Changing the Model

What are factors that might make binary search either unusable or not optimal?

- We know something about the distribution.
- Data are not sorted. (Preprocessing?)
- Data sorted, but probes not all the same cost (not an array).
- Data are static, know all search requests in advance.

Interpolation Search

(Also known as Dictionary Search)

Search L at a position that is appropriate to the value of X.

$$p = \frac{X - L[1]}{L[n] - L[1]}$$

Repeat as necessary to recalculate p for future searches.

Quadratic Binary Search

This is easier to analyze:

Compute p and examine $L[\lceil pn \rceil]$.

If $X < L[\lceil pn \rceil]$ then sequentially probe

$$L[\lceil pn - i\sqrt{n} \rceil], i = 1, 2, 3, ...$$

until we reach a value less than or equal to X.

Similar for $X > L[\lceil pn \rceil]$.

We are now within $\sqrt{n}$ positions of X.

ASSUME (for now) that this takes a constant number of comparisons.

Now we have a sublist of size $\sqrt{n}$.

Repeat the process recursively.

What is the cost?

QBS Probe Count

Cost is $\Theta(\log \log n)$ IF the number of probes on jump search is constant.

Number of comparisons needed is:

$$\sum_{i=1}^{\sqrt{n}} i\mathbf{P}(\text{need exactly } i \text{ probes})$$

$$= 1\mathbf{P}_1 + 2\mathbf{P}_2 + 3\mathbf{P}_3 + \cdots + \sqrt{n}\mathbf{P}_{\sqrt{n}}$$

This is equal to:

$$\sum_{i=1}^{\sqrt{n}} \mathbf{P}(\text{need at least } i \text{ probes})$$

$$\begin{aligned}
&= 1 + (1 - \mathbf{P}_1) + (1 - \mathbf{P}_1 - \mathbf{P}_2) + \cdots + \mathbf{P}_{\sqrt{n}} \\
&= (\mathbf{P}_1 + \ldots + \mathbf{P}_{\sqrt{n}}) + (\mathbf{P}_2 + \ldots + \mathbf{P}_{\sqrt{n}}) + \\
&\qquad (\mathbf{P}_3 + \ldots + \mathbf{P}_{\sqrt{n}}) + \cdots \\
&= 1\mathbf{P}_1 + 2\mathbf{P}_2 + 3\mathbf{P}_3 + \cdots + \sqrt{n}\mathbf{P}_{\sqrt{n}}
\end{aligned}$$

QBS Probe Count (cont.)

We require at least two probes to set the bounds, so cost is:

$$2 + \sum_{i=3}^{\sqrt{n}} \mathbf{P}(\text{need at least } i \text{ probes})$$

Useful fact (Čebyšev's Inequality):

The probability that we need probe i times ($\mathbf{P}_i$) is:

$$\mathbf{P}_i \leq \frac{p(1-p)n}{(i-2)^2 n} \leq \frac{1}{4(i-2)^2}$$

since $p(1-p) \leq 1/4$.

This assumes uniformly distributed data.

Final result:

$$2 + \sum_{i=3}^{\sqrt{n}} \frac{1}{4(i-2)^2} \approx 2.4112$$

Is this better than binary search?

What happened to our proof that binary search is optimal?

Comparison

Let's compare $\log\log n$ to $\log n$.

n	$\log n$	$\log\log n$	Diff
16	4	2	2
256	8	3	2.7
$64K$	16	4	4
2^{32}	32	5	6.4

Now look at the actual comparisons used.

- Binary search $\approx \log n - 1$
- Interpolation search $\approx 2.4 \log\log n$

n	$\log n - 1$	$2.4 \log\log n$	Diff
16	3	4.8	worse
256	7	7.2	$\approx$ same
$64K$	15	9.6	1.6
2^{32}	31	12	2.6

Not done yet! This is only a count of comparisons!

- Which is more expensive: calculating the midpoint or calculating the interpolation point?

Which algorithm is dependent on good behavior by the input?

Hashing

Assume we can preprocess the data.

- How should we do it to minimize search?

Put record with key value X in $L[X]$.

If the range is too big, then use hashing.

How much can we get from this?

Simplifying assumptions:

- We hash to each slot with equal probability
- We probe to each (new) slot with equal probability
- This is called uniform hashing

Hashing Insertion Analysis

Define $\alpha = N/M$ (Records stored/Table size)

Insertion cost: sum of costs times probabilities for looking at 1, 2, ..., $N + 1$ slots

- Probability of collision on insertion? $\alpha = N/M$
- Probability of initial collision and another collision when probing? α^2

$$\sum_{i=0}^{i=N} i(\frac{N}{M})^i \frac{M-N}{M}$$

Simpler formulation: Always look at least once, look at least twice with probability α, look at least three times with probability α^2, etc.

$$\sum_{i=0}^{\infty} \alpha^i = 1 + \alpha + \alpha^2 \cdots = \frac{1}{1-\alpha}$$

How does this grow?

Searching Linked Lists

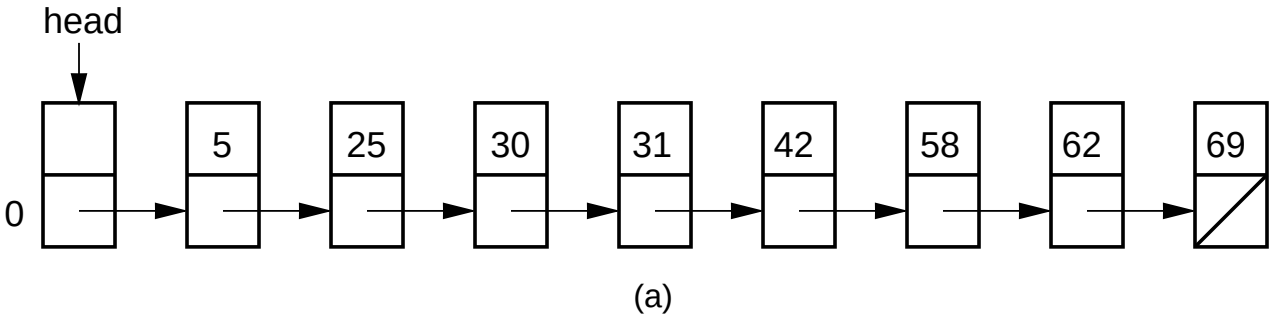
Assume the list is sorted, but is stored in a linked list.

Can we use binary search?

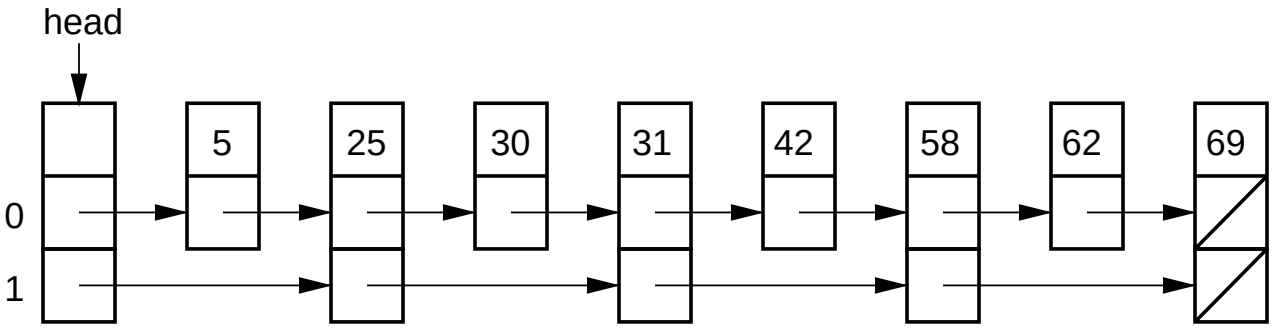
- Comparisons?
- "Work?"

What if we add additional pointers?

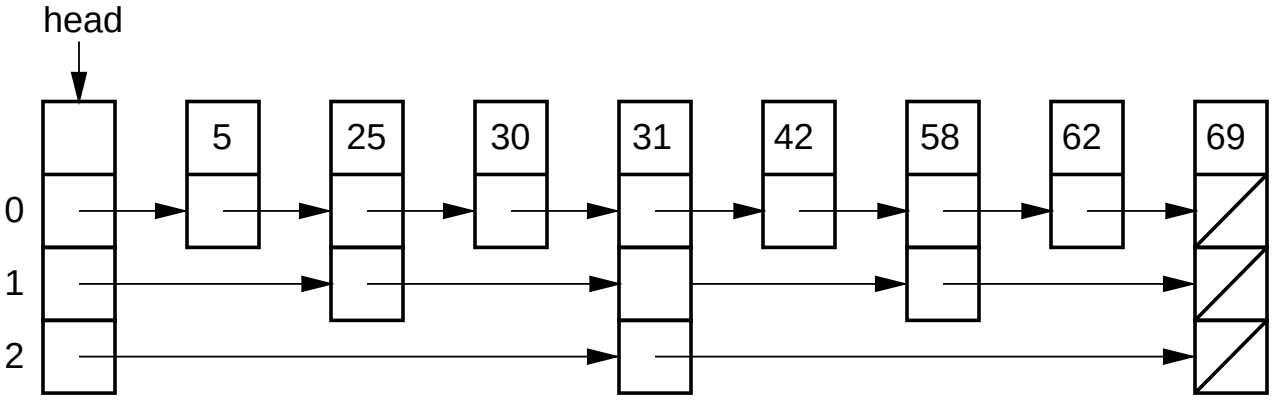
"Perfect" Skip List



(a)



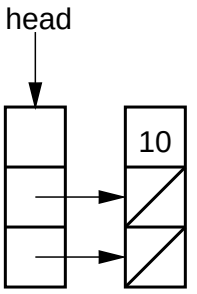
(b)



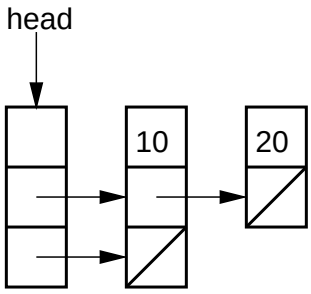
(c)

Building a Skip List

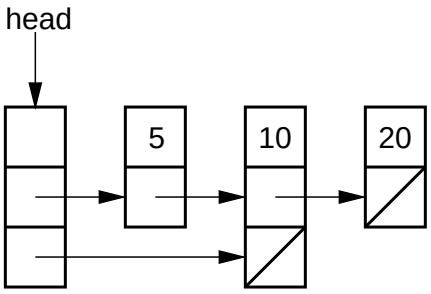
Pick the node size at random (from a suitable probability distribution).



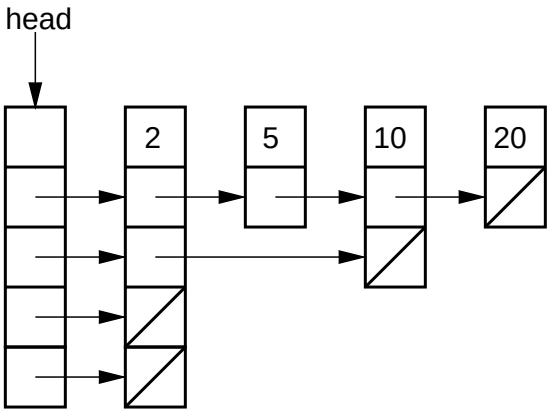
(a)



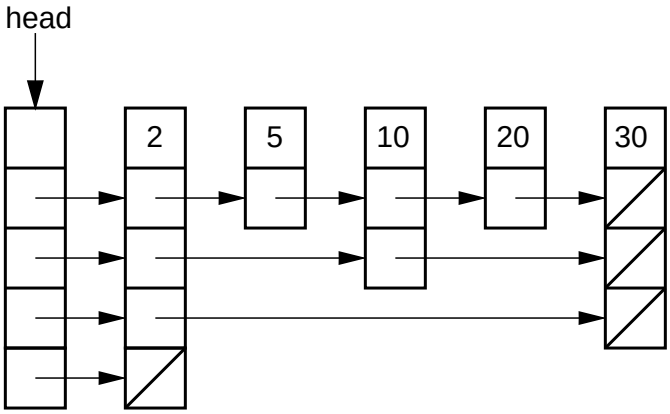
(b)



(c)



(d)



(e)

Skip List Analysis

What distribution do we want for the node depths?

```
int randomLevel(void) { // Exponential distribution
  for (int level=0; Random(2) == 0; level++); // No-op
  return level;
}
```

What is the worst cost to search in the "perfect" Skip List?

What is the average cost to search in the "perfect" Skip List?

What is the cost to insert?

What is the average cost in the "typical" Skip List?

How does this differ from a BST?

- Simpler or more complex?
- More or less efficient?
- Which relies on data distribution, which on basic laws of probability?

Other Types of Search

- Nearest neighbor (if X not in L).
- Exact Match Query.
- Range query.
- Multi-dimensional search.
- Is L static?

Is linear search on a sorted list ever better than binary search?