Outline

1. Dictionaries
   - Definitions
   - Dictionary operations
   - Dictionary implementation

2. Skip Lists
   - Why Skip Lists?
   - The Idea Behind All of It!!!
   - Skip List Definition
   - Skip list implementation
   - Insertion for Skip Lists
   - Deletion in Skip Lists
   - Properties
   - Search and Insertion Times
   - Applications
   - Summary
Dictionaries

Definition
A dictionary is a collection of elements; each of which has a unique search key.
- Uniqueness criteria may be relaxed (multi-set).
- Do not force uniqueness.

Purpose
Dictionaries keep track of current members, with periodic insertions and deletions into the set (similar to a database).

Examples
- Membership in a club.
- Course records.
- Symbol table (with duplicates).
- Language dictionary (Webster, RAE, Oxford).

Example: Course records

<table>
<thead>
<tr>
<th>key ID</th>
<th>Student Name</th>
<th>HW1</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Stan Smith</td>
<td>49</td>
</tr>
<tr>
<td>125</td>
<td>Sue Margolin</td>
<td>45</td>
</tr>
<tr>
<td>128</td>
<td>Billie King</td>
<td>24</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>190</td>
<td>Roy Miller</td>
<td>36</td>
</tr>
</tbody>
</table>
The dictionary ADT operations

Some operations on dictionaries
- size(): Returns the size of the dictionary.
- empty(): Returns TRUE if the dictionary is empty.
- findItem(key): Locates the item with the specified key.
- findAllItems(key): Locates all items with the specified key.
- removeItem(key): Removes the item with the specified key.
- removeAllItems(key): Removes all items with the specified key.
- insertItem(key,element): Inserts a new key-element pair.

Example of unordered dictionary

Consider an empty unordered dictionary, we have then...

<table>
<thead>
<tr>
<th>Operation</th>
<th>Dictionary</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>InsertItem(5, A)</td>
<td>{(5, A)}</td>
<td></td>
</tr>
<tr>
<td>InsertItem(7, B)</td>
<td>{(5, A), (7, B)}</td>
<td></td>
</tr>
<tr>
<td>findItem(7)</td>
<td>{(5, A), (7, B)}</td>
<td>B</td>
</tr>
<tr>
<td>findItem(4)</td>
<td>{(5, A), (7, B)}</td>
<td>No Such Key</td>
</tr>
<tr>
<td>size()</td>
<td>{(5, A), (7, B)}</td>
<td>2</td>
</tr>
<tr>
<td>removeItem(5)</td>
<td>{(7, B)}</td>
<td>A</td>
</tr>
<tr>
<td>findItem(4)</td>
<td>{(7, B)}</td>
<td>No Such Key</td>
</tr>
</tbody>
</table>
How to implement a dictionary?

There are many ways of implementing a dictionary

- Sequences / Arrays
  - Ordered
  - Unordered
- Binary search trees
- Skip lists
- Hash tables

Recall Arrays...

Unordered array

| 34 | 14 | 12 | 22 | 18 |

Complexity

- Searching and removing takes $O(n)$.
- Inserting takes $O(1)$.

Applications

This approach is good for log files where insertions are frequent but searches and removals are rare.
More Arrays

**Ordered array**

| 12 | 14 | 18 | 22 | 34 |

**Complexity**

- Searching takes $O(\log n)$ time (binary search).
- Insert and removing takes $O(n)$ time.

**Applications**

This approach is good for look-up tables where searches are frequent but insertions and removals are rare.

---

Binary searches

**Features**

- Narrow down the search range in stages
- "High-low" game.
Binary searches

Example find Element(22)

Recall binary search trees

Implement a dictionary with a BST

A binary search tree is a binary tree $T$ such that:
- Each internal node stores an item $(k, e)$ of a dictionary.
- Keys stored at nodes in the left subtree of $v$ are less than or equal to $k$.
- Keys stored at nodes in the right subtree of $v$ are greater than or equal to $k.$
Binary searches Trees

Problem!!! Keeping a Well Balanced Binary Search Tree can be difficult!!!

Not only that...

Binary Search Trees

- They are not so well suited for parallel environments.
  - Unless a heavy modifications are done

In addition

- We want to have a
  - Compact Data Structure.
  - Using as little memory as possible
Thus, we have the following possibilities

**Unordered array complexities**
- **Insertion:** $O(1)$
- **Search:** $O(n)$

**Ordered array complexities**
- **Insertion:** $O(n)$
- **Search:** $O(n \log n)$

**Well balanced binary trees complexities**
- **Insertion:** $O(\log n)$
- **Search:** $O(\log n)$

**Big Drawback - Complex parallel Implementation and waste of memory.**

We want something better!!!

For this

**We will present a probabilistic data structure known as Skip List!!!**
Starting from Scratch

First
- Imagine that you only require to have searches.
- A first approximation to it is the use of a link list for it (Θ(n) search complexity).
- Then, using this How do we speed up searches?

Something Notable
- Use two link list, one a subsequence of the other.

Imagine the two lists as a road system
- The Bottom is the normal road system, L₂.
- The Top is the high way system, L₁.

Example

High-Bottom Way System

L₁

14 --- 34 --- 42

L₂

14 --- 23 --- 34 --- 42 --- 47 --- 63
Thus, we have...

The following rule

To Search first search in the top one ($L_1$) as far as possible, then go down and search in the bottom one ($L_2$).

We can use a little bit of optimization

We have the following worst cost

$$\text{Search Cost High-Bottom Way System} = \text{Cost Searching Top} + \text{Cost Search Bottom}$$

Or

$$\text{Search Cost} = \text{length} (L_1) + \text{Cost Search Bottom}$$

The interesting part is "Cost Search Bottom"

This can be calculated by the following quotient:

$$\frac{\text{length} (L_2)}{\text{length} (L_1)}$$
**Why?**

If we think we are jumping

Then cost of searching each of the bottom segments = 2

Thus the ratio is a “decent” approximation to the worst case search

\[
\frac{\text{length}(L_2)}{\text{length}(L_1)} = \frac{5}{3} = 1.66
\]

Thus, we have...

Then, the cost for a search (when \(\text{length}(L_2) = n\))

**Search Cost**

\[
\text{Search Cost} = \text{length}(L_1) + \frac{\text{length}(L_2)}{\text{length}(L_1)} = \text{length}(L_1) + \frac{n}{\text{length}(L_1)}
\]

(1)

Taking the derivative with respect to \(\text{length}(L_1)\) and making the result equal 0

\[
1 - \frac{n}{\text{length}^2(L_1)} = 0
\]
Final Cost

We have that the optimal length for $L_1$

$$\text{length}(L_1) = \sqrt{n}$$

Plugging back in (Eq. 1)

$$\text{Search Cost} = \sqrt{n} + \frac{n}{\sqrt{n}} = \sqrt{n} + \sqrt{n} = 2 \times \sqrt{n}$$

Data structure with a Square Root Relation

Something like this
Now

For a three layer link list data structure, we get a search cost of $3 \times \sqrt[3]{n}$.

In general for $k$ layers, we have $k \times \sqrt[3]{n}$.

Thus, if we make $k = \log_2 n$, we get

\[
\text{Search Cost} = \log_2 n \times \log_2 \sqrt[3]{n} \\
= \log_2 n \times (n)^{1/\log_2 n} \\
= \log_2 n \times (n)^{\log_n 2} \\
= \log_2 n \times 2 \\
= \Theta (\log_2 n)
\]

Thus

Something Notable

We get the advantages of the binary search trees with a simpler architecture!!!
Thus

Binary Search Trees

New Architecture

$L_1$ 14 34 42
$L_2$ 14 23 34 42 47 63

Now

We are ready to give a

Definition for Skip List
A Little Bit of History

Skip List
They were invented by William Worthington "Bill" Pugh Jr.!!!

How is him?
- He is is an American computer scientist who invented the skip list and the Omega test for deciding Presburger arithmetic.
- He was the co-author of the static code analysis tool FindBugs.
- He was highly influential in the development of the current memory model of the Java language together with his PhD student Jeremy Manson.

Skip List Definition

Definition
A skip list for a set $S$ of distinct (key,element) items is a series of lists $S_0, S_1, ..., S_h$ such that:
- Each list $S_i$ contains the special keys $+\infty$ and $-\infty$
- List $S_0$ contains the keys of $S$ in nondecreasing order
- Each list is a subsequence of the previous one
  - $S_0 \supseteq S_1 \supseteq S_2 \supseteq ... \supseteq S_h$
- List $S_h$ contains only the two special keys
Skip list search

We search for a key $x$ in a skip list as follows:

- We start at the first position of the top list.
- At the current position $p$, we compare $x$ with $y == p.next.key$
  - $x == y$: we return $p.next.element$
  - $x > y$: we scan forward
  - $x < y$: we “drop down”
- If we try to drop down past the bottom list, we return null.
### Example search for 78

#### $x < p.next.key$: “drop down”

<table>
<thead>
<tr>
<th>$S_3$</th>
<th>$S_2$</th>
<th>$S_1$</th>
<th>$S_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\infty$</td>
<td>$12$</td>
<td>$23$</td>
<td>$\infty$</td>
</tr>
<tr>
<td></td>
<td>$26$</td>
<td>$31$</td>
<td>$44$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$34$</td>
<td>$64$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$64$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$78$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

#### $x > p.next.key$: “scan forward”

<table>
<thead>
<tr>
<th>$S_3$</th>
<th>$S_2$</th>
<th>$S_1$</th>
<th>$S_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\infty$</td>
<td>$12$</td>
<td>$23$</td>
<td>$\infty$</td>
</tr>
<tr>
<td></td>
<td>$26$</td>
<td>$31$</td>
<td>$44$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$34$</td>
<td>$56$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$64$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$78$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\infty$</td>
</tr>
</tbody>
</table>
Example search for 78

\[ x < p . \text{next.key}: \text{“drop down”} \]

\[
S_3 \rightarrow \infty & \rightarrow \infty \\
S_2 \rightarrow \infty & \rightarrow 31 & \rightarrow \infty \\
S_1 \rightarrow \infty & \rightarrow 23 & 31 & 34 & 64 & \rightarrow \infty \\
S_0 \rightarrow \infty & \rightarrow 12 & 23 & 26 & 31 & 34 & 44 & 56 & 64 & 78 & \rightarrow \infty
\]

Example search for 78

\[ x > p . \text{next.key}: \text{“scan forward”} \]

\[
S_3 \rightarrow \infty & \rightarrow \infty \\
S_2 \rightarrow \infty & \rightarrow 31 & \rightarrow \infty \\
S_1 \rightarrow \infty & \rightarrow 23 & 31 & 34 & 64 & \rightarrow \infty \\
S_0 \rightarrow \infty & \rightarrow 12 & 23 & 26 & 31 & 34 & 44 & 56 & 64 & 78 & \rightarrow \infty
\]
**Example search for 78**

\[ x > p.\text{next.key}: \text{“scan forward”} \]

\[
\begin{array}{c|cccccc}
S_3 & \infty & & & & +\infty \\
S_2 & \infty & 31 & p & & +\infty \\
S_1 & \infty & 23 & 31 & 34 & 64 & +\infty \\
S_0 & \infty & 12 & 23 & 26 & 31 & 34 & 44 & 56 & 64 & 78 & +\infty \\
\end{array}
\]

**Example search for 78**

\[ x < p.\text{next.key}: \text{“drop down”} \]

\[
\begin{array}{c|cccccc}
S_3 & \infty & & & & +\infty \\
S_2 & \infty & 31 & & & +\infty \\
S_1 & \infty & 23 & 31 & 34 & 64 & +\infty \\
S_0 & \infty & 12 & 23 & 26 & 31 & 34 & 44 & 56 & 64 & 78 & +\infty \\
\end{array}
\]
Example search for 78

\[
x == y: \text{we return } p.\text{next.element}
\]

How do we implement this data structure?

We can implement a skip list with quad-nodes

A quad-node stores:
- Entry Value
- Link to the previous node
- Link to the next node
- Link to the above node
- Link to the below node

Also we define special keys PLUS-INF and MINUS-INF, and we modify the key comparator to handle them.
Skip lists uses Randomization

**Use of randomization**
We use a randomized algorithm to insert items into a skip list.

**Running time**
We analyze the expected running time of a randomized algorithm under the following assumptions:
- The coins are unbiased.
- The coin tosses are independent.

**Worst case running time**
The worst case running time of a randomized algorithm is often large but has very low probability.
- e.g. It occurs when all the coin tosses give “heads.”
Insertion

To insert an entry \((key, object)\) into a skip list, we use a randomized algorithm:

- We repeatedly toss a coin until we get tails:
  - We denote with \(i\) the number of times the coin came up heads.
- If \(i \geq h\), we add to the skip list new lists \(S_{h+1}, \ldots, S_{i+1}\):
  - Each containing only the two special keys.
- We search for \(x\) in the skip list and find the positions \(p_0, p_1, \ldots, p_i\) of the items with largest key less than \(x\) in each lists \(S_0, S_1, \ldots, S_i\).
- For \(j \leftarrow 0, \ldots, i\), we insert item \((key, object)\) into list \(S_j\) after position \(p_j\).

Example: Insertion of 15 in the skip list

First, we use \(i = 2\) to insert \(S_3\) into the skip list:

\[
\begin{array}{c}
S_3 & - \infty & + \infty \\
S_2 & - \infty & + \infty \\
S_1 & - \infty & 23 & + \infty \\
S_0 & - \infty & 12 & 23 & 26 & + \infty
\end{array}
\]
Example: Insertion of 15 in the skip list

Clearly, you first search for the predecessor key!!!

Example: Insertion of 15 in the skip list

Insert the necessary Quad-Nodes and necessary information
Example: Insertion of 15 in the skip list

Insert the necessary Quad-Nodes and necessary information

\[
\begin{array}{c|c|c|c|c}
S_3 & -\infty & & +\infty \\
S_2 & -\infty & & +\infty \\
S_1 & -\infty & pred & 23 & +\infty \\
S_0 & -\infty & 12 & 15 & 23 & 26 & +\infty \\
\end{array}
\]
Example: Insertion of 15 in the skip list

Finally!!!

Deletion

To remove an entry with key $x$ from a skip list, we proceed as follows:

- We search for $x$ in the skip list and find the positions $p_0, p_1, \ldots, p_i$ of the items with key $x$, where position $p_j$ is in list $S_j$.
- We remove positions $p_0, p_1, \ldots, p_i$ from the lists $S_0, S_1, \ldots, S_i$.
- We remove all but one list containing only the two special keys.
Example: Delete of 34 in the skip list

We search for 34 in the skip list and find the positions \( p_0, p_1, \ldots, p_2 \) of the items with key 34.

Notes
Example: Delete of 34 in the skip list

We start doing the deletion!!

Example: Delete of 34 in the skip list

One Quad-Node after another
Example: Delete of 34 in the skip list

One Quad-Node after another

- $S_3$: $-\infty$ to $+\infty$
- $S_2$: $-\infty$ to $+\infty$
- $S_1$: $-\infty$ to $23$, $23$ to $+\infty$
- $S_0$: $-\infty$ to $12$, $12$ to $34$, $34$ to $45$, $45$ to $+\infty$

Example: Delete of 34 in the skip list

One Quad-Node after another

- $S_3$: $-\infty$ to $+\infty$
- $S_2$: $-\infty$ to $+\infty$
- $S_1$: $-\infty$ to $23$, $23$ to $+\infty$
- $S_0$: $-\infty$ to $12$, $12$ to $23$, $23$ to $45$, $45$ to $+\infty$
Example: Delete of 34 in the skip list

| S_2 | -∞   |       | +∞   |
| S_1 | -∞   | 23    | +∞   |
| S_0 | -∞   | 12    | 23    | 45    | +∞   |

Space usage

The space used by a skip list depends on the random bits used by each invocation of the insertion algorithm.
Theorem
The expected space usage of a skip list with \( n \) items is \( O(n) \).

Proof
We use the following two basic probabilistic facts:

1. **Fact 1**: The probability of getting \( i \) consecutive heads when flipping a coin is \( \frac{1}{2^i} \).
2. **Fact 2**: If each of \( n \) entries is present in a set with probability \( p \), the expected size of the set is \( np \).

Now consider a skip list with \( n \) entries
Using Fact 1, an element is inserted in list \( S_i \) with a probability of
\[
\frac{1}{2^i}
\]

Now by Fact 2
The expected size of list \( S_i \) is
\[
\frac{n}{2^i}
\]
Proof

The expected number of nodes used by the skip list with height $h$

$$\sum_{i=0}^{h} \frac{n}{2^i} = n \sum_{i=0}^{h} \frac{1}{2^i}$$

Here, we have a problem!!! What is the value of $h$?

Height $h$

First

The running time of the search and insertion algorithms is affected by the height $h$ of the skip list.

Second

We show that with high probability, a skip list with $n$ items has height $O(\log n)$.
For this, we have the following fact!!!

We use the following Fact 3:

We can view the level \( l(x_i) = \max \{j|\text{where } x_i \in S_j\} \) of the elements in the skip list as the following random variable

\[
X_i = l(x_i)
\]

for each element \( x_i \) in the skip list.

And this is a random variable!!!

- Remember the insertions!!! Using an unbiased coin!!
- Thus, all \( X_i \) have a geometric distribution.

Example for \( l(x_i) \)

We have

\[
\begin{align*}
  l(x) & \quad 1 \quad 2 \quad 3 \quad 1 \\
  S_3 & \quad -\infty \quad +\infty \\
  S_2 & \quad -\infty \quad \quad 34 \quad +\infty \\
  S_1 & \quad -\infty \quad 23 \quad 34 \quad +\infty \\
  S_0 & \quad -\infty \quad 12 \quad 23 \quad 34 \quad 45 \quad +\infty
\end{align*}
\]
BTW What is the geometric distribution?

$k$ failures where

\[ k = \{1, 2, 3, \ldots\} \]

Probability mass function

\[ Pr(X = k) = (1 - p)^{k-1} p \]
Then

We have the following inequality for the geometric variables

\[ Pr [X_i > t] \leq (1 - p)^t \quad \forall i = 1, 2, ..., n \]

Because if the cdf \( F(t) = P(X \leq t) = 1 - (1 - p)^{t+1} \)

Then, we have

\[ Pr \left\{ \max_i X_i > t \right\} \leq n(1 - p)^t \]

This comes from \( F_{\max, X_i}(t) = (F(t))^n \)

Observations

The \( \max_i X_i \)

It represents the list with the one entry apart from the special keys.

- \( S_3 \) - \( -\infty \) - \( +\infty \)
- \( S_2 \) - \( -\infty \) - 34 - \( +\infty \)
- \( S_1 \) - \( -\infty \) - 23 - 34 - \( +\infty \)
- \( S_0 \) - \( -\infty \) - 12 - 23 - 34 - 45 - \( +\infty \)
Observations

**REMEMBER!!!**
We are talking about a fair coin, thus $p = \frac{1}{2}$.

**Height:** $3 \log_2 n$ with probability at least $1 - \frac{1}{n^2}$

**Theorem**
A skip list with $n$ entries has height at most $3 \log_2 n$ with probability at least $1 - \frac{1}{n^2}$.
Proof

Consider a skip list with $n$ entries.
By Fact 3, the probability that list $S_i$ has at least one item is at most $\frac{n}{2^i}$.

$$P(|S| \geq 1) = P(\max_i X_i > t) = \frac{n}{2^i}.$$ 

By picking $t = 3 \log n$
We have that the probability that $S_{3 \log_2 n}$ has at least one entry is at most:

$$\frac{n}{2^{3 \log_2 n}} = \frac{n}{n^3} = \frac{1}{n^2}.$$ 

Look at we want to model

We want to model
- The height of the Skip List is at most $t = 3 \log n$
- Equivalent to the negation of having list $S_{3 \log_2 n}$

Then, the probability that the height $h = 3 \log_2 n$ of the skip list is

$$P(\text{Skip List height } 3 \log_2 n) = 1 - \frac{1}{n^2}$$

Notes
Finally

The expected number of nodes used by the skip list with height $h$

Given that $h = 3 \log_2 n$

$$\sum_{i=0}^{3 \log_2 n} \frac{n}{2^i} = n \sum_{i=0}^{3 \log_2 n} \frac{1}{2^i}$$

Given the geometric sum

$$S_m = \sum_{k=0}^{m} r^k = \frac{1 - r^{m+1}}{1 - r}$$

We have finally

The Upper Bound on the number of nodes

$$n \sum_{i=0}^{3 \log_2 n} \frac{1}{2^i} = n \left( \frac{1 - \left( \frac{1}{2} \right)^{3 \log_2 n+1}}{1 - \frac{1}{2}} \right)$$

$$= n \left( \frac{1 - \left( \frac{1}{2} \left( \frac{1}{2} \log_2 n \right)^3 \right)}{1/2} \right)$$

We have then

$$\frac{1}{2^{\log_2 n}} = \frac{1}{n}$$

Then

$$n \left( \frac{1 - \frac{1}{2} \left( \frac{1}{2} \log_2 n \right)^3}{1/2} \right) = n \left( \frac{1 - \frac{1}{2 n^2}}{1/2} \right) = n \left( 2 - \frac{1}{2 n^2} \right) = 2n - \frac{1}{2n}$$
Finally

The Upper Bound with probability $1 - \frac{1}{n^2}$

$$2n - \frac{1}{2n} \leq 2n = O(n)$$

Search and Insertion Times

Something Notable

The expected number of coin tosses required in order to get tails is 2.

We use this

To prove that a search in a skip list takes $O(\log n)$ expected time.

- After all insertions require searches!!!
Search and Insertions times

**Search time**
The search time in skip list is proportional to

the number of drop-down steps + the number of scan-forward steps

**Drop-down steps**
The drop-down steps are bounded by the height of the skip list and thus are $O(\log_2 n)$ with high probability.

**Theorem**
A search in a skip list takes $O(\log_2 n)$ expected time.

**Proof**

First
When we scan forward in a list, the destination key does not belong to a higher list.

A scan-forward step is associated with a former coin toss that gave tails.

By Fact 4, in each list the expected number of scan-forward steps is 2.
Why?

Given the list $S_i$
Then, the scan-forward intervals (Jumps between $x_i$ and $x_{i+1}$) to the right of $S_i$ are

\[ I_1 = [x_1, x_2], I_2 = [x_2, x_3], \ldots, I_k = [x_k, +\infty] \]

These intervals exist at level $i$ if and only if all $x_1, x_2, \ldots, x_k$ belong to $S_i$.

We introduce the following concept based on these intervals

Scan-forward siblings
These are elements that you find in the search path before finding an element in the upper list.
Given that a search is being done, \( S_i \) contains \( l \) forward siblings. It must be the case that given \( x_1, ..., x_l \) scan-forward siblings, we have that:

\[
x_1, ..., x_l \notin S_{i+1}
\]

and \( x_{l+1} \in S_{i+1} \).

Thus

We have

Since each element of \( S_i \) is independently chosen to be in \( S_{i+1} \) with probability \( p = \frac{1}{2} \).

We have

The number of scan-forward siblings is bounded by a geometric random variable \( X_i \) with parameter \( p = \frac{1}{2} \).

Thus, we have that

The expected number of scan-forward siblings is bounded by 2!!!

\[
\text{Expected # Scan-Forward Siblings at } i \leq E[X_i] = \frac{1}{1/2} = 2
\]

Mean
In the worst case scenario

A search is bounded by $2 \log_2 n = O(\log_2 n)$

An given that a insertion is a (search) + (deletion bounded by the height)

Thus, an insertion is bounded by $2 \log_2 n + 3 \log n = O(\log_2 n)$

Applications

We have

- Cyrus IMAP servers offer a "skiplist" backend Data Base implementation.
- Lucene uses skip lists to search delta-encoded posting lists in logarithmic time.
- Redis, an ANSI-C open-source persistent key/value store for Posix systems, uses skip lists in its implementation of ordered sets.
- leveldb, a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values.
- Skip lists are used for efficient statistical computations of running medians.
A skip list is a data structure for dictionaries that uses a randomized insertion algorithm. In a skip list with \( n \) entries:
- The expected space used is \( O(n) \)
- The expected search, insertion and deletion time is \( O(\log n) \)

Skip lists are fast and simple to implement in practice.