Lists

\[ \text{find } \] \quad \text{walk along list}

\begin{align*}
\text{if } x \in \text{position } i : \\
&\Theta(i) \text{ time} \\
\text{if } x \not\in \text{ list:} \\
&\Theta(n)
\end{align*}
Skip List [Pugh late '80s]

Assume list elements are sorted.

Based on linked list for fast modification.
Again, find in a linked list is in \( \Theta(n) \) time so you can walk along elements.

Add shortcuts to make a second linked list with \( \approx n/2 \) of the elements.
also sorted links to next node & copy of node in original list
Also two sentinel nodes in both sorted lists representing $-\infty$ & $+\infty$. 
Finally, a link to -∞ in second list.

Only check \( \leq \frac{n}{2} \) elements in top list.

+ like 2 more in
original list.

So \( n^{1/2} + 2 \) checks
private static class Node { // internal to a SkipList
class
  public int key;
  public Node down;
  public Node right;
}

More,
Add higher & higher
short cut lists. Each
with ~1/2 the elements
of the previous
public class SkipList {
    // ...

    // assumes this.first has key Integer.MIN_VALUE and
    // this.last has key Integer.MAX_VALUE
    public boolean find(int x) {
        Node current = this.first;
        while (current != this.last && current.key != x) {
            if (current.right.key > x && current.down != null) {
                current = current.down;
            } else {
                current = current.right;
            }
        }

        return (current != this.last);
    }
}
Has $O(\log n)$ levels.

Expect to touch $\sim 2 \log n$ elements per list so $O(\log n)$ time!

Insert/remove:
Find the position to change & update 0(1) links per list
For each node in each list, flip a coin.

Heads: duplicate for next list up

Tails: don't

equivalently -
when we insert an element, repeatedly flip a coin until
We hit tails. Use as many lists as coin flips.
A of levels:

Thm: A skip list with \( n \) elements has \( O(\log n) \) levels with high probability.

\[
\text{prob. } \geq 1 - \frac{1}{n^c} \quad \text{for some constant } c \geq 1.
\]
\( L(x) \): \( \geq \) levels containing element \( x \), not counting bottom level.

\( \Pr \left[ L(x) \geq l \right] = 2^{-l} \)

Skip list has \( \geq l \) levels if \( \Pr \left[ L(x) \geq l \right] \geq 1 \) for at least one
Union bound:

For two events $A$ and $B$,

$$
\Pr[A \cup B] = \Pr[A] + \Pr[B]
$$
\[
\begin{align*}
L_\ell &= \max_x L(x) \\
Pr(L \geq \ell) &= \sum_x Pr[L(x) \geq \ell] \\
&= n \cdot 2^{-\ell} \\
&= \frac{n}{2^\ell}
\end{align*}
\]
Pick any $c \geq 2$.

$$P_r \left[ L \geq c \log n \right] \leq \frac{n}{c \log n}$$

$$= \frac{2^n}{c}$$

$$= \frac{1}{\eta^{c-1}}$$
With a bit more work...

Expected # levels ≤ \text{log}_2 n + 2

Also, expected value of \( L(x) = 1 \) for all \( x \).

\[ \Rightarrow \text{Expected size of skip list is } \Theta(n). \]
Search Time: Imagine going backwards from x's node.

```java
time
```

```java
private findBackwards(Node current) {
    while (current != this.first) {
        if (current.up != null) {
            current = current.up;
        } else {
            current = current.left;
        }
    }
}
```

- essentially same as -

```java
private flipWalk(Node current) {
    while (current != this.first) {
        if (coinFlip == true) {// i.e., current appears in the list above
            current = current.up;
        } else {
            current = current.left;
        }
    }
}
```

heads
Expected # heads seen = expected # tails.

List not too tall, so expected # beads is $O(\log n)$.

So $2 \cdot O(\log n) = O(\log^2 n)$ steps.