Consider a lottery with 1000 tickets and the following prizes.

- 1 ticket wins 100$.
- 9 tickets win 10$ each.
- 90 tickets 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.
- \(P : \Omega \rightarrow [0, 1]\) 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$)
\(A_2 = \{2, 3, \ldots, 10\}\) (the event of winning 10$), etc.
\(\Pr[A_1] = 1/1000\), \(\Pr[A_2] = 9/1000\).
For every two events $A$, $B$,

$$\Pr[A \cup B] = \Pr[A] + \Pr[B] - \Pr[A \cap B] \leq \Pr[A] + \Pr[B]$$

Therefore, for every events $A_1, \ldots, A_k$,

$$\Pr[A_1 \cup A_2 \cup \cdots \cup A_k] \leq \Pr[A_1] + \Pr[A_2] + \cdots + \Pr[A_k]$$
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**

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

$$E[X] = 100 \cdot \frac{1}{1000} + 10 \cdot \frac{900}{1000} + 5 \cdot \frac{90}{1000} + 0 \cdot \frac{900}{1000} = 0.64$$
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 $X(y) = \text{number of ‘H’s in } y$.
  - For example, $X(\text{HHH}) = 3$, $X(\text{HHT}) = 2$.
- $E[X] = 0 \cdot \frac{1}{8} + 1 \cdot \frac{3}{8} + 2 \cdot \frac{3}{8} + 3 \cdot \frac{1}{8} = \frac{3}{2}$. 

Skip lists
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 $A$,

$$E[X] = 0 \cdot \Pr[X = 0] + 1 \cdot \Pr[X = 1] = \Pr[A]$$
**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 $X = \text{number of times the coin lands on “head”}$.
- 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**

$E[X] = 2$.

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

$$+ \frac{1}{4} + \frac{1}{8} + \cdots \frac{1}{2}$$

$$+ \frac{1}{8} + \cdots \frac{1}{4}$$

...
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 \(k\) (return NULL if no element has key \(k\)).

**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:

- Search, 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 $\infty$. 

![Diagram of a skip list](image-url)
Skip list

- Each node stores 4 pointers: next, prev, below, above.
- The structure stores a pointer $S.topleft$ to the first element in $S_h$.
- For efficient Minimum/Maximum, the structure stores pointers to the first/last element in $S_0$. 

![Diagram of Skip List]
**Find**

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

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

**FIND**($S, k$)

1. $p \leftarrow S\.topleft$
2. while $p\.below \neq \text{NULL}$
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 $\leq k$

**Skip lists**
**Find**

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

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**

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

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

Diagram:

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)
**Find**

**FIND**\((S, k)\)

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

\[
\begin{array}{ccccccccc}
-\infty & & & & & & & & \infty \\
& & 17 & & & & & & \\
& & 17 & 20 & 25 & & & & \\
& & 17 & 20 & 25 & & & & \\
& 12 & 17 & 20 & 25 & 31 & 38 & 44 & 50 & \infty
\end{array}
\]

Find(S,50)
Search

\textbf{Search}(S, k)

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

#### Skip lists

-∞ -∞ -∞ -∞ -∞ -∞ 17 20 25 31 38 39 44 50 55 ∞ 12 17 20 25 31 38 39 44 55 ∞ 17 25 55 ∞ 17 ∞ ∞ 20 20 20
**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
**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

*Diagram of Skip lists*
**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. \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**

**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. \textbf{while} \( \text{random}() < 1/2 \)
4. \textbf{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. \textbf{if} \( p = S.\text{topleft} \) create 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** \(\text{random()} < 1/2\)
4. **while** \(p\).above = NULL
5. \(p \leftarrow p\).prev
6. \(p \leftarrow p\).above
7. Insert a copy \(y\) of \(x\) after \(p\)
8. **if** \(p = S\).topleft crate 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. $x \leftarrow x.\text{above}$
**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
**Delete**$(S, x)$

1. while $x \neq$ NULL
2. Remove $x$ from its list
3. if the list becomes empty, delete the list
4. $x \leftarrow x$.above
The height of skip list

- 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 $O(\log n)$. 
Space complexity

- 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, $E[|S_i|] = 2 + n/2^i$.
- The expected size of the skip list is

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

- The expected space of a skip list is $\Theta(n)$. 

Skip lists
Time complexity – search/insertion

\textbf{FIND}(S, k)

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

- 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 \( O(\log n) \).
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 $\Theta(1)$. 