Skip Lists

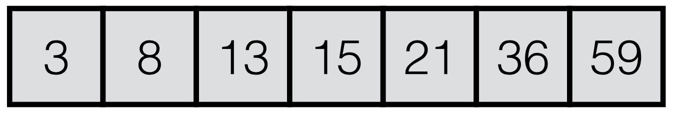
Yet another Map implementation…

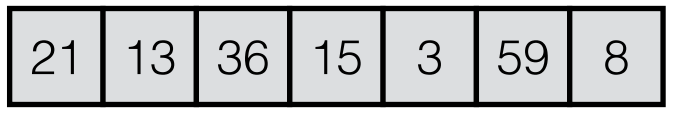
Maps So Far…

We've seen various ways to implement a *map*…

Sorted Array

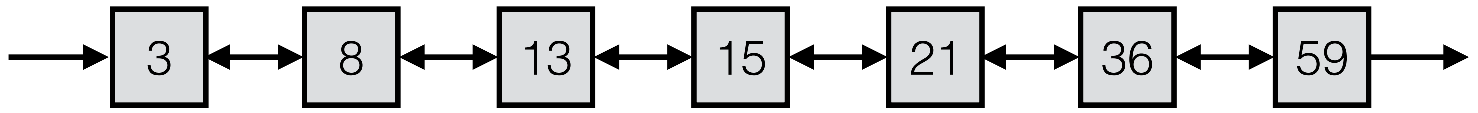
- efficient search — O(lg(n))
- inefficient insert/delete — O(n)





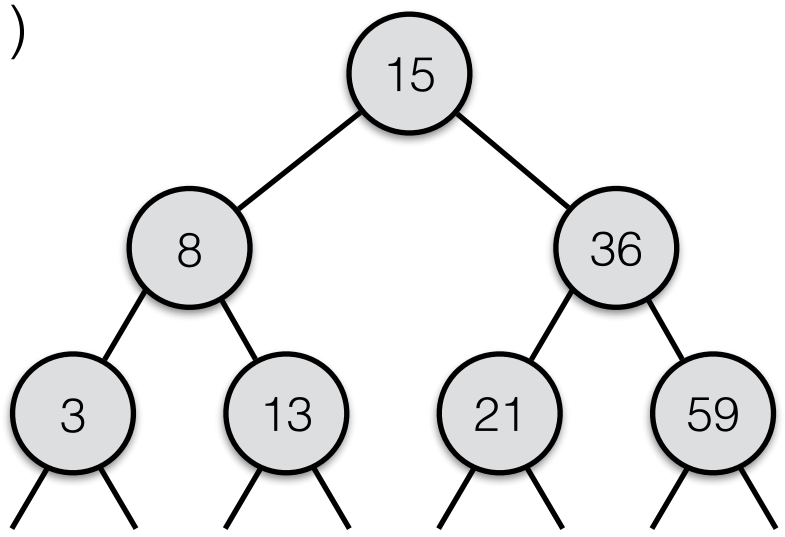
Unsorted Array

- efficient insert/delete — O(1)
- inefficient search — O(n)



Linked Lists

- efficient insert/delete (if we know the position in the list) — O(1)
- no "fast" way to search…



Binary Search Trees

- solves theses problems — search/insert/delete O(lg(n))

Question: *But is there a way that we can use a sorted linked list and still get fast lookup?*

Skip Lists: Setup

- Yes!
- A collection of lists…
- The bottommost list keeps all elements in sorted order, w/ end markers (“sentinels”) -INF and INF.
- Each successive level ***skips*** every 2nd line item
- There are links between items with the same value in successive levels (***towers***) and between the sorted items in a horizontal list (***levels***).
- So, we have…
  - horizontal linked lists of elements (***next*** and ***prev***),
  - vertical towers of the same element (***above*** and ***below***), and
  - note that each “element” holds some data (in this example, ints).

```
S4   -INF <--------------------------------------------------------------------------> INF
       |                                                                               |
S3   -INF <--------------------------------------------------------->  80 <----------> INF
       |                                                                |              |
S2   -INF <----------------------->  15 <--------------------------->  80 <----------> INF
       |                              |                                 |              |
S1   -INF <-------->  8 <--------->  15 <---------->  50  <--------->  80 <----------> INF
       |              |               |                |                |              |
S0   -INF <->  4 <->  8 <->  10 <->  15 <->  20  <->  50  <-> 61  <->  80 <->  100 <-> INF
```

Some questions….

Skip Lists: Searching

Question: *How can we search for a particular key in a skip list?*

Ex. search(61)

- Start at the upper left corner.
- Search through the topmost list until we find something ***bigger than*** 61.
- INF is the first thing that we see.
- Back up, go down a level, and search that list. This time we find 80, back up + drop to list S2 at -INF.
- We go past 15 and find 80. We back up to 15, drop down a level to 15 in S1, and start searching.
- We skip over 50 and stop at 80. We back up to 50 and drop to 50 in S0.
- The next item is 61, which is not bigger than 61, so we go to 80, back up to 61, and discover that there is nothing to drop down to. ***Therefore we have found the biggest item less than or equal to 61, and it is indeed 61.***

```
S4   -INF <--------------------------------------------------------------------------> INF
       |                                                                               |
S3   -INF <--------------------------------------------------------->  80 <----------> INF
       |                                                                |              |
S2   -INF <----------------------->  15 <--------------------------->  80 <----------> INF
       |                              |                                 |              |
S1   -INF <-------->  8 <--------->  15 <---------->  50  <--------->  80 <----------> INF
       |              |               |                |                |              |
S0   -INF <->  4 <->  8 <->  10 <->  15 <->  20  <->  50  <-> 61  <->  80 <->  100 <-> INF
```

Skip Lists: Searching

```
skipSearch(k):
    i = top, leftmost, item in the skip list (ex. -INF of S4).
    while(i.below() != null)
        i = i.below()     // drop down
        while(k >= i.next().data))
            i = i.next()  // scan forward
    return i
```

```
S4   -INF <--------------------------------------------------------------------------> INF
       |                                                                                |
S3   -INF <--------------------------------------------------------->  80 <----------> INF
       |                                                                |               |
S2   -INF <----------------------->  15 <--------------------------->  80 <----------> INF
       |                              |                                 |               |
S1   -INF <-------->  8 <--------->  15 <---------->  50  <--------->  80 <----------> INF
       |              |               |                |                |               |
S0   -INF <->  4 <->  8 <->  10 <->  15 <->  20  <->  50  <-> 61  <->  80 <->  100 <-> INF
```

Skip Lists: Searching

Question: <u>How long does this sort of searching take?</u>

- There are 1 + log(n) (base 2) levels b/c log(n) is precisely the # of times we can divide n by 2 before we get down to a single item.
- lg(11) = 3.45 … — round up —> lg(11) = 4
- Add 1
- 5 levels!

Question: <u>How much time do we spend on each level?</u>

- We look at 1 item or 2 items per level.
- When we find something bigger and back up there was at most one item between the corresponding items at the next level down.
- So we look at it and drop down, or skip over it and find the same item that made us drop down from the level above. So we compare to at most 2(1 + log(n)) items.

==> O(lg(n))

```
S4   -INF <--------------------------------------------------------------------------> INF
       |                                                                                |
S3   -INF <--------------------------------------------------------->  80 <----------> INF
       |                                                                |               |
S2   -INF <----------------------->  15 <--------------------------->  80 <----------> INF
       |                              |                                 |               |
S1   -INF <-------->  8 <--------->  15 <---------->  50  <--------->  80 <----------> INF
       |              |               |                |                |               |
S0   -INF <->  4 <->  8 <->  10 <->  15 <->  20  <->  50  <-> 61  <->  80 <->  100 <-> INF
```

Skip Lists

Question: Doesn't this take a lot of space?

- Ex. We have a tower of log(n) 80's, etc.
- That is true….
- But it doesn't matter!
  - We have n things in S0, n/2 things in S1, and in general n/2^k things in Sk.
  - Summing all of these gives 2n-1.
    - think: *geometric series* if you want to derive this yourself!
  - We also have 2(1 + log2 n) end markers.
  - but the whole thing is Θ(n).

```
S4   -INF <--------------------------------------------------------------------------> INF
       |                                                                                |
S3   -INF <--------------------------------------------------------->  80 <----------> INF
       |                                                                |               |
S2   -INF <----------------------->  15 <--------------------------->  80 <----------> INF
       |                              |                                 |               |
S1   -INF <-------->  8 <--------->  15 <---------->  50  <--------->  80 <----------> INF
       |              |               |                |                |               |
S0   -INF <->  4 <->  8 <->  10 <->  15 <->  20  <->  50  <-> 61  <->  80 <->  100 <-> INF
```

Skip Lists

Question: How does this help us?

- We can do better with a sorted array and binary search…

```
S4   -INF <--------------------------------------------------------------------------> INF
       |                                                                                |
S3   -INF <----------------------------------------------------------> 80 <----------> INF
       |                                                                |               |
S2   -INF <------------------------> 15 <----------------------------> 80 <----------> INF
       |                              |                                 |               |
S1   -INF <---------> 8 <----------> 15 <---------->  50  <----------> 80 <----------> INF
       |              |               |                |                |               |
S0   -INF <->  4 <->  8 <->  10 <->  15 <->  20  <->  50  <-> 61  <->  80 <->  100 <-> INF
```

Skip Lists: Inserting

Question: How does this help us?
- We can do better with a sorted array and binary search…

Question: How about inserting?
- To insert first do a search, and then insert after the thing found in S0.
- O(1) *after* search.

```
skipInsert(k):
    i = skipSearch(k) // get position of bottom-level entry w/ largest key <= k
    // insert k immediately after i (or replace if k == i.data)
```

```
S4   -INF <--------------------------------------------------------------------------> INF
       |                                                                               |
S3   -INF <--------------------------------------------------------->  80 <----------> INF
       |                                                                |              |
S2   -INF <----------------------->  15 <--------------------------->  80 <----------> INF
       |                              |                                 |              |
S1   -INF <-------->  8 <--------->  15 <---------->  50  <--------->  80 <----------> INF
       |              |               |                |                |              |
S0   -INF <->  4 <->  8 <->  10 <->  15 <->  20  <->  50  <-> 61  <->  80 <->  100 <-> INF
```

Skip Lists: Deleting

Question: How does this help us?

- We can do better with a sorted array and binary search…

Question: How about deleting?

- To delete find the thing and then delete it and its tower (if any) from S0 and all of the lists above where it appears.
- O(log(n)) *after* search (since deletion in DLL can be done in O(1))

```
skipDelete(k):
    i = skipSearch(k) // get position of bottom-level entry w/ largest key <= k
    if(key == i.data)
        i.delete()   // easy - we have prev/next/above/below references (takes O(lg(n)))
        return i
    return null
```

```
S4   -INF <--------------------------------------------------------------------------> INF
       |                                                                               |
S3   -INF <----------------------------------------------------------> 80 <----------> INF
       |                                                                |              |
S2   -INF <-----------------------> 15 <----------------------------> 80 <----------> INF
       |                              |                                 |              |
S1   -INF <--------> 8 <---------> 15 <----------> 50  <----------> 80 <----------> INF
       |              |               |                |                |              |
S0   -INF <->  4 <->  8 <->  10 <->  15 <->  20  <->  50  <-> 61   <->  80 <->  100 <-> INF
```

Skip Lists

<u>**Question:** How does this help us?</u>

- We can do better with a sorted array and binary search…

Question: *Won't inserting/deleting this way mess up the structure?*

```
S4   -INF <--------------------------------------------------------------------------> INF
       |                                                                                |
S3   -INF <--------------------------------------------------------->  80 <----------> INF
       |                                                               |                |
S2   -INF <----------------------->  15 <--------------------------->  80 <----------> INF
       |                              |                                |                |
S1   -INF <-------->  8 <--------->  15 <---------->  50  <--------->  80 <----------> INF
       |              |               |                |               |                |
S0   -INF <->  4 <->  8 <->  10 <->  15 <->  20  <->  50  <-> 61  <->  80 <->  100 <-> INF
```

Randomizing Operations

Skip Lists: Randomization

- We don't change our ***search*** or ***delete*** algorithms…
- Need to build new towers when we ***insert***…
- **Question:** How do we know how to "grow" towers?
- Do this by "flipping a coin".
  - Always insert item at S0.
  - Flip a coin — ***head***: insert into S1; flip coin again [repeat].
  - Continue inserting until (result == ***tail***) — then stop.

  [DEMO]

```
S4   -INF <--------------------------------------------------------------------------> INF
       |                                                                               |
S3   -INF <--------------------------------------------------------->  80 <----------> INF
       |                                                                |              |
S2   -INF <----------------------->  15 <--------------------------->  80 <----------> INF
       |                              |                                 |              |
S1   -INF <-------->  8 <--------->  15 <---------->  50  <--------->  80 <----------> INF
       |              |               |                |                |              |
S0   -INF <->   4 <->   8 <->   10 <->   15 <->   20   <->   50   <-> 61   <->   80 <->   100 <-> INF
```

Skip Lists: Insert

[UPDATED]

```
skipInsertCoinFlips(k):
    i = skipSearch(k) // get position of bottom-level entry w/ largest key <= k
    // insert k immediately after i (or replace if k == i.data
    while(coinFlip() == head)
        i.skipInsertUp() // easy since we have “above” reference
        i = i.above()
```

```
S4   -INF <------------------------------------------------------------> INF
       |                                                                   |
S3   -INF <----------------------------------------------->  80 <-----------> INF
       |                                                    |              |
S2   -INF <------------------------->  15 <---------------------------->  80 <-----------> INF
       |                              |                     |              |
S1   -INF <---------->   8 <---------->  15 <-----------> 50  <----------> 80 <-----------> INF
       |               |              |              |      |              |
S0   -INF <->   4 <->  8 <->  10 <->  15 <->  20  <->  50  <-> 61  <->  80 <->  100 <-> INF
```

Skip Lists: Randomization

- Note: We don't have quite as nice of a structure as we had before, but it is close…
- **Question:** Could we end up with a terrible structure?
  - Yes…
  - We could keep flipping heads…
  - Unlikely, but possible…
  - Or, we could keep flipping tails (all items only in S0) —> searching O(n)…
  - However, in the ***expected*** case, we have O(lg(n)) insert, delete, and search!!!
    - Note: unlike BTs, the runtime depends on coin-flips, not the # of elements
    - Implications: no good/bad data sets — just good/bad series of coin tosses.
      - *rant about "malicious" users….*

But what happens to the expected height, expected run time, and the expected size? It takes some probability to see it, but they all end up being the same as in the non-randomized case! The book does these analyses, but here is the outline:

```
S4   -INF <--------------------------------------------------------------------------> INF
       |                                                                                |
S3   -INF <--------------------------------------------------------->  80 <----------> INF
       |                                                                |               |
S2   -INF <----------------------->  15 <--------------------------->  80 <----------> INF
       |                              |                                 |               |
S1   -INF <-------->  8 <--------->  15 <---------->  50  <--------->  80 <----------> INF
       |              |               |                |                |               |
S0   -INF <->  4 <->  8 <->  10 <->  15 <->  20  <->  50  <-> 61  <->  80 <->  100 <-> INF
```

Skip Lists: Expected Height

- The ***expected height*** of the structure (number of lists) is 2 + lg n
- The exact analysis is more appropriate for our grad. level algorithms course…
- Notes (Intuition):
  - Prob. item in list Sk is 2^-k — need k successive “heads” (each has prob. 1/2)
  - *<fancy math>* —> prob. height > 5 lg(n) is no larger than 1/n^4
  - The book notes that 1 of 2 things can be done to deal with growing towers that exceed (or will exceed) the current height of the list:
    1. Allow the tower to grow (it becomes less and less likely the tower will go higher as it grows).
    2. Cap the growth of the list (e.g., max(10, 3 lg(n))) / stop flipping.
       - —> allows us to guarantee the height of the tree is Θ(log n).

```
S4   -INF <--------------------------------------------------------------------------> INF
       |                                                                               |
S3   -INF <--------------------------------------------------------->  80 <----------> INF
       |                                                                |              |
S2   -INF <----------------------->  15 <--------------------------->  80 <----------> INF
       |                              |                                 |              |
S1   -INF <-------->  8 <--------->  15 <---------->  50  <--------->  80 <----------> INF
       |              |               |                |                |              |
S0   -INF <->  4 <->  8 <->  10 <->  15 <->  20  <->  50  <-> 61  <->  80 <->  100 <-> INF
```

Skip Lists: Expected Size

- Easier to analyze — ***how many items do we expect in level Sk?***
- Previously, we noted that an item appears in level k with probability 2^-k.
- There are n items…
- [theorem] linearity of expectation —> we know expectation (mean) of a sum is the sum of the expectations.
  - E[x1 + x2 + … + xN] = E[x1] + E[x2] + … + E[xN]
  - expected # items in Sk = n*2^-k — same as non-probabilistic version!
  - sum # items over 0 to infinity…
- Thus, ***expected size*** is 2n (linear space)

```
S4   -INF <--------------------------------------------------------------------------> INF
       |                                                                               |
S3   -INF <---------------------------------------------------------->  80 <----------> INF
       |                                                                |              |
S2   -INF <----------------------->  15 <--------------------------->  80 <----------> INF
       |                              |                                 |              |
S1   -INF <-------->  8 <--------->  15 <---------->  50  <--------->  80 <----------> INF
       |              |               |                |                |              |
S0   -INF <->  4 <->  8 <->  10 <->  15 <->  20  <->  50  <-> 61  <->  80 <->  100 <-> INF
```

Skip Lists: Expected # Comparisons / Level

- What about the number of comparisons on a given level?
- In general the number of times that we expect to repeat an action before we "succeed" when the probability of success is 1/p is p.
    - ex. If you roll a die until you get a 2, the probability of success is 1/6 and the expected number of rolls is 6.
- Similarly, if we go past an item on Sk, it is because it wasn't promoted to Sk+1.
    - (if it had been, we would have looked at it on that level, not now)
- Therefore, the coin flip when we inserted at that level must have been a "tail"
- The prob. of passing by an item ***without*** dropping down is 1/2 (prob[drop down] = 1/2 also!)
- This is like asking how many flips we need before we get a head — 2 since prob. is 1/2.
- Thus, ***expected # comparisons*** is 2*height ==> expected O(lg(n))

```
S4   -INF <-----------------------------------------------------------------------> INF
       |                                                                              |
S3   -INF <-------------------------------------------------------->  80 <-----------> INF
       |                                                               |              |
S2   -INF <----------------------->  15 <---------------------------->  80 <-----------> INF
       |                              |                                |              |
S1   -INF <--------->  8 <---------->  15 <----------->  50  <---------->  80 <-----------> INF
       |               |              |                 |               |              |
S0   -INF <->  4 <->  8 <->  10 <->  15 <->  20  <->  50  <-> 61  <->  80 <->  100 <-> INF
```

Skip Lists: Final Thoughts…

- Skip Lists are nice because the algorithms are relatively simple, yet they provide efficient map operations.
- No guarantees though — we talk about *expected* running times rather than *best/worst case* running times.
- In practice, there are optimizations that we use to improve the performance of skip lists…
  - Can store references to values rather than actual values in each element.
  - Can store horizontal lists as SLL — cleverly doing operations in a scan forward/down way
  - A tower can be represented as a single object (list of *j* references)
  - Note, however, that these don't actually improve the asymptotic performance of skip lists by more than a constant factor.
- Experimental evidence suggests that optimized skip lists are faster in practice than AVL trees and other balanced search trees (which we've already discussed — 2-3-4 Trees & R-B Trees)