Skip lists
Consider a lottery with 1000 tickets and the following prizes.

- 1 ticket wins 100$.
- 9 ticket win 10$ each.
- 90 ticket win 5$ each.
- The remaining tickets do not have prizes.

The cost of a ticket is 1$. Is it worthwhile to participate?

The sum of the prizes is

\[ 100 \cdot 1 + 10 \cdot 9 + 5 \cdot 90 + 0 \cdot 900 = 640 < 1000 \]

The average win is

\[ \frac{100 \cdot 1 + 10 \cdot 9 + 5 \cdot 90 + 0 \cdot 900}{1000} = 0.64 < 1 \]
The previous computation can be done using probability theory.

- The probability to win 100$ is 1/1000.
- The probability to win 10$ is 9/1000.
- The probability to win 5$ is 90/1000.
- The probability to win 0$ is 900/1000.

The expected win is

$$100 \cdot \frac{1}{1000} + 10 \cdot \frac{9}{1000} + 5 \cdot \frac{90}{1000} + 0 \cdot \frac{900}{1000} = 0.64$$
A probability space is a pair \((\Omega, P)\) where
- \(\Omega\) is a set of outcomes. \(\Omega\) is called sample space.
- \(P : \Omega \to [0, 1]\) is a function that satisfies \(\sum_{x \in \Omega} P(x) = 1\). \(P\) is called probability measure.
- An event is a subset of \(\Omega\).
- For an event \(A \subseteq \Omega\), \(\Pr[A] = \sum_{x \in A} P(x)\).

**Example**

The lottery example can be modeled by the following probability space.
- \(\Omega = \{1, 2, \ldots, 1000\}\).
- \(P(x) = 1/1000\) for all \(x \in \Omega\).
- \(A_1 = \{1\}\) (the event of winning 100$)
- \(A_2 = \{2, 3, \ldots, 9\}\) (the event of winning 10$), etc.
- \(\Pr[A_1] = 1/1000\), \(\Pr[A_2] = 9/1000\).
A random variable is a function \( X : \Omega \rightarrow R \).

The expectation of a random variable \( X \) is

\[
E[X] = \sum_k k \cdot \Pr[X = k].
\]

Example

\[
X(y) = \begin{cases} 
100 & \text{if } y = 1 \\
10 & \text{if } 2 \leq y \leq 10 \\
5 & \text{if } 11 \leq y \leq 100 \\
0 & \text{otherwise}
\end{cases}
\]

\[
E[X] = 100 \cdot \frac{1}{1000} + 10 \cdot \frac{9}{1000} + 5 \cdot \frac{90}{1000} + 0 \cdot \frac{900}{1000} = 0.64
\]
Suppose we toss a coin 3 times. What is the expected number of times it lands on “head”?

- \( \Omega = \{ \text{HHH, HHT, HTH, HTT, THH, THT, TTH, TTT} \} \)
- \( P(x) = 1/8 \) for every \( x \in \Omega \).
- Let \( X(y) = \) number of ‘H’s in \( y \).
  For example, \( X(\text{HHH}) = 3 \), \( X(\text{HHT}) = 2 \).
- \( E[X] = 0 \cdot \frac{1}{8} + 1 \cdot \frac{3}{8} + 2 \cdot \frac{3}{8} + 3 \cdot \frac{1}{8} = \frac{3}{2} \).
An indicator random variable for an event $A$ is a random variable $X$ such that

$$X(y) = \begin{cases} 1 & \text{if } y \in A \\ 0 & \text{otherwise} \end{cases}$$

If $X$ is an indicator random variable for an event $A$,

$$E[X] = 0 \cdot \Pr[X = 0] + 1 \cdot \Pr[X = 1] = \Pr[A]$$
Linearity of expectation

**Theorem**

For random variables $X_1, \ldots, X_k$, $E[\sum_{i=1}^{k} X_i] = \sum_{i=1}^{k} E[X_i]$. 

**Example**

Suppose we toss a coin 3 times. What is the expected number of times it lands on “head”? 

- Let $X =$ number of times the coin lands on “head”. 
- Let $X_i$ be an indicator random variable for the event that the coin lands on “head” in the $i$-th toss. 
- $X = X_1 + X_2 + X_3$, so 

$$E[X] = E[X_1] + E[X_2] + E[X_3] = \frac{1}{2} + \frac{1}{2} + \frac{1}{2} = \frac{3}{2}$$
A random variable $X$ with binomial distribution with parameters $n, p$ is the number of times a coin lands on "head" when the coin is tossed $n$ times, and the probability to land on "head" is $p$.

- $\Omega = \text{all H/T strings of length } n$.
- $P(y) = p^k(1 - p)^{n-k}$ where $k$ in the number of ‘H’s in $y$.
- $X(y) = \text{number of ‘H’s in } y$.

**Theorem**

$E[X] = np$. 
A random variable $X$ with geometric distribution is the number of coin tosses when a coin is tossed until the first “head”.

- $\Omega = \{H, \text{TH}, \text{TTH}, \ldots\}$.
- $P(H) = \frac{1}{2}$, $P(\text{TH}) = \frac{1}{4}$, $P(\text{TTH}) = \frac{1}{8}$, $\ldots$
- $X(y) = \text{length of } y$.

**Theorem**

$E[X] = 2$.

\[ E[X] = 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + \cdots = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots \]

\[ = \left(1 - \frac{1}{2}\right) \cdot \frac{1}{2} + \left(1 - \frac{1}{4}\right) \cdot \frac{1}{4} + \left(1 - \frac{1}{8}\right) \cdot \frac{1}{8} + \cdots = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 1 + \frac{1}{8} \cdot 1 + \cdots \]

\[ + \frac{1}{4} \cdot \frac{1}{4} + \frac{1}{8} \cdot \frac{1}{8} + \cdots = \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{4} \cdot \frac{1}{4} + \frac{1}{8} \cdot \frac{1}{8} + \cdots \]

\[ = \left(1 - \frac{1}{2}\right) + \left(1 - \frac{1}{4}\right) + \left(1 - \frac{1}{8}\right) + \cdots = \left(1 - \frac{1}{2}\right) + \left(1 - \frac{1}{4}\right) + \left(1 - \frac{1}{8}\right) + \cdots \]

\[ = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots \]
A **dynamic set ADT** is a structure that stores a set of elements. Each element has a (unique) key and satellite data. The structure supports the following operations.

**Search**($S$, $k$)  Return the element whose key is $k$.

**Insert**($S$, $x$)  Add $x$ to $S$.

**Delete**($S$, $x$)  Remove $x$ from $S$ (the operation receives a pointer to $x$).

**Minimum**($S$)  Return the element in $S$ with smallest key.

**Maximum**($S$)  Return the element in $S$ with largest key.

**Successor**($S$, $x$)  Return the element in $S$ with smallest key that is larger than $x$.key.

**Predecessor**($S$, $x$)  Return the element in $S$ with largest key that is smaller than $x$.key.
Skip list is an implementation of dynamic set with the following complexities:

- Insert: $\Theta(\log n)$ expected.
- Delete: $\Theta(1)$ expected.
- Minimum, Maximum, Successor, Predecessor: $\Theta(1)$ (worst case).
A skip list for a set $S$ consists of sorted linked lists $S_0, S_1, \ldots, S_h$ such that:

- Each list $S_i$ contains dummy elements $-\infty$ and $\infty$.
- $S_0$ contains all the elements of $S$.
- $S_i$ is a sublist of $S_{i-1}$.
- $S_h$ contains only the elements $-\infty$ and $\infty$. 

```
-\infty \quad 12 \quad 17 \quad 20 \quad 25 \quad 31 \quad 38 \quad 44 \quad 50 \quad 55 \quad \infty
```
Skip list

- Each node stores 4 pointers: next, prev, below, above.
- The structure stores a pointer \( S.\text{topleft} \) to the first element in \( S_h \).
- For efficient Minimum/Maximum, the structure stores pointers to the first/last element in \( S_0 \).
Assume that $\text{Find}(S, k)$ is a procedure that returns the element in $S$ with largest key that is at most $k$.

**Search**($S, k$)

1. $p \leftarrow \text{Find}(S, k)$
2. if $p.\text{key} = k$
3. \hspace{1em} return $p$
4. else
5. \hspace{1em} return NULL
**Find**

\[ \text{FIND}(S, k) \]

1. \( p \leftarrow S.\text{topleft} \)
2. \( \text{while } p.\text{below} \neq \text{NULL} \)
3. \( p \leftarrow p.\text{below} \)
4. \( \text{while } p.\text{next.key} \leq k \)
5. \( p \leftarrow p.\text{next} \)
6. \( \text{return } p \quad // p.\text{key} \text{ is largest key that is } \leq k \)

---

**Diagram:**

```
-∞ --> 17 --> 20 --> 25 --> 31 --> 38 --> 44 --> 50 --> 55 --> ∞
```

---

**Note:**

- The diagram represents a skip list structure.
- Each level represents a list, with each level being a sub-list of the one below it.
- The key at each level is an element of the skip list.
- The arrows indicate the direction of traversal.

---

**Skip lists:**

- Skip lists are a probabilistic abstract data type used for searching data structures.
- They allow for fast search, insertion, and deletion operations.
- The structure is composed of multiple sorted linked lists, each level being a sub-list of the previous one.
- The search operation involves traversing these levels, thereby skipping irrelevant data, hence the name 'skip list'.
**Find**

\[ \text{FIND}(S, k) \]

1. \( p \leftarrow S\text{.topleft} \)
2. \( \text{while } p\text{.below} \neq \text{NULL} \)
3. \( p \leftarrow p\text{.below} \)
4. \( \text{while } p\text{.next.key} \leq k \)
5. \( p \leftarrow p\text{.next} \)
6. \( \text{return } p \quad // \quad p\text{.key} \text{ is largest key that is } \leq k \)

**Find(S,50)**
**FIND** (*S*, *k*)

1. \( p \leftarrow S\).topleft
2. while \( p\).below \( \neq \) NULL
3. \( p \leftarrow p\).below
4. while \( p\).next.key \( \leq \) *k*
5. \( p \leftarrow p\).next
6. return \( p \)  // \( p\).key is largest key that is \( \leq \) *k*

Find(*S*, 50)
**Find**

\begin{align*}
\textbf{FIND}(S, k) \\
(1) & \quad p \leftarrow S.\text{topleft} \\
(2) & \quad \textbf{while } p.\text{below} \neq \text{NULL} \\
(3) & \quad p \leftarrow p.\text{below} \\
(4) & \quad \textbf{while } p.\text{next}.\text{key} \leq k \\
(5) & \quad p \leftarrow p.\text{next} \\
(6) & \quad \text{return } p \quad \text{ // } p.\text{key} \text{ is largest key that is } \leq k
\end{align*}
**Find**

**Find**($S, k$)

1. $p \leftarrow S$.topleft
2. **while** $p$.below $\neq$ NULL
3. $p \leftarrow p$.below
4. **while** $p$.next.key $\leq k$
5. $p \leftarrow p$.next
6. **return** $p$  // $p$.key is largest key that is $\leq k$

Find($S, 50$)
**Find**

\[\text{Find}(S, k)\]

1. \( p \leftarrow S\.\text{topleft} \)
2. \( \text{while } p\.\text{below} \neq \text{NULL} \)
3. \( p \leftarrow p\.\text{below} \)
4. \( \text{while } p\.\text{next\.key} \leq k \)
5. \( p \leftarrow p\.\text{next} \)
6. \( \text{return } p \quad \text{// } p\.\text{key is largest key that is } \leq k \)

\(\text{Find}(S, 50)\)
**Find**

**FIND**(S, k)

1. $p \leftarrow S\text{.topleft}$
2. **while** $p\text{.below} \neq \text{NULL}$
3. $p \leftarrow p\text{.below}$
4. **while** $p\text{.next}\text{.key} \leq k$
5. $p \leftarrow p\text{.next}$
6. **return** $p$  // $p\text{.key}$ is largest key that is $\leq k$

Find(S,50)
**Insertion**

**INSERT(S, x)**

1. \( p \leftarrow \text{FIND}(S, x.\text{key}) \)
2. Insert \( x \) after \( p \)
3. while random() < 1/2
4. \( \text{while } p.\text{above} = \text{NULL} \)
5. \( p \leftarrow p.\text{prev} \)
6. \( p \leftarrow p.\text{above} \)
7. Insert a copy \( y \) of \( x \) after \( p \)
8. if \( p = S.\text{topleft} \) crate a new top list

![Skip lists diagram](image-url)
**Insertion**

**Insert**($S, x$)

1. $p \leftarrow \text{FIND}(S, x.\text{key})$
2. Insert $x$ after $p$
3. while random() < 1/2
4. \hspace{1em} while $p.\text{above} = \text{NULL}$
5. \hspace{2em} $p \leftarrow p.\text{prev}$
6. \hspace{2em} $p \leftarrow p.\text{above}$
7. Insert a copy $y$ of $x$ after $p$
8. if $p = S.\text{topleft}$ create a new top list
**Insertion**

\[\text{INSERT}(S, x)\]

1. \[p \leftarrow \text{FIND}(S, x.\text{key})\]
2. Insert \(x\) after \(p\)
3. While random() < 1/2
4. While \(p.\text{above} = \text{NULL}\)
5. \[p \leftarrow p.\text{prev}\]
6. \[p \leftarrow p.\text{above}\]
7. Insert a copy \(y\) of \(x\) after \(p\)
8. If \(p = S.\text{topleft}\) create a new top list
**Insertion**

**Insert**($S, x$)

1. $p \leftarrow \text{FIND}(S, x.\text{key})$
2. Insert $x$ after $p$
3. **while** random() < 1/2
4. **while** $p.\text{above} = \text{NULL}$
5. $p \leftarrow p.\text{prev}$
6. $p \leftarrow p.\text{above}$
7. Insert a copy $y$ of $x$ after $p$
8. **if** $p = S.\text{topleft}$ crate a new top list
**Insertion**

**Insert**\((S, x)\)

1. \( p \leftarrow \text{FIND}(S, x.\text{key}) \)
2. Insert \( x \) after \( p \)
3. while \( \text{random()} < 1/2 \)
4. while \( p.\text{above} = \text{NULL} \)
5. \( p \leftarrow p.\text{prev} \)
6. \( p \leftarrow p.\text{above} \)
7. Insert a copy \( y \) of \( x \) after \( p \)
8. if \( p = S.\text{topleft} \) crate a new top list
**Insertion**

**Insert**($S, x$)

1. $p \leftarrow \text{FIND}(S, x.\text{key})$
2. Insert $x$ after $p$
3. while random() $< 1/2$
4. while $p.\text{above} = \text{NULL}$
5. $p \leftarrow p.\text{prev}$
6. $p \leftarrow p.\text{above}$
7. Insert a copy $y$ of $x$ after $p$
8. if $p = S.\text{topleft}$ create a new top list
**Insertion**

**Insert**($S, x$)

1. $p ← \text{FIND}(S, x.\text{key})$
2. Insert $x$ after $p$
3. while random() $< 1/2$
4. while $p.\text{above} = \text{NULL}$
5. $p ← p.\text{prev}$
6. $p ← p.\text{above}$
7. Insert a copy $y$ of $x$ after $p$
8. if $p = S.\text{topleft}$ create a new top list
**Delete**($S$, $x$)

1. **while** $x \neq \text{NULL}$
2. Remove $x$ from its list
3. **if** the list becomes empty, delete the list
4. $x \leftarrow x$.above

![Diagram of skip lists](image)
**Delete**

\[ \textbf{Delete}(S, x) \]

1. \textbf{while} \( x \neq \text{NULL} \)
2. Remove \( x \) from its list
3. \textbf{if} the list becomes empty, delete the list
4. \( x \leftarrow x.\text{above} \)
**DELETE**(*S, x*)

1. while *x* ≠ NULL
2. Remove *x* from its list
3. if the list becomes empty, delete the list
4. *x* ← *x*.above
Let $S$ be a skip list with $n$ elements.

The probability that an element $x$ is in the list $S_i$ is $1/2^i$.

The probability that $S_i$ has at least one element (excluding $\pm\infty$) is at most $n \cdot 1/2^i$.

The probability that $S_{3 \log n}$ has at least one element is at most $n/2^{3 \log n} = n/n^3 = 1/n^2$.

The probability that $S_{3 \log n}$ doesn’t have elements is at least $1 - 1/n^2$.

In other words, with high probability, the height of the skip list is $O(\log n)$. 
The expected space for the $\pm \infty$ elements is $O(\log n)$.

The probability that an element $x$ is in the list $S_i$ is $1/2^i$.

The size of $S_i$ (excluding $\pm \infty$) has binomial distribution with parameters $n$ and $1/2^i$.

Therefore, $E[|S_i|] = n/2^i$.

The expected size of the skip list is

$$O(\log n) + \sum_{i=0}^{\infty} \frac{n}{2^i} = O(\log n) + n \sum_{i=0}^{\infty} \frac{1}{2^i} = O(\log n) + 2n$$

The expected space of a skip list is $\Theta(n)$.
**Find**(*S, k*)

(1) \( p \leftarrow S\.\text{topleft} \)

(2) **while** \( p\.\text{below} \neq \text{NULL} \)

(3) \( p \leftarrow p\.\text{below} \)

(4) **while** \( p\.\text{next}.\text{key} \leq k \)

(5) \( p \leftarrow p\.\text{next} \)

(6) **return** \( p \)

The expected number of iterations of the while loop in line 2 is \( O(\log n) \).

The number of iterations of while loop in line 4 on one list \( S_i \) has geometric distribution, so the expected number of iterations is 2.

The expected total number of iterations of the while loop in line 4 is \( O(\log n) \).

The expected time is \( O(\log n) \).
Time complexity — deletion

- The time of deletion is linear in the height of the tower of the deleted element.
- The height of a tower has geometric distribution.
- The expected height of a tower is 2.
- The expected time of Delete is $\Theta(1)$. 