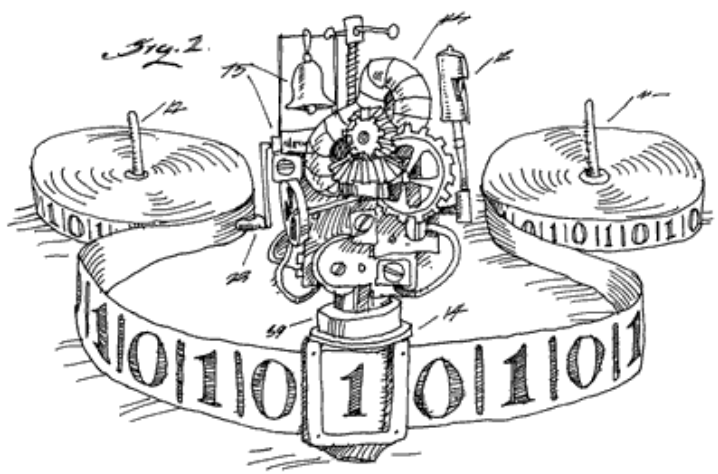
EECS 376: Foundations of Computer Science

Lecture 21 - Randomness in Computation 2



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:

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

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

Max-3SAT

Input: 3CNF formula with m clauses

E.g. $(x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z)$
$\wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z)$

Algo: Pick an assignment uniformly at random!

We showed: *For any 3CNF formula,*

the ***expected value*** of the fraction of satisfied clauses is ⅞.

Let N' = #unsat clauses. So, $E[N'] = m/8$.

We say that we **fail** if $N' \geq m/4$.

By Markov, $\Pr[N' \geq m/4] \leq \frac{E[N']}{m/4} = \frac{m/8}{m/4} = 0.5$.

Can we make **failure** probability even smaller?

Simple trick: repeat!

Can we reduce failure probability to 0.0001? Easy!

Algo:

- Sample 100 random assignments **independently**.
- Take the best one.

$$\Pr[\text{all 100 assignments fail}] \leq \left(\frac{1}{2}\right)^{100}$$

Conclude:

- We fail with probability $\left(\frac{1}{2}\right)^{100}$.
- Reducing failure probability by repeating is **super useful.**

Today

Simpler algorithms/data structures via randomization

* Quicksort
* Skiplist

Quicksort

randomized *algorithm*

Quicksort

- Quicksort is an efficient, general-purpose sorting algorithm.
- Quicksort was developed by British computer scientist Tony Hoare in 1959 and published in 1961.
- It is still a commonly used algorithm for sorting and is slightly faster than merge sort and heapsort for randomized data, particularly on larger distributions.

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**?

QuickSort

- What if we pick a pivot **uniformly at random!**
- Let X be the number of *comparisons* (main cost in runtime) made in **QuickSort** on array $A[1 \dots n]$

Theorem: $\mathbb{E}[X] = O(n \log n)$,
the *expected* runtime of **QuickSort** is $O(n \log n)$.

Computing $\mathbb{E}[X]$

* Let $S[1 \dots n]$ be the sorted array of $A[1 \dots n]$
* Let $X_{ij} \in \{0,1\}$ be the *indicator* for whether S$[i]$ and S$[j]$ are compared.
* **Claim:** $X = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}$ (i, j compared at most once)

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbb{E}[X_{ij}]$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr[S[i] \text{ and } S[j] \text{ are compared}]$$

We will analyze this!

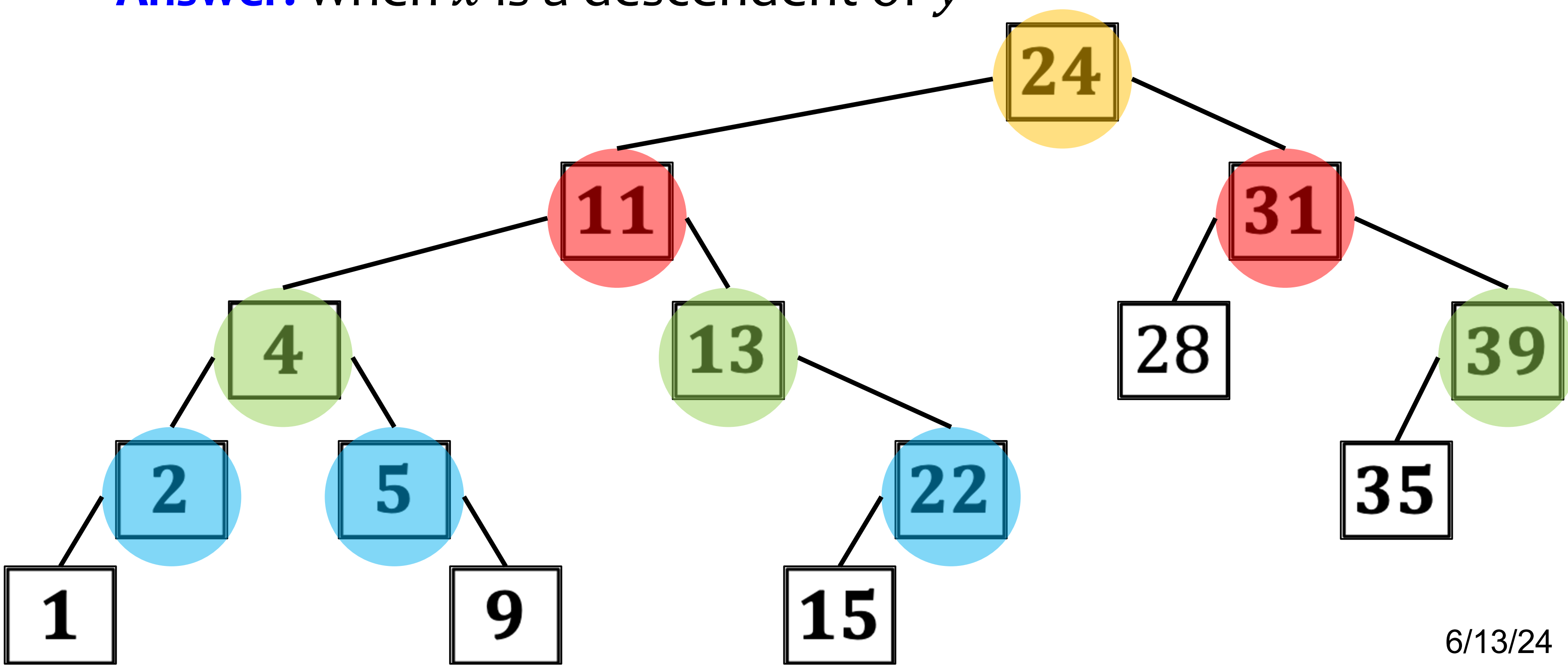
Analysis of Quicksort

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 (x, y) were compared during the execution of the algorithm?

Answer: when x is a descendent of y



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

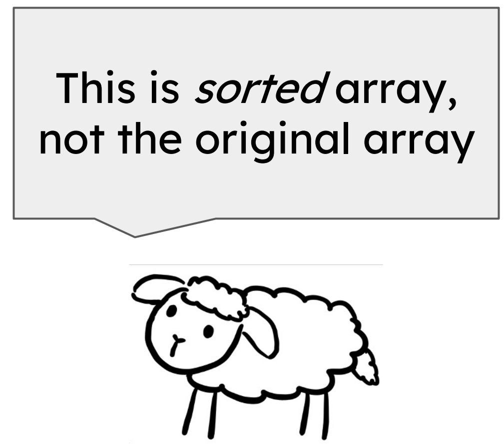
1	2	4	5	9	11	13	22	15	24	28	31	35	39
9	7		10	12			13	14					

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

Analysis of Quicksort

Consider the sorted array $S[1 \ldots \mathrm{n}]$ and priorities below

						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



Analysis of Quicksort

Consider the sorted array $\mathrm{S}[1 \ldots \mathrm{n}]$ and priorities below

						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: Give a condition when $S[i]$ and $S[j]$ compared based on priorities.

Answer: Compared if and only if $S[i]$ or $S[j]$ has smallest priority among $S[i \ldots j]$

- If $S[k]$ has smallest priority where $i < k < j$,
  then $S[i]$ and $S[j]$ are **not compared.** Why?

- If $S[i]$ or $S[j]$ has smallest priority,
  then $S[i]$ and $S[j]$ are **compared.** Why?

Analysis of Quicksort

$\Pr[S[i]$ and $S[j]$ were compared$]$

$= \Pr[\, S[i]$ or $S[j]$ has smallest priority among $A[i \ldots j]]$

This is exactly $\frac{2}{j-i+1}$. Why?

Message: If the two elements are close the sorted array,
They have smaller probability to be compared

Wrap-up

Therefore:

$\Pr[S[i] \text{ and } S[j] \text{ are compared}] = 2/(j - i + 1)$

So:

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbb{E}[X_{ij}]$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr[S[i] \text{ and } S[j] \text{ are compared}]$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j - i + 1} \leq 2n \ln n = O(n \log n)$$

Skiplist

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.)

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))

Discovered by
William Pugh, 1989

Usages of Skip Lists

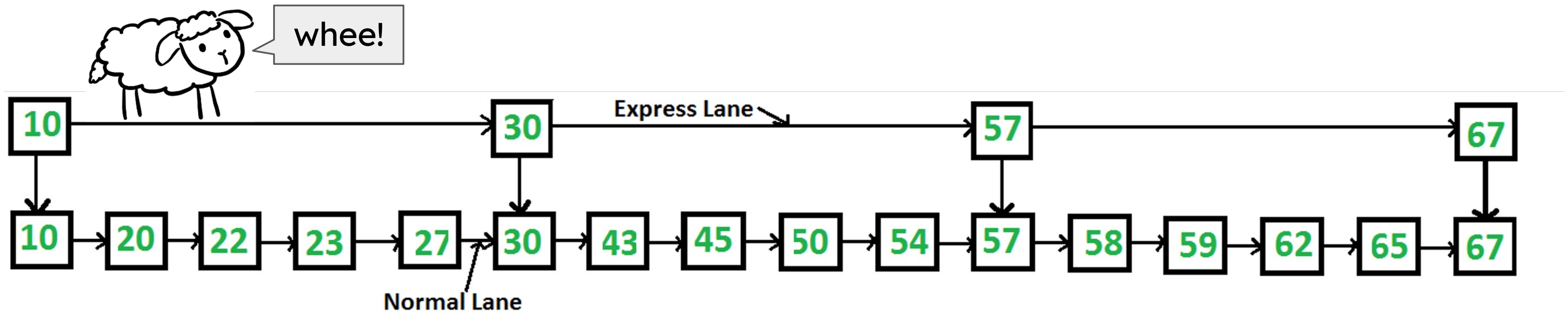
List of applications and frameworks that use skip lists:
- Apache Portable Runtime implements skip lists.
- MemSQL uses lock-free skip lists as its prime indexing structure for its database technology.
- MuQSS, for the Linux kernel, is a CPU scheduler built on skip lists.
- Cyrus IMAP server offers a "skip list" backend DB implementation
- Lucene uses skip lists to search delta-encoded posting lists in logarithmic time.
- The "QMap" key/value dictionary (up to Qt 4) template class of Qt is implemented with skip lists.
- Redis, an ANSI-C open-source persistent key/value store for Posix systems, uses skip lists to implement ordered sets.
- Discord uses skip lists to handle storing and updating the list of members in a server.
- RocksDB uses skip lists for its default Memtable implementation.

Usages of Skip Lists (Cont'd)

Skip lists are also used in distributed applications (where the nodes represent physical computers, and pointers represent network connections) and for implementing highly scalable concurrent priority queues with less lock contention, or even without locking, and lock-free concurrent dictionaries. There are also several US patents for using skip lists to implement (lockless) priority queues and concurrent dictionaries.

Warm-up: 2-level Linked List

A linked list with an "express lane"

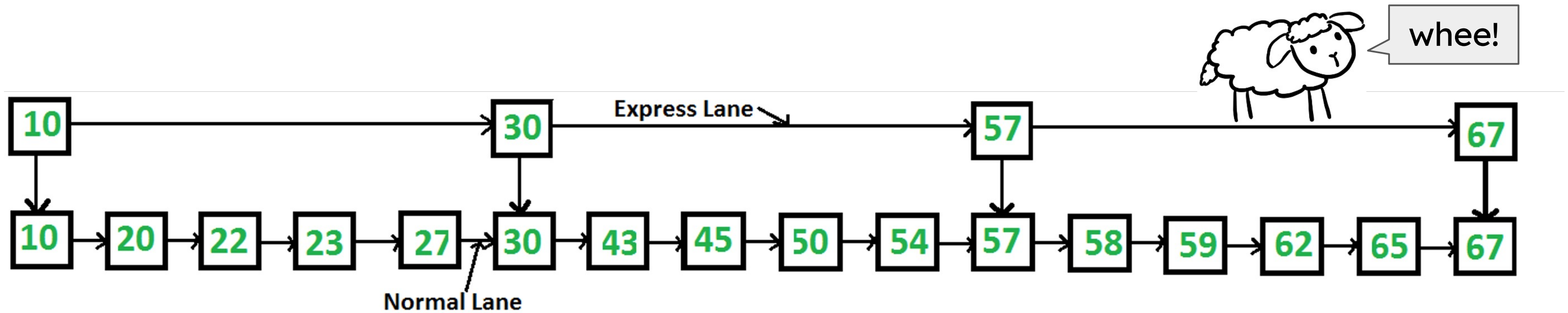


Worst-case running time for search?

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!

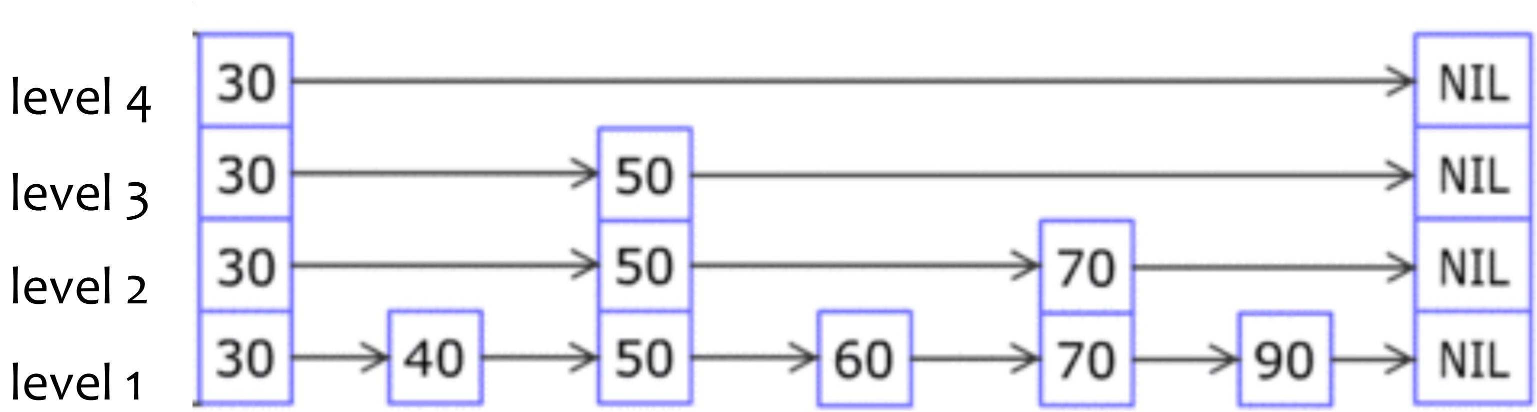
With what probability should we promote each element?

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 insert an element?

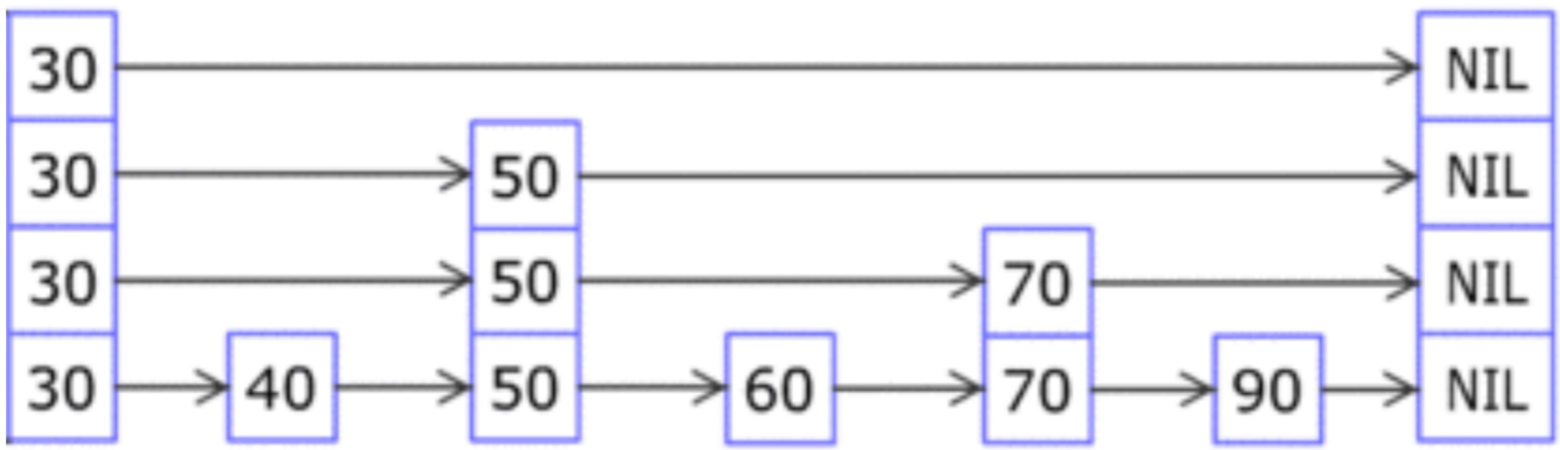
(delete and find are very similar to insert)

level 4

level 3

level 2

level 1



Plan

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

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 <u>constant</u> number of elements per level.

Properties of Skip Lists

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

After inserting elements $x_1, \dots, x_n$.

Question 1: how many elements are on level i?

n_i = number of elements on level i

$n_1 = n$ (lowest level always contains all the elements)

Define X_{ij} indicator variable for event "x_j on level i".

$\mathbb{E}[n_i] = \mathbb{E}[X_{i1} + \cdots + X_{in}]$

$= \sum_j \mathbb{E}\left[X_{ij}\right]$ *(linearity of expectation)*

$= \sum_j 1/2^{i-1}$ *(had to flip $i-1$ heads to get up to level i)*

$= n/2^{i-1}$

Properties of Skip Lists

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

Question 2: How many (non-empty) levels *in expectation*?

Probability that level $\log(n) + k$ is non-empty is

$$\Pr\left[n_{\log(n)+k} \geq 1\right] \leq \frac{\mathbb{E}\left[n_{\log(n)+k}\right]}{1} \qquad \textit{(Markov's inequality)}$$

$$= \frac{n}{2^{\log(n)+k-1}} = \frac{n}{n2^{k-1}} = \frac{1}{2^{k-1}}$$

Define Z_k as an indicator that $n_{\log(n)+k} \geq 1$ (level $\log(n) + k$ is non-empty)

Expected number of levels is at most

$$\log n + \mathbb{E}[\Sigma_{i\geq 1} Z_i] = \log n + \mathbb{E}[Z_1] + \mathbb{E}[Z_2] + \mathbb{E}[Z_3] + \cdots$$
$$= \log n + 2$$

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 **Zj** be an indicator variable for whether **ej** is touched (on level **i**)

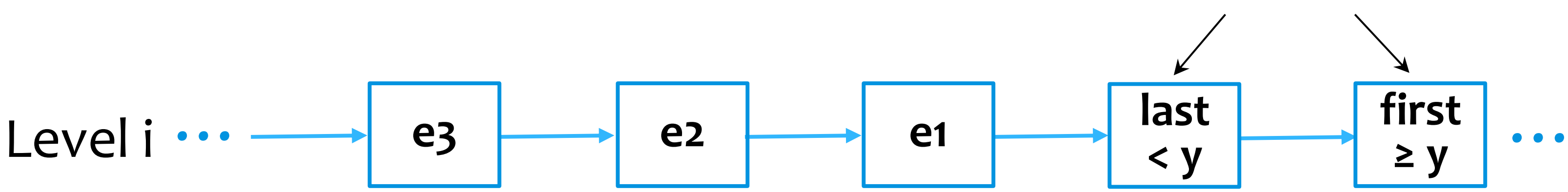
$$E[\mathbf{Z}] = 2 + E[\Sigma_j \mathbf{Zj}]$$

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

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

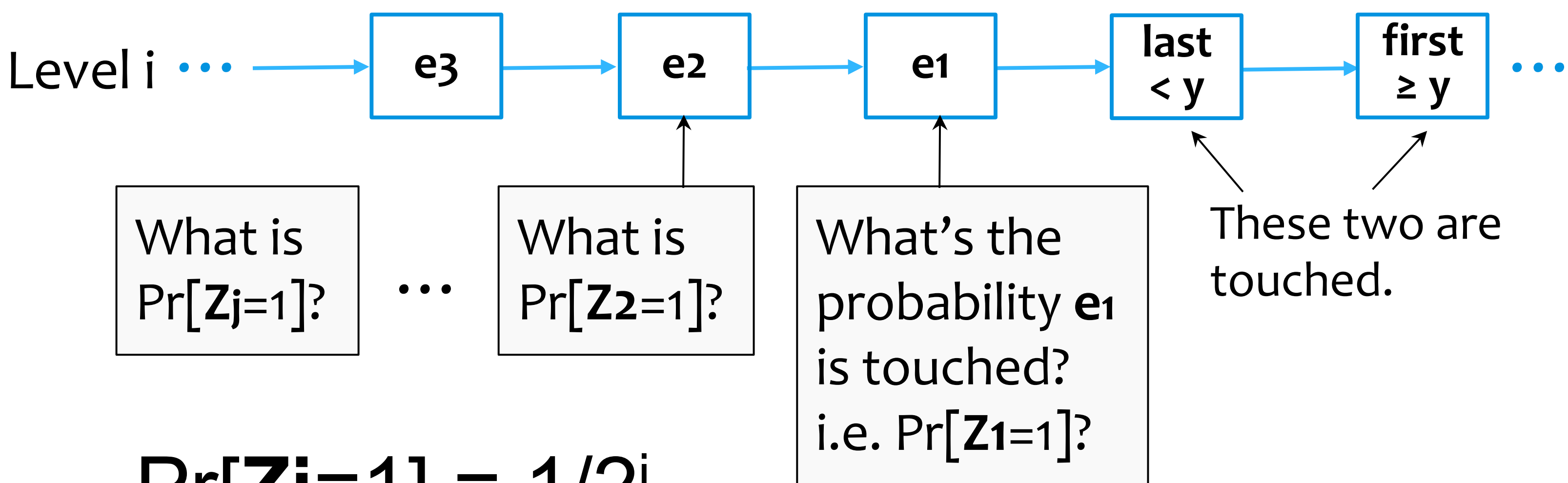
$$= 2 + \Sigma_j (1/2^j) = 3$$

Claim: These two are touched. Why?



Properties of Skip Lists

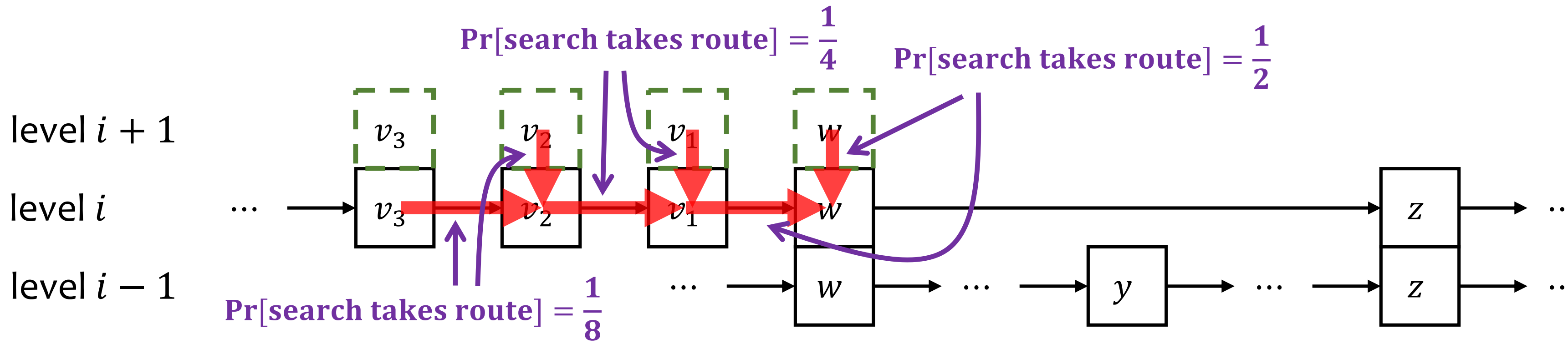
Goal #2: Show that in expectation each operation (insert, delete find) touches a <u>constant</u> number of elements per level.



$$\Pr[Z_j=1] = 1/2^j$$

Goal # 2

* $Y_i = \mathbf{Z}$
* $V_{ij} = \mathbf{e_j}$



Thus, v_1 is encountered on level i exactly when w is not replicated to level $i+1$, which occurs with probability $1/2$. We can inductively repeat the same reasoning for preceding elements. The search only encounters v_2 on level i if neither w nor v_1 are replicated on level $i+1$. Since they are replicated independently with probability $1/2$, the probability that neither is replicated is $1/4$. In general, the search encounters v_j on level i exactly when none of $w, v_1, v_2, \ldots, v_{j-1}$ are replicated to level $i+1$, which happens with probability $1/2^j$.

We can now compute the expected number of nodes encountered on level i. Let Y_i be this number, and let V_{ij} be an indicator for whether or not element v_j is encountered. Then we have

$$Y_i = 2 + \sum_j V_{ij}$$

Goal # 2

The first term 2 is due to the fact that the search encounters the nodes corresponding to w and z. From our reasoning above, we have

$$\mathbb{E}[V_{ij}] = \Pr[V_{ij} = 1] = \frac{1}{2^j}$$

Then by linearity of expectation,

$$\begin{aligned}
\mathbb{E}[Y_i] &= \mathbb{E}\left[2 + \sum_j V_{ij}\right] \\
&= 2 + \sum_j \mathbb{E}[V_{ij}] \\
&= 2 + \sum_j \frac{1}{2^j} \\
&< 2 + \sum_{k=1}^{\infty} \frac{1}{2^k} \\
&= 3
\end{aligned}$$

* Y_i = **Z**
* V_{ij} = $\mathbf{e_j}$

Conclude

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

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.

https://en.wikipedia.org/wiki/Skip_list#

Caveat

Our analysis assumed that the choices of operations <u>do not depend on</u> the random choices of the algorithm.

Why?

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.

Skip Lists