Ordered Dictionary
An **ordered dictionary** stores a set $S$ whose elements are from an ordered universe $U$, and supports the following operations:

- **Insert**($S$, $x$) Insert an element $x$ to $S$.
- **Delete**($S$, $x$) Delete an element $x$ from $S$.
- **Successor**($S$, $x$) Return the smallest $s \in S$ s.t. $s \geq x$.
- **Predecessor**($S$, $x$) Return the largest $s \in S$ s.t. $s \leq x$.

**Example**

$S = \{1, 4, 8, 10\}$.

$\text{Successor}(S, 5) = 8$

$\text{Predecessor}(S, 5) = 4$
An ordered dictionary stores a set $S$ whose elements are from an ordered universe $U$, and supports the following operations:

- $\text{Insert}(S, x)$ Insert an element $x$ to $S$.
- $\text{Delete}(S, x)$ Delete an element $x$ from $S$.
- $\text{Successor}(S, x)$ Return the smallest $s \in S$ s.t. $s \geq x$.
- $\text{Predecessor}(S, x)$ Return the largest $s \in S$ s.t. $s \leq x$.

Using a balanced search tree, each of the operations above can be implemented in $\Theta(\log n)$ time, where $n$ is the current size of $S$. 
Ordered dictionary

An ordered dictionary stores a set $S$ whose elements are from an ordered universe $U$, and supports the following operations:

- **Insert**($S$, $x$) Insert an element $x$ to $S$.
- **Delete**($S$, $x$) Delete an element $x$ from $S$.
- **Successor**($S$, $x$) Return the smallest $s \in S$ s.t. $s \geq x$.
- **Predecessor**($S$, $x$) Return the largest $s \in S$ s.t. $s \leq x$.

- Using a balanced search tree, each of the operations above can be implemented in $\Theta(\log n)$ time, where $n$ is the current size of $S$.
- Suppose that $U = \{1, \ldots, u\}$ for “small” $u$. Can we do better in this case?
van Emde Boas tree

- If \( u \leq 3 \), store the elements of \( S \) in a linked list.
- Otherwise, partition \( U \) into \( \sqrt{u} \) blocks \( B_1, \ldots, B_{\sqrt{u}} \) of size \( \sqrt{u} \).
- \( \text{block}(x) = \) the index \( i \) such that \( x \in B_i \).

Example

For \( U = \{1, 2, \ldots, 16\} \), \( S = \{1, 2, 5, 6, 8, 11\} \),

- \( B_1 = \{1, 2, 3, 4\} \)
- \( B_2 = \{5, 6, 7, 8\} \)
- \( B_3 = \{9, 10, 11, 12\} \)
- \( B_4 = \{13, 14, 15, 16\} \)

\( \text{block}(10) = 3 \).
van Emde Boas tree

- For all $i$, store a structure for $S_i = S \cap B_i$ if $S_i \neq \emptyset$.
- Store a structure for the set $S' = \{ i : S_i \neq \emptyset \}$.
- Store array min: $\text{min}[i] = \text{min}(S_i)$.
- Store array max: $\text{max}[i] = \text{max}(S_i)$.

Example

For $U = \{1, 2, \ldots, 16\}, S = \{1, 2, 5, 6, 8, 11\},$

- $S_1 = S \cap \{1, 2, 3, 4\} = \{1, 2\}$
- $S_2 = S \cap \{5, 6, 7, 8\} = \{5, 6, 8\}$
- $S_3 = S \cap \{9, 10, 11, 12\} = \{11\}$
- $S_4 = S \cap \{13, 14, 15, 16\} = \emptyset$
- $S' = \{1, 2, 3\}$

$\text{min} = 1, 5, 11, \infty, \text{max} = 2, 8, 11, -\infty.$
Example

\[ U = \{1, 2, \ldots, 16\}, \quad S = \{1, 2, 5, 6, 8, 11\} \]
Successor

- $S_i = S \cap B_i$.
- $S' = \{i : S_i \neq \emptyset\}$.
- $\text{block}(x) =$ the index $i$ such that $x \in B_i$.

$\text{Successor}(S, x)$:

\[
\begin{align*}
\text{if } x & \leq \max[\text{block}(x)] \text{ then} \\
& \quad \text{return } \text{Successor}(S_{\text{block}(x)}, x) \\
\text{else} \\
& \quad i \leftarrow \text{Successor}(S', \text{block}(x) + 1) \\
& \quad \text{return } \min[i]
\end{align*}
\]

Case 1:

$x$ and its successor are in the same block.

\[S = \{7, 10, 26, 29, 33\}\]
\[S' = \{2, 5, 6\}\]
Successor

- $S_i = S \cap B_i$.
- $S' = \{i : S_i \neq \emptyset\}$.
- $\text{block}(x) = \text{the index } i \text{ such that } x \in B_i$.

Successor($S, x$):

\[
\text{if } x \leq \max[\text{block}(x)] \text{ then } \\
\quad \text{return } \text{Successor}(S_{\text{block}(x)}, x) \\
\text{else } \\
\quad i \leftarrow \text{Successor}(S', \text{block}(x) + 1) \\
\quad \text{return } \min[i]
\]

Case 2:

$x$ and its successor are in different blocks.

$S = \{7, 10, 26, 29, 33\}$

$S' = \{2, 5, 6\}$
Successor($S, x$):

if $x \leq \max[block(x)]$ then
    return Successor($S_{\text{block}(x)}, x$)
else
    $i \leftarrow \text{Successor}(S', \text{block}(x) + 1)$
    return $\min[i]$

$T_{\text{Suc}}(u) = T_{\text{Suc}}(\sqrt{u}) + \Theta(1) \implies T_{\text{Suc}}(u) = \Theta(\log \log u)$. 
Insert

Insert($S, x$):
\[
j \leftarrow \text{block}(x)
\]
\[
\text{if } \min[j] = \infty \text{ then}
\quad \text{Insert($S', j$)}
\]
\[
\text{Insert($S_j, x$)}
\]
\[
\min[j] \leftarrow \min(\min[j], x)
\]
\[
\max[j] \leftarrow \max(\max[j], x)
\]
\[
T_{\text{Ins}}(u) = 2T_{\text{Ins}}(\sqrt{u}) + \Theta(1) \implies T_{\text{Ins}}(u) = \Theta(\log u).
\]
Build structures on $\hat{S}_i = S_i \setminus \{\min(S_i)\}$ instead of $S_i$.

$\mathbb{S} = \{1,2,5,6,8,11\} \subseteq [1,16]$

$\text{min: } 1,5,11,\infty \quad \text{max: } 2,8,11,\infty$

$\hat{S}_1$

$2 \subseteq [1,4]$

$\text{min: } 2,\infty \quad \text{max: } 2,-\infty$

$\hat{S}_2$

$6,8 \subseteq [5,8]$

$\text{min: } 6,8 \quad \text{max: } 6,8$

S'

$1,2,3 \subseteq [1,4]$

$\text{min: } 1,3 \quad \text{max: } 2,3$

$\hat{S}_1$

$\hat{S}_2$

1

2

1,2

1,2
Improved Insert

Build structures on $\hat{S}_i = S_i \setminus \{\min(S_i)\}$ instead of $S_i$.

Insert$(S, x)$:

$\begin{align*}
    j & \leftarrow \text{block}(x) \\
    \text{if } \min[j] = \infty & \text{ then} \\
        \text{Insert}(S', j) \\
        \min[j] & \leftarrow x \\
        \max[j] & \leftarrow x
\end{align*}$

else

$\begin{align*}
    \text{if } x < \min[j] & \text{ then} \\
        \text{Insert}(\hat{S}_j, \min[j]) \\
        \min[j] & \leftarrow x \\
    \text{else} \\
        \text{Insert}(\hat{S}_j, x) \\
        \max[j] & \leftarrow \max(\max[j], x)
\end{align*}$

$T_{\text{Ins}}(u) = T_{\text{Ins}}(\sqrt{u}) + \Theta(1) \implies T_{\text{Ins}}(u) = \Theta(\log \log u)$. 

Ordered Dictionary
Successor\((S, x)\):
\[
\begin{align*}
    j &\leftarrow \text{block}(x) \\
    \text{if } x \leq \min[j] &\text{ then return } \min[j] \\
    \text{if } x \leq \max[\text{block}(x)] &\text{ then return } \text{Successor}(\hat{S}_j, x) \\
    i &\leftarrow \text{Successor}(S', j + 1) \\
    \text{return } \min[i]
\end{align*}
\]

\[
T_{\text{Suc}}(u) = \Theta(\log \log u).
\]
Let \( S(u) \) be the space complexity of a van Emde Boas tree.

The top structure uses \( \Theta(\sqrt{u}) \) words (min/max arrays and pointers to the sub-structures).

Recurrence:

\[
S(u) = (\sqrt{u} + 1)S(\sqrt{u}) + \sqrt{u}
\]

The solution to the recurrence is \( S(u) = \Theta(u) \).
Reducing space

- The min/max arrays and pointer arrays can be replaced by a hash table (assume we use a hash table with $\Theta(1)$ worst case search time).
- For every $i$ such that $S_i \neq \emptyset$ the table stores a tuple $(i, \min[i], \max[i], p_i)$ where $p_i$ is a pointer to the structure on $\hat{S}_i$.

| min:         | $\infty$, 11, $\infty$, $\infty$, 42, $\infty$, $\infty$, $\infty$, $\infty$, $\infty$, $\infty$, $\infty$ |
| max:        | $-\infty$, 17, $-\infty$, $-\infty$, 45, $-\infty$, $-\infty$, $-\infty$, $-\infty$, $-\infty$, $-\infty$, $-\infty$ |

2,11,17, 5,42,45,
Reducing space

- The min/max arrays and pointer arrays can be replaced by a hash table (assume we use a hash table with $\Theta(1)$ worst case search time).
- For every $i$ such that $S_i \neq \emptyset$ the table stores a tuple $(i, \text{min}[i], \text{max}[i], p_i)$ where $p_i$ is a pointer to the structure on $\hat{S}_i$.
- The space is $\Theta(n)$.
- Insert takes $\Theta(\log \log u)$ expected time.
- Successor/Predecessor take $\Theta(\log \log u)$ time (worst case).
Y-fast Trie
Build a trie from the binary representations (of length $\lceil \log_2 u \rceil$) of the elements of $S$. 

$U = \{0, 1, \ldots, 15\}$
$S = \{0, 2, 8, 9, 13, 14\}$
Create a doubly linked list on the leaves.
Each internal node store pointers to its minimum & maximum descendants leaves (figure shows only some of these pointers)
To find the successor of $x$, find the node $y$ in which the path of $x$ exits the trie.
If $y$ exits the trie to the right, go to the max. descendant leaf of $y$, and then move one position in the list of leaves.
If $y$ exits the trie to the right, go to the max. descendant leaf of $y$, and then move one position in the list of leaves.
If $y$ exits the trie to the right, go to the max. descendant leaf of $y$, and then move one position in the list of leaves.
x-fast trie

\[ U = \{0,1,\ldots,15\} \]
\[ S = \{0,2,8,9,13,14\} \]

Time complexity: \( \Theta(\log u) \).
Store the nodes in a hash table with $\Theta(1)$ worst case search time.

The key of a node $v$ is the concatenation of the characters on the path from the root to $v$. 
To find \( y \), we need to find the longest prefix of \( \text{bin}(x) \) which is a key of a node in the trie.

For \( x = 1011 \), the prefix is 10.
Successor in $\Theta(\log \log u)$ time

- To find the successor of $x$, find the longest prefix of $\text{bin}(x)$ in the hash table using binary search.
- Time complexity: $\Theta(\log \log u)$ (worst case).
To find the successor of $x$, find the longest prefix of $\text{bin}(x)$ in the hash table using binary search.

**Time complexity:** $\Theta(\log \log u)$ (worst case).
Successor in $\Theta(\log \log u)$ time

- To find the successor of $x$, find the longest prefix of $\text{bin}(x)$ in the hash table using binary search.
- Time complexity: $\Theta(\log \log u)$ (worst case).

Y-fast Trie

x=011001
Find(0110)
Find(01)
Successor in $\Theta(\log \log u)$ time

- To handle keys in constant time, a key is treated as a bit sequence.
- To differentiate between, for example the keys 1 and 01, the actual key consists of the length of the key in binary, followed by the actual key (padded to length $\lceil \log u \rceil$).

Example

- $1 \rightarrow 0011000$
- $01 \rightarrow 0100100$
- $1011 \rightarrow 1001011$
Analysis

- Successor/Predecessor: $\Theta(\log \log u)$ time (worst case).
- Insert/Delete: $\Theta(\log u)$ expected time.
- Space: $O(n \log u)$.

U = \{0, 1, ..., 15\}
S = \{0, 2, 8, 9, 13, 14\}

Y-fast Trie
y-fast trie (x-fast trie + indirection)

- Partition the elements of $S$ into consecutive groups $S_1, S_2, \ldots$ of sizes between $\frac{1}{2} \log u$ and $2 \log u - 1$.

![Diagram of elements partitioned into groups $S_1$, $S_2$, $S_3$]
Partition the elements of $S$ into consecutive groups $S_1, S_2, \ldots$ of sizes between $\frac{1}{2} \log u$ and $2 \log u - 1$.

Store each $S_i$ in a balanced search tree.

Space complexity:
The search tree of $S_i$ takes $\Theta(|S_i|)$ space.

The x-fast trie takes $\Theta(n \log u \cdot \log u) = \Theta(n)$ space.

Total space complexity: $\Theta(n + \sum_i |S_i|) = \Theta(n)$. 

**Y-fast Trie**
y-fast trie (x-fast trie + indirection)

- Partition the elements of $S$ into consecutive groups $S_1, S_2, \ldots$ of sizes between $\frac{1}{2} \log u$ and $2 \log u - 1$.
- Store each $S_i$ in a balanced search tree.
- Select separators $r_1, r_2, \ldots$ s.t. $\max(S_i) \leq r_i < \min(S_{i+1})$.
- Store $S' = \{r_1, r_2, \ldots\}$ in an x-fast trie.

Space complexity:
The search tree of $S_i$ takes $\Theta(|S_i|)$ space.
The x-fast trie takes $\Theta(n \log u \cdot \log u) = \Theta(n)$ space.
Total space complexity: $\Theta(n + \sum_i |S_i|) = \Theta(n)$. 

![Y-fast Trie](image)
Partition the elements of $S$ into consecutive groups $S_1, S_2, \ldots$ of sizes between $\frac{1}{2} \log u$ and $2 \log u - 1$.

Store each $S_i$ in a balanced search tree.

Select separators $r_1, r_2, \ldots$ s.t. $\max(S_i) \leq r_i < \min(S_{i+1})$.

Store $S' = \{r_1, r_2, \ldots\}$ in an x-fast trie.

Space complexity:

- The search tree of $S_i$ takes $\Theta(|S_i|)$ space
- The x-fast trie takes $\Theta\left(\frac{n}{\log u} \cdot \log u\right) = \Theta(n)$ space.
- Total space complexity: $\Theta(n + \sum_i |S_i|) = \Theta(n)$. 
Successor($S, x$)

- Find $r_i = \text{the successor of } x \text{ in the } x\text{-fast trie.}$
- If $\text{Successor}(S_i, x)$ exists, return it.
- Return $\text{Min}(S_{i+1})$.

Time complexity: $\Theta(\log \log u)$ (worst case).
Successor($S, x$)

- Find $r_i = \text{the successor of } x \text{ in the } x\text{-fast trie.}$
- If Successor($S_i, x$) exists, return it.
- Return Min($S_{i+1}$).

Time complexity: $\Theta(\log \log u)$ (worst case).
Insert($S, x$)

- Find the group $S_i$ in which $x$ fits.
- Insert $x$ to the search tree of $S_i$.
- If $|S_i| < 2 \log u$, stop.
- Split $S_i$ into two sets $S'_i$ and $S''_i$ of size $\log u$ each, and built search trees for $S'_i$ and $S''_i$.
- Select a separator $r'_i$ between $S'_i$ and $S''_i$, and insert it to the x-fast trie.
Insert($S, x$)

- Find the group $S_i$ in which $x$ fits.
- Insert $x$ to the search tree of $S_i$.
- If $|S_i| < 2 \log u$, stop.
- Split $S_i$ into two sets $S_i'$ and $S_i''$ of size $\log u$ each, and build search trees for $S_i'$ and $S_i''$.
- Select a separator $r_i'$ between $S_i'$ and $S_i''$, and insert it to the x-fast trie.
Insert($S, x$)

- Find the group $S_i$ in which $x$ fits.
- Insert $x$ to the search tree of $S_i$.
- If $|S_i| < 2 \log u$, stop.
- Split $S_i$ into two sets $S'_i$ and $S''_i$ of size $\log u$ each, and built search trees for $S'_i$ and $S''_i$.
- Select a separator $r'_i$ between $S'_i$ and $S''_i$, and insert it to the x-fast trie.

![Diagram of Y-fast Trie](image-url)
Insert($S, x$)

- Find the group $S_i$ in which $x$ fits.
- Insert $x$ to the search tree of $S_i$.
- If $|S_i| < 2 \log u$, stop.
- Split $S_i$ into two sets $S'_i$ and $S''_i$ of size $\log u$ each, and built search trees for $S'_i$ and $S''_i$.
- Select a separator $r'_i$ between $S'_i$ and $S''_i$, and insert it to the x-fast trie.
Insert($S, \ x$)

- Find the group $S_i$ in which $x$ fits.
- Insert $x$ to the search tree of $S_i$.
- If $|S_i| < 2 \log u$, stop.
- Split $S_i$ into two sets $S'_i$ and $S''_i$ of size $\log u$ each, and built search trees for $S'_i$ and $S''_i$.
- Select a separator $r'_i$ between $S'_i$ and $S''_i$, and insert it to the x-fast trie.

Cost of insert is

- **Fast insert** (w/o split): $\Theta(\log \log u)$.
  - insert to the tree of $S_i$ costs $\Theta(\log |S_i|)$.
- **Slow insert** (with split): $\Theta(\log u)$.
  - cost of insertion to the x-fast trie is $\Theta(\log u)$.
  - cost of splitting the tree of $S_i$ is $\Theta(\log u)$.
Cost of a single insert is either $\Theta(\log \log u)$ (fast) or $\Theta(\log u)$ (slow).

A split occurs due to $\log u$ fast insertions that increase the size of the set from $\log u$ to $2\log u$.

The cost of a slow insert can be charged to the cost of these $\log u$ fast insertions.

Time complexity: $\Theta(\log \log u)$ expected amortized.