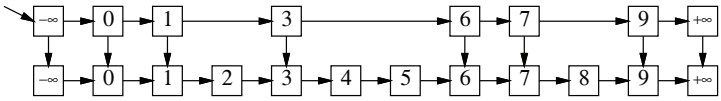
subtree and the right subtree. Since both algorithms define the same two subproblems, by induction, both algorithms perform the same comparisons.

We even saw the probability $\frac{1}{|k-i|+1}$ before, when we were talking about sorting nuts and bolts with a variant of randomized quicksort. In the more familiar setting of sorting an array of numbers, the probability that randomized quicksort compares the ith largest and kth largest elements is exactly $\frac{2}{|k-i|+1}$. The binary tree version compares x_i and x_k if and only if x_i is an ancestor of x_k or vice versa, so the probabilities are exactly the same.

8.5 Skip Lists

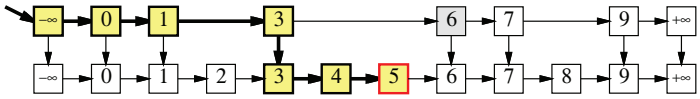
Skip lists, which were first discovered by Bill Pugh in the late 1980's,[7] have many of the usual desirable properties of balanced binary search trees, but their structure is very different.

At a high level, a skip list is just a sorted linked list with some random shortcuts. To do a search in a normal singly-linked list of length n, we obviously need to look at n items in the worst case. To speed up this process, we can make a second-level list that contains roughly half the items from the original list. Specifically, for each item in the original list, we duplicate it with probability $1/2$. We then string together all the duplicates into a second sorted linked list, and add a pointer from each duplicate back to its original. Just to be safe, we also add sentinel nodes at the beginning and end of both lists.



A linked list with some randomly-chosen shortcuts.

Now we can find a value x in this augmented structure using a two-stage algorithm. First, we scan for x in the shortcut list, starting at the $-\infty$ sentinel node. If we find x, we're done. Otherwise, we reach some value bigger than x and we know that x is not in the shortcut list. Let w be the largest item less than x in the shortcut list. In the second phase, we scan for x in the original list, starting from w. Again, if we reach a value bigger than x, we know that x is not in the data structure.



Searching for 5 in a list with shortcuts.

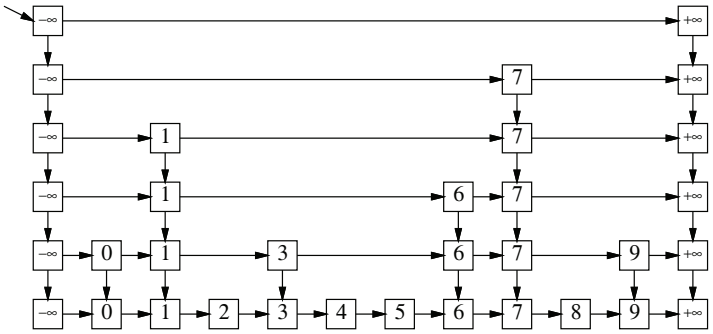
Since each node appears in the shortcut list with probability $1/2$, the expected number of nodes examined in the first phase is at most $n/2$. Only one of the nodes examined in the second phase has a duplicate. The probability that any node is followed by k nodes without duplicates is 2^{-k}, so the expected number of nodes examined in the second phase is at most $1 + \sum_{k\geq 0} 2^{-k} = 2$. Thus, by adding these random shortcuts, we've reduced the cost of a search from n to $n/2 + 2$, roughly a factor of two in savings.

8.6 Recursive Random Shortcuts

Now there's an obvious improvement—add shortcuts to the shortcuts, and repeat recursively. That's exactly how skip lists are constructed. For each node in the original list, we flip a coin over and over until we get tails. For each heads, we make a duplicate of the node. The duplicates are stacked up in levels, and the nodes on each level are strung together into sorted linked lists. Each node v stores a

[7] William Pugh. Skip lists: A probabilistic alternative to balanced trees. *Commun. ACM* 33(6):668–676, 1990.

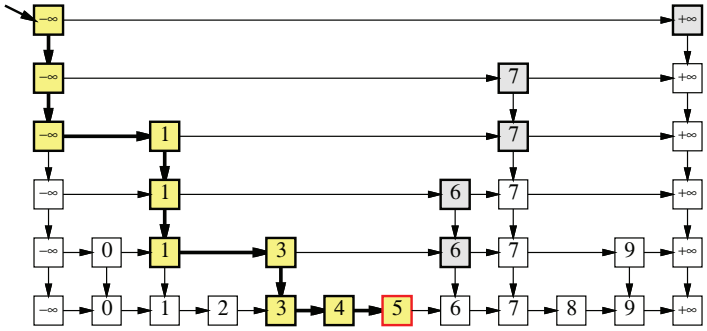
search key ($key(v)$), a pointer to its next lower copy ($down(v)$), and a pointer to the next node in its level ($right(v)$).



A skip list is a linked list with recursive random shortcuts.

The search algorithm for skip lists is very simple. Starting at the leftmost node L in the highest level, we scan through each level as far as we can without passing the target value x, and then proceed down to the next level. The search ends when we either reach a node with search key x or fail to find x on the lowest level.

```
SkipListFind(x, L):
  v ← L
  while (v ≠ Null and key(v) ≠ x)
      if key(right(v)) > x
          v ← down(v)
      else
          v ← right(v)
  return v
```



Searching for 5 in a skip list.

Intuitively, since each level of the skip lists has about half the number of nodes as the previous level, the total number of levels should be about $O(\log n)$. Similarly, each time we add another level of random shortcuts to the skip list, we cut the search time roughly in half, except for a constant overhead, so after $O(\log n)$ levels, we should have a search time of $O(\log n)$. Let's formalize each of these two intuitive observations.

8.7 Number of Levels

The actual values of the search keys don't affect the skip list analysis, so let's assume the keys are the integers 1 through n. Let $L(x)$ be the number of levels of the skip list that contain some search key x, not

counting the bottom level. Each new copy of x is created with probability $1/2$ from the previous level, essentially by flipping a coin. We can compute the expected value of $L(x)$ recursively—with probability $1/2$, we flip tails and $L(x) = 0$; and with probability $1/2$, we flip heads, increase $L(x)$ by one, and recurse:

$$E[L(x)] = \frac{1}{2} \cdot 0 + \frac{1}{2}\big(1 + E[L(x)]\big)$$

Solving this equation gives us $\boldsymbol{E[L(x)] = 1}$.

In order to analyze the expected worst-case cost of a search, however, we need a bound on the *number of levels* $L = \max_x L(x)$. Unfortunately, we can't compute the average of a maximum the way we would compute the average of a sum. Instead, we will derive a stronger result, showing that the depth is $O(\log n)$ ***with high probability***. 'High probability' is a technical term that means the probability is at least $1 - 1/n^c$ for some constant $c \geq 1$; the hidden constant in the $O(\log n)$ bound could depend on c.

In order for a search key x to appear on level ℓ, it must have flipped ℓ heads in a row when it was inserted, so $\boldsymbol{\Pr[L(x) \geq \ell] = 2^{-\ell}}$. The skip list has at least ℓ levels if and only if $L(x) \geq \ell$ for at least one of the n search keys.

$$\Pr[L \geq \ell] = \Pr\big[(L(1) \geq \ell) \vee (L(2) \geq \ell) \vee \cdots \vee (L(n) \geq \ell)\big]$$

Using the *union bound* — $\Pr[A \vee B] \leq Pr[A] + \Pr[B]$ for any random events A and B — we can simplify this as follows:

$$\Pr[L \geq \ell] \leq \sum_{x=1}^{n} \Pr[L(x) \geq \ell] = n \cdot \Pr[L(x) \geq \ell] = \frac{n}{2^\ell}.$$

When $\ell \leq \lg n$, this bound is trivial. However, for any constant $c > 1$, we have a strong upper bound

$$\Pr[L \geq c \lg n] \leq \frac{1}{n^{c-1}}.$$

We conclude that ***with high probability, a skip list has $O(\log n)$ levels.***

This high-probability bound indirectly implies a bound on the *expected* number of levels. Some simple algebra gives us the following alternate definition for expectation:

$$\mathrm{E}[L] = \sum_{\ell \geq 0} \ell \cdot \Pr[L = \ell] = \sum_{\ell \geq 1} \Pr[L \geq \ell]$$

Clearly, if $\ell < \ell'$, then $\Pr[L(x) \geq \ell] > \Pr[L(x) \geq \ell']$. So we can derive an upper bound on the expected number of levels as follows:

$$\begin{aligned}
\mathrm{E}[L(x)] &= \sum_{\ell \geq 1} \Pr[L \geq \ell] \\
&= \sum_{\ell=1}^{\lg n} \Pr[L \geq \ell] + \sum_{\ell \geq \lg n + 1} \Pr[L \geq \ell] \\
&\leq \sum_{\ell=1}^{\lg n} 1 + \sum_{\ell \geq \lg n + 1} \frac{n}{2^\ell} \\
&= \lg n + \sum_{i \geq 1} \frac{1}{2^i} && [i = \ell - \lg n] \\
&= \boxed{\lg n + 2}
\end{aligned}$$

So in expectation, a skip list has *at most two* more levels than an ideal version where each level contains exactly half the nodes of the next level below.

8.8 Logarithmic Search Time

It's a little easier to analyze the cost of a search if we imagine running the algorithm backwards. SKIPLISTFIND takes the output from SKIPLISTFIND as input and traces back through the data structure to the upper left corner. Skip lists don't really have up and left pointers, but we'll pretend that they do so we don't have to write '$v \rightarrow$ down(v)' or '$v \rightarrow$ right(v)'.[8]

```
SKIPLISTFIND(v):
  while (v ≠ L)
      if up(v) exists
          v ← up(v)
      else
          v ← left(v)
```

Now for *every* node v in the skip list, $up(v)$ exists with probability $1/2$. So for purposes of analysis, SKIPLISTFIND is equivalent to the following algorithm:

```
FLIPWALK(v):
  while (v ≠ L)
      if COINFLIP = HEADS
      v ← up(v)
  else
      v ← left(v)
```

Obviously, the expected number of heads is exactly the same as the expected number of TAILS. Thus, the expected running time of this algorithm is twice the expected number of upward jumps. Since we already know that the number of upward jumps is $O(\log n)$ with high probability, we can conclude that the worst-case search time is $O(\log n)$ with high probability (and therefore in expectation).

[8] Leonardo da Vinci used to write everything this way, but not because he wanted to keep his discoveries secret. He just had really bad arthritis in his right hand!