Show that the expected number of levels is $O(\log n)$.

Question 3: In expectation, how many levels have ≥ 1 element?

Let $\mathbf{Y_k}$ be an indicator that **level $\mathbf{\log_2 n + k}$** has ≥ 1 element.

$$E[\#\text{ levels above }\log_2 n] = E\Big[\sum_{k=1}^{\infty} \mathbf{Y_k}\Big]$$

$$= \sum_{k=1}^{\infty} E[\mathbf{Y_k}] \qquad \text{(linearity of expectation)}$$

$$= \sum_{k=1}^{\infty} \Pr[\mathbf{Y_k}=1] \qquad \text{(expectation of indicator)}$$

$$\leq$$

Properties of Skip Lists

Goal #2: Show that in expectation each operation (insert, delete find) touches a <u>constant</u> number of elements per level.

Combining both goals, by linearity of expectation we've proved the original goal:

> **Original Goal:** *For any fixed sequence of n operations* (insert, delete, find), the expected running time per operation is O(log n).

Properties of Skip Lists

Goal #2: Show that in expectation each operation (insert, delete find) touches a <u>constant</u> number of elements per level.

Say we are inserting (or deleting or finding) **y**.

Fix a level **i**. Let **Z** be the number of elements touched on level **i**.

Let $\mathbf{Z_j}$ be an indicator variable for whether $\mathbf{e_j}$ is touched (on level **i**)

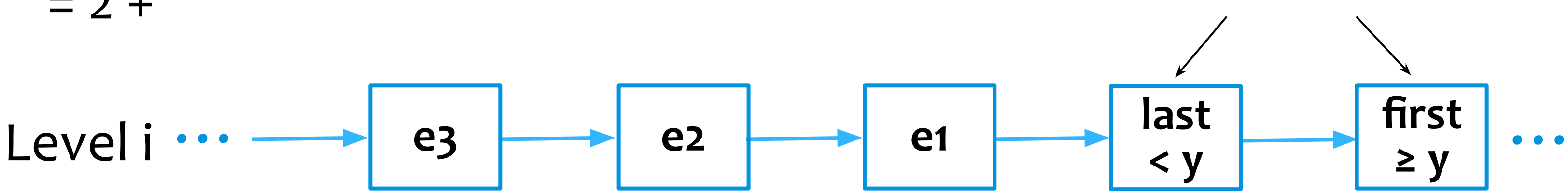
$$E[\mathbf{Z}] = 2 + E[\Sigma_j \mathbf{Z_j}]$$

$$= 2 + \Sigma_j E[\mathbf{Z_j}] \qquad \text{(linearity of expectation)}$$

$$= 2 + \Sigma_j \Pr[\mathbf{Z_j}=1] \qquad \text{(expectation of indicator)}$$

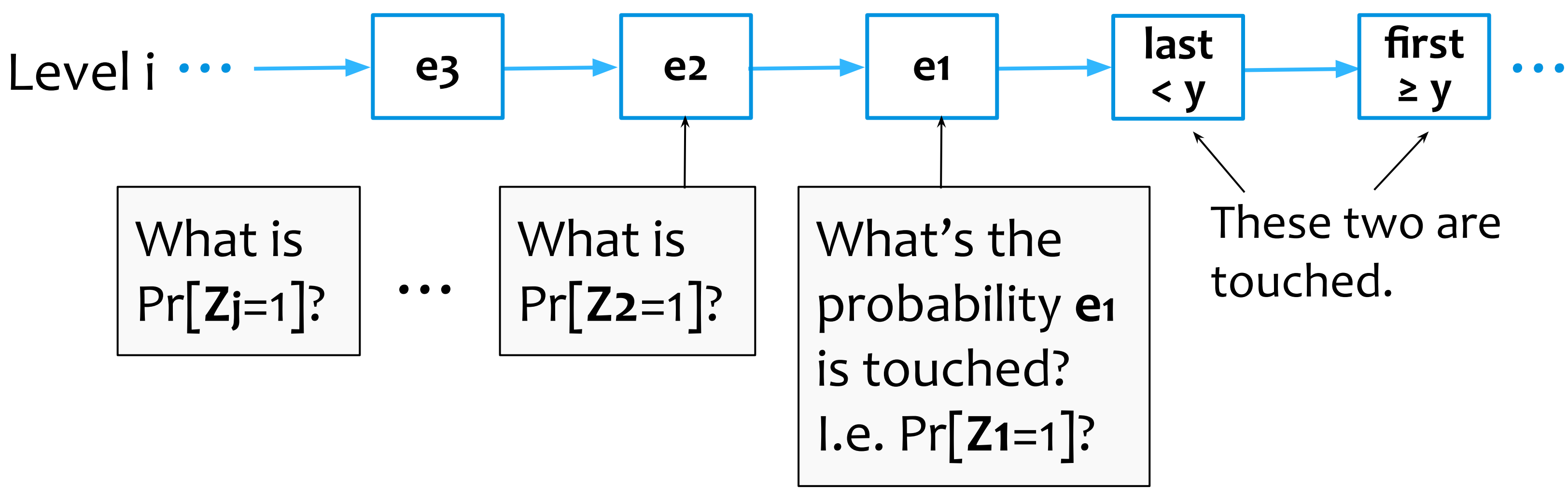
$$= 2 +$$

Claim: These two are touched. Why?



Properties of Skip Lists

Goal #2: Show that in expectation each operation (insert, delete find) touches a <u>constant</u> number of elements per level.



What we showed

Goal #1: Show that the expected number of levels is O(log n).

Goal #2: Show that in expectation each operation (insert, delete find) touches a constant number of elements per level.

Original Goal: *For any fixed sequence of n operations* (insert, delete, find), the expected running time per operation is O(log n).

High-level takeaway:

- Often, deterministic data structures use a lot of bookkeeping to maintain the desired properties.

- With randomization, we stop micromanaging our data structure and use random choices that satisfy the desired properties on average.