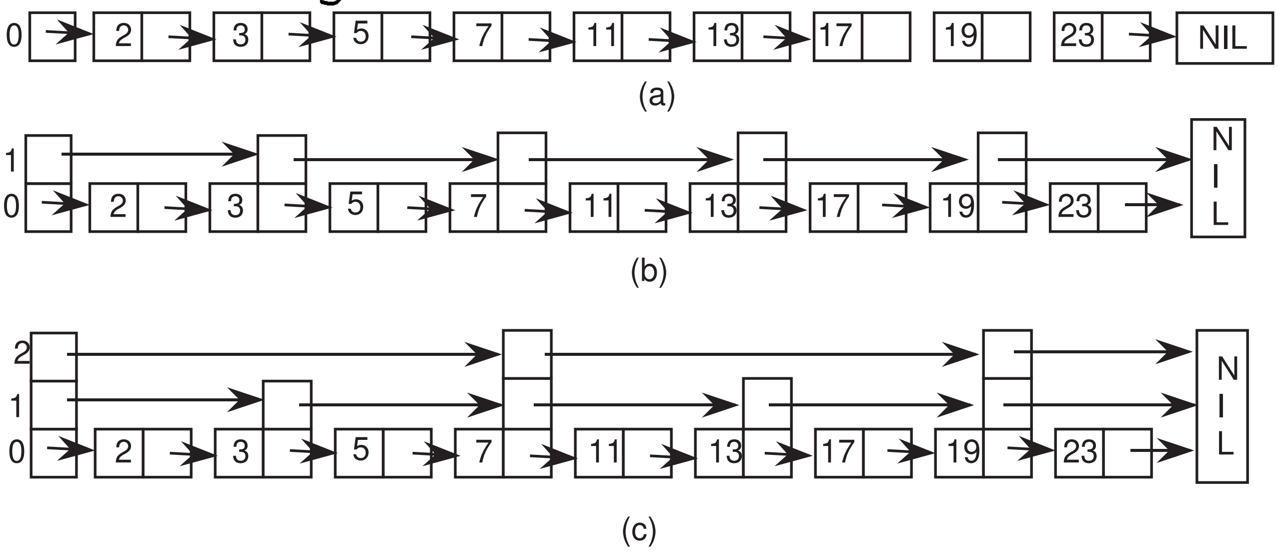
Skip Lists

1. Can't do binary search on a linked list

2. Imagine you use a separate chain to link every other element, and another chain to link up to every 4th element ... these chains are also connected properly, then we can do binary search

3. Total $\log n$ levels



4. But such a data structure is useless – can't insert/delete efficiently.

- Idea: *randomize*

- If the list contains n elements. Let MaxLevel $= \log n - 1$.

- To insert a node, first find its position in the list, then generate its levle randomly with probability:

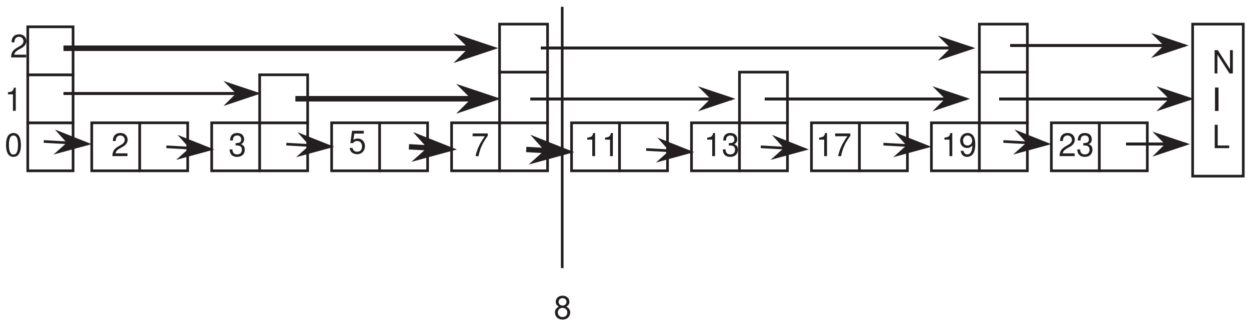
  - 1/2: level 0.
  - 1/4: level 1.
  - 1/8: level 2.
  - $\ldots$
  - $1/2^{k+1}$: level k.

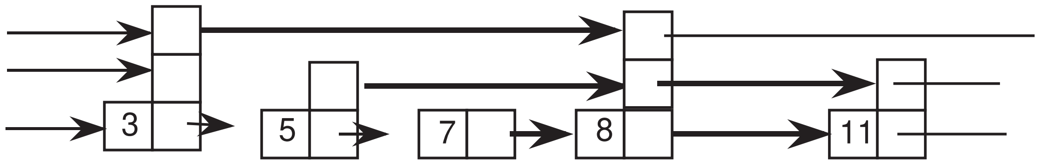
That is, for any i, level i is twice as likely as level $i+1$.

- To search, follow the highest level pointer first, find the proper interval, then go down, ...

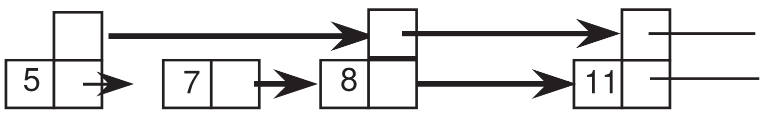
- If the node is assigned level k, then hook up the pointers to it for all levels from 0 to k

- To delete, just delete the node and maintain the pointers properly.





(a)



(b)

- The dictionary operations "insert", "delete", "search" all take $O(\log n)$ expected time with skip list.

  (Pugh, 1990, Munro, Papadakis, Sedgewick, 1992)

- Experimental evidence suggests that skip lists are competitive to AVL trees. They are also easier to implement.

- Worst Case: $O(n)$.