

Advanced algorithms and data structures

Lecture: van Emde Boas Trees

Jacob Holm (jaho@di.ku.dk)

Dec 18th 2024

Today's Lecture

van Emde Boas Trees

- Predecessor search/ordered sets

- Naive

- Twolevel

- Recursive

- vEB: worst case $\mathcal{O}(\log \log |U|)$ time

- RS-vEB: expected $\mathcal{O}(\log \log |U|)$ time, $\mathcal{O}(n \log \log |U|)$ space

- R²S-vEB: expected $\mathcal{O}(\log \log |U|)$ time, $\mathcal{O}(n)$ space

- Bonus: vEB is optimal for $w = \Theta(\log n)$

- Bonus: Integer sorting in expected $\mathcal{O}(n \log \log |U|)$ time.

Predecessor search/ordered sets

Problem:

Given a universe $U = [u]$ where $u = 2^w$,
maintain subset $S \subseteq U$, $|S| = n$ under:

member(x, S): Return $[x \in S]$.

insert(x, S): Add x to S (assumes $x \notin S$).

delete(x, S): Remove x from S (assumes $x \in S$).

empty(S): Return $[S = \emptyset]$.

min(S): Return $\min S$ (assumes $S \neq \emptyset$).

max(S): Return $\max S$ (assumes $S \neq \emptyset$).

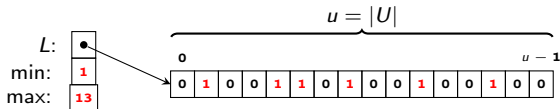
predecessor(x, S): Return $\max\{y \in S \mid y < x\}$
(assumes $\{y \in S \mid y < x\}$ is nonempty,
i.e. $S \neq \emptyset$ and $x > \min(S)$).

successor(x, S): Return $\min\{y \in S \mid y > x\}$
(assumes $\{y \in S \mid y > x\}$ is nonempty,
i.e. $S \neq \emptyset$ and $x < \max(S)$).

Naive

Idea: If we are willing to spend $\mathcal{O}(|U|)$ space...

Store S as a bit-array L of length $|U|$ such that $L[x] = [x \in S]$, and keep track of the min and max values explicitly.



$\{1, 4, 5, 7, 10, 13\}$

How fast is:

$\text{empty}(S)$, $\text{min}(S)$, and $\text{max}(S)$? worst case $\mathcal{O}(1)$ — $\text{empty}(S)$ is tricky.

$\text{member}(x, S)$? worst case $\mathcal{O}(1)$.

$\text{predecessor}(x, S)$ and $\text{successor}(x, S)$? worst case $\Theta(|U|)$.

$\text{delete}(x, S)$? worst case $\mathcal{O}(|U|)$ — need to update min / max.

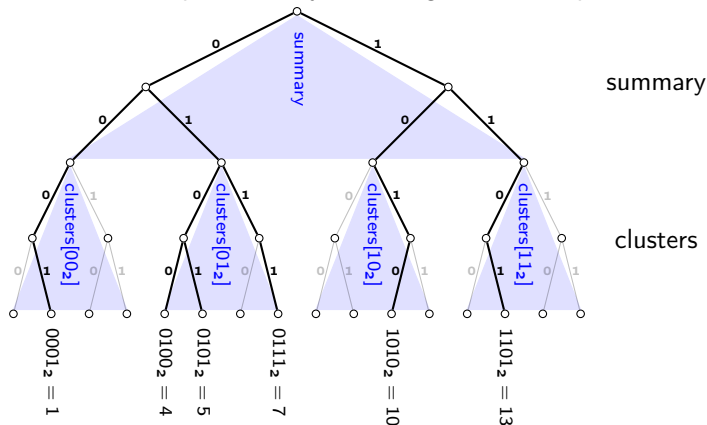
$\text{insert}(x, S)$? worst case $\mathcal{O}(1)$.

Bit-Trie

Idea: Think of each key as a w -bit string describing a path in a binary *trie* (= a special kind of tree).

The naive structure can be thought of as just storing which leaves exist. An alternative naive choice would be to store the actual trie, which we could maintain in $\mathcal{O}(w) = \mathcal{O}(\log|U|)$ time.

What if we instead split each key into a high and a low part?



Twollevel

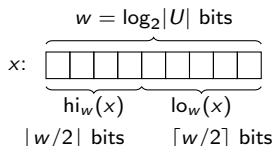
Idea: Split each key into *high* and *low* parts. Use naive for each.

Recall that $U = [2^w]$, and define:

$$\text{hi}_w(x) := \left\lfloor \frac{x}{2^{\lceil w/2 \rceil}} \right\rfloor$$

$$\text{lo}_w(x) := x \bmod 2^{\lceil w/2 \rceil}$$

$$\text{index}_w(h, \ell) := h \cdot 2^{\lceil w/2 \rceil} + \ell$$



note that $x = \text{index}_w(\text{hi}_w(x), \text{lo}_w(x))$.

Now let the structure directly store the values:

$$\text{min} := \min(S) \text{ if } S \neq \emptyset, \text{ else } 1$$

$$\text{max} := \max(S) \text{ if } S \neq \emptyset, \text{ else } 0$$

and use the naive structure to store

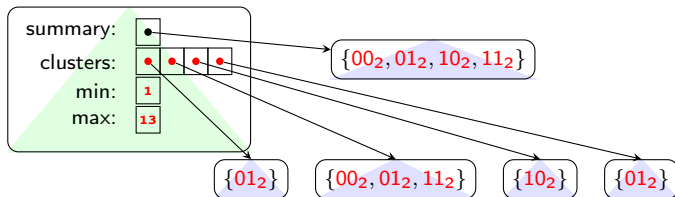
$$\text{summary} := \{\text{hi}_w(x) \mid x \in S\}$$

$$\text{clusters}[h] := \{\ell \in [2^{\lceil w/2 \rceil}] \mid \text{index}_w(h, \ell) \in S\} \quad \forall h \in [2^{\lfloor w/2 \rfloor}]$$

note that $S = \bigcup_{h \in \text{summary}} \{\text{index}_w(h, \ell) \mid \ell \in \text{clusters}[h]\}$.

Twollevel

We can draw the structure for the set $S = \{1, 4, 5, 7, 10, 13\} = \{0001_2, 0100_2, 0101_2, 0111_2, 1010_2, 1101_2\} \subseteq [2^4]$ as:



How fast (worst case) is:

$\text{empty}(S)$, $\text{min}(S)$, $\text{max}(S)$?

$\mathcal{O}(1)$

$\text{member}(x, S)$?

$\mathcal{O}(1) + 1 \times \text{naive} = \mathcal{O}(1)$

$\text{predecessor}(x, S)$, $\text{successor}(x, S)$?

$\mathcal{O}(1) + 1 \times \text{naive}$

$= \Theta(2^{\lceil w/2 \rceil}) = \Theta(\sqrt{|U|})$

$\text{delete}(x, S)$?

$\mathcal{O}(1) + 2 \times \text{naive} = \Theta(\sqrt{|U|})$

$\text{insert}(x, S)$?

$\mathcal{O}(1) + 2 \times \text{naive} = \mathcal{O}(1)$

```
function empty(S)
```

```
    return S.min > S.max
```

```
function min(S)
```

```
    ▷ Assumes  $S \neq \emptyset$ 
```

```
    return S.min
```

```
function max(S)
```

```
    ▷ Assumes  $S \neq \emptyset$ 
```

```
    return S.max
```

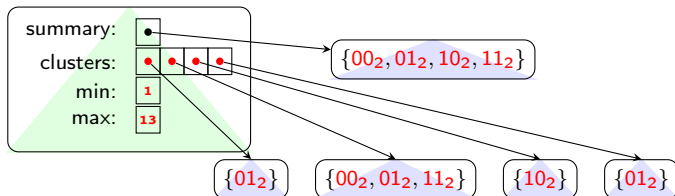
```
function memberw(x, S)
```

```
    return member $\lceil w/2 \rceil$ (loww(x), S.clusters[hiw(x)])
```

How can we improve the time further?

Twollevel

We can draw the structure for the set $S = \{1, 4, 5, 7, 10, 13\} = \{0001_2, 0100_2, 0101_2, 0111_2, 1010_2, 1101_2\} \subseteq [2^4]$ as:



How fast (worst case) is:

$\text{empty}(S)$, $\text{min}(S)$, $\text{max}(S)$?

$\mathcal{O}(1)$

$\text{member}(x, S)$?

$\mathcal{O}(1) + 1 \times \text{naive} = \mathcal{O}(1)$

$\text{predecessor}(x, S)$, $\text{successor}(x, S)$?

$\mathcal{O}(1) + 1 \times \text{naive}$

$= \Theta(2^{\lceil w/2 \rceil}) = \Theta(\sqrt{|U|})$

$\text{delete}(x, S)$?

$\mathcal{O}(1) + 2 \times \text{naive} = \Theta(\sqrt{|U|})$

$\text{insert}(x, S)$?

$\mathcal{O}(1) + 2 \times \text{naive} = \mathcal{O}(1)$

function $\text{predecessor}_w(x, S)$

▷ Assumes $S \neq \emptyset$ and $\text{min}(S) < x$

if $x > S.\text{max}$ **then**

return $S.\text{max}$

$p \leftarrow \text{hi}_w(x)$, $s \leftarrow \text{lo}_w(x)$, $C \leftarrow S.\text{clusters}[p]$

if not $\text{empty}(C)$ **and** $C.\text{min} < s$ **then**

return $\text{index}_w(p, \text{predecessor}_{\lceil w/2 \rceil}(s, C))$

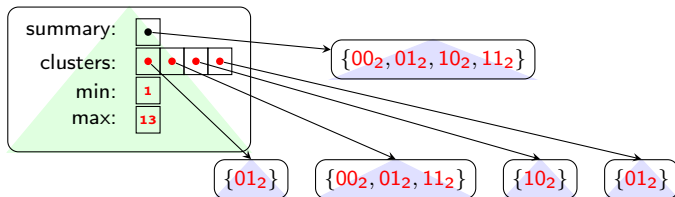
$p \leftarrow \text{predecessor}_{\lfloor w/2 \rfloor}(p, S.\text{summary})$

return $\text{index}_w(p, S.\text{clusters}[p].\text{max})$

How can we improve the time further?

Twollevel

We can draw the structure for the set $S = \{1, 4, 5, 7, 10, 13\} = \{0001_2, 0100_2, 0101_2, 0111_2, 1010_2, 1101_2\} \subseteq [2^4]$ as:



How fast (worst case) is:

$\text{empty}(S)$, $\text{min}(S)$, $\text{max}(S)$?

$\mathcal{O}(1)$

$\text{member}(x, S)$?

$\mathcal{O}(1) + 1 \times \text{naive} = \mathcal{O}(1)$

$\text{predecessor}(x, S)$, $\text{successor}(x, S)$?

$\mathcal{O}(1) + 1 \times \text{naive}$

$= \Theta(2^{\lceil w/2 \rceil}) = \Theta(\sqrt{|U|})$

$\text{delete}(x, S)$?

$\mathcal{O}(1) + 2 \times \text{naive} = \Theta(\sqrt{|U|})$

$\text{insert}(x, S)$?

$\mathcal{O}(1) + 2 \times \text{naive} = \mathcal{O}(1)$

function $\text{delete}_w(x, S)$

▷ Assumes $x \in S$

$p \leftarrow \text{hi}_w(x)$, $s \leftarrow \text{lo}_w(x)$, $C \leftarrow S.\text{clusters}[p]$

delete $_{\lceil w/2 \rceil}(s, C)$

if $\text{empty}(C)$ **then**

delete $_{\lfloor w/2 \rfloor}(p, S.\text{summary})$

if $S.\text{min} = S.\text{max}$ **then**

$S.\text{min} \leftarrow 1$, $S.\text{max} \leftarrow 0$

else if $x = S.\text{min}$ **then**

$p \leftarrow S.\text{summary}.min$, $S.\text{min} \leftarrow \text{index}_w(p, S.\text{clusters}[p].min)$

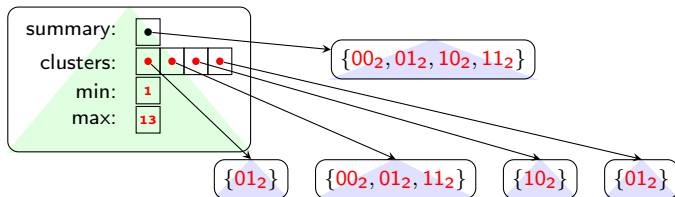
else if $x = S.\text{max}$ **then**

$p \leftarrow S.\text{summary}.max$, $S.\text{max} \leftarrow \text{index}_w(p, S.\text{clusters}[p].max)$

How can we improve the time further?

Twollevel

We can draw the structure for the set $S = \{1, 4, 5, 7, 10, 13\} = \{0001_2, 0100_2, 0101_2, 0111_2, 1010_2, 1101_2\} \subseteq [2^4]$ as:



How fast (worst case) is:

$\text{empty}(S)$, $\text{min}(S)$, $\text{max}(S)$?

$\mathcal{O}(1)$

$\text{member}(x, S)$?

$\mathcal{O}(1) + 1 \times \text{naive} = \mathcal{O}(1)$

$\text{predecessor}(x, S)$, $\text{successor}(x, S)$?

$\mathcal{O}(1) + 1 \times \text{naive}$

$= \Theta(2^{\lceil w/2 \rceil}) = \Theta(\sqrt{|U|})$

$\text{delete}(x, S)$?

$\mathcal{O}(1) + 2 \times \text{naive} = \Theta(\sqrt{|U|})$

$\text{insert}(x, S)$?

$\mathcal{O}(1) + 2 \times \text{naive} = \mathcal{O}(1)$

function $\text{insert}_w(x, S)$

▷ Assumes $x \notin S$

if $\text{empty}(S)$ **then**

$S.\text{min} \leftarrow x$, $S.\text{max} \leftarrow x$

if $x < S.\text{min}$ **then** $S.\text{min} \leftarrow x$

if $x > S.\text{max}$ **then** $S.\text{max} \leftarrow x$

$p \leftarrow \text{hi}_w(x)$, $s \leftarrow \text{lo}_w(x)$

if $\text{empty}(S.\text{clusters}[p])$ **then**

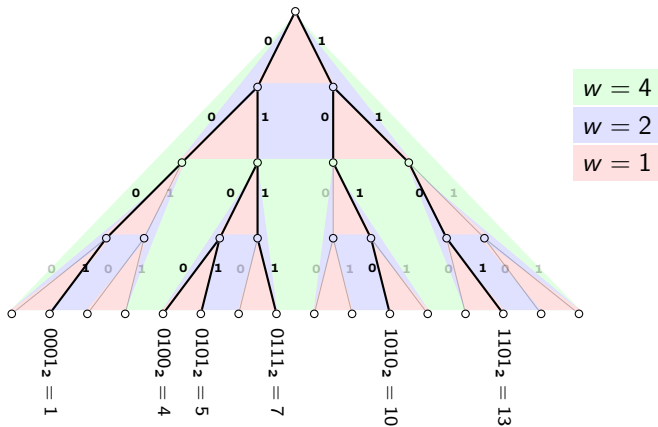
$\text{insert}_{\lfloor w/2 \rfloor}(p, S.\text{summary})$

$\text{insert}_{\lceil w/2 \rceil}(s, S.\text{clusters}[p])$

How can we improve the time further?

Recursive

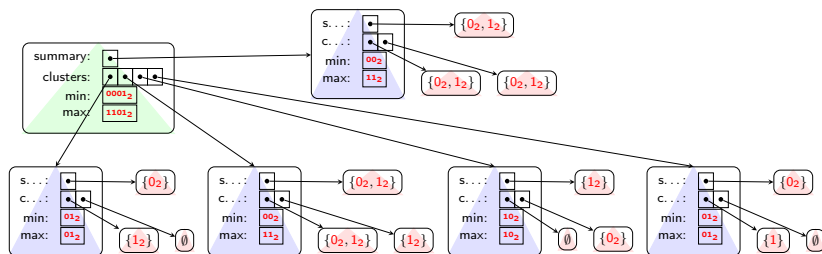
Idea: Instead of using the naive structure for summary and clusters[h], recursively use the same type of structure (stop recursion when $w = 1$).



Recursive

Idea: Instead of using the naive structure for summary and clusters[h], recursively use the same type of structure (stop recursion when $w = 1$).

We can draw the recursive structure for the set $S = \{1, 4, 5, 7, 10, 13\} = \{0001_2, 0100_2, 0101_2, 0111_2, 1010_2, 1101_2\} \subseteq [2^4]$ as:



This structure is (essentially) what the book calls “proto-vEB”. Note that the naive structure we started with is almost completely gone.

Recursive

Theorem

The recursion depth of this structure, when used on the universe $U = [2^w]$ is $\lceil \log_2 w \rceil = \mathcal{O}(\log \log |U|)$.

Proof.

Let $d(w)$ be the recursion depth when working with a universe of size 2^w . We will prove by strong induction that $d(w) = \lceil \log_2 w \rceil$.

The base case is $w = 1$ where there is no recursion, so $d(1) = 0 = \lceil \log_2(1) \rceil$.

For the induction case, suppose $w > 1$ and that $d(w') = \lceil \log_2(w') \rceil$ for all $w' \in [w]_+$. Now let $w' = \lceil w/2 \rceil \in [w]_+$, then the largest universe size used in the recursion is $2^{w'}$, and (by induction)

$d(w) = 1 + d(w') = 1 + \lceil \log_2(w') \rceil = \lceil \log_2(2w') \rceil$. What remains is to show $\lceil \log_2(2w') \rceil = \lceil \log_2(w) \rceil$.

If w is even, $2w' = w$ and we are done.

Otherwise $w > 1$ is odd so $w \geq 3$ and the smallest integer $k = \lceil \log_2(w) \rceil$ such that $2^k \geq w$ satisfies $k \geq 1$. Thus 2^k is even and so must satisfy $2^k \geq w + 1 = 2w'$ and therefore also $k = \lceil \log_2(2w') \rceil = \lceil \log_2(w) \rceil$. \square

Recursive

Using the recursive structure, how fast is:

$\text{empty}(S)$, $\text{min}(S)$, and $\text{max}(S)$? worst case $\mathcal{O}(1)$.

$\text{member}(x, S)$? $\mathcal{O}(1) + 1 \times \text{recursion} = \mathcal{O}(d(w)) = \mathcal{O}(\log \log |U|)$.

$\text{predecessor}(x, S)$ and $\text{successor}(x, S)$? $\mathcal{O}(1) + 1 \times \text{recursion}$
 $= \mathcal{O}(d(w)) = \mathcal{O}(\log \log |U|)$.

$\text{insert}(x, S)$ and $\text{delete}(x, S)$? $\mathcal{O}(1) + 2 \times \text{recursion}$
 $= \Theta(2^{d(w)}) = \Theta(w) = \Theta(\log |U|)$.

How can we improve the update time? Somehow make insert and delete recurse in only one substructure.

vEB: worst case $\mathcal{O}(\log \log |U|)$ time

Idea: Exclude $\min(S)$ and/or $\max(S)$ from the set of keys stored in summary and clusters. (CLRS excludes only $\min(S)$, I exclude both).

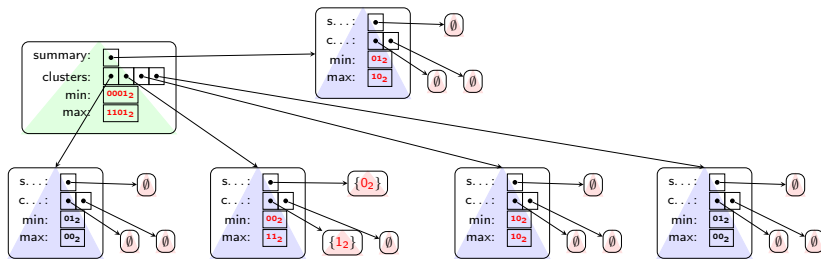
Specifically, redefine summary and clusters to recursively store the sets:

$$\text{summary} := \{\text{hi}_w(x) \mid x \in S \setminus \{\min, \max\}\}$$

$$\text{clusters}[h] := \{\ell \in [2^{\lceil w/2 \rceil}] \mid \text{index}_w(h, \ell) \in S \setminus \{\min, \max\}\} \quad \forall h \in [2^{\lfloor w/2 \rfloor}]$$

We can draw the van Emde Boas tree for the set $S =$

$$\{1, 4, 5, 7, 10, 13\} = \{0001_2, 0100_2, 0101_2, 0111_2, 1010_2, 1101_2\} \subseteq [2^4] \text{ as:}$$



vEB: worst case $\mathcal{O}(\log \log |U|)$ time, $\mathcal{O}(|U|)$ space

```
1: function predecessorw( $x, S$ )           ▷ Assumes  $S \neq \emptyset$  and  $x > S.\text{min}$ 
2:   if  $x > S.\text{max}$  then
3:     return  $S.\text{max}$ 
4:   if  $w = 1$  then
5:     return  $S.\text{min}$ 
6:    $p \leftarrow \text{hi}_w(x), s \leftarrow \text{lo}_w(x), C \leftarrow S.\text{clusters}[p]$ 
7:   if not empty( $C$ ) and  $C.\text{min} < s$  then
8:     return indexw( $p, \text{predecessor}_{\lceil w/2 \rceil}(s, C)$ )
9:   if empty( $S.\text{summary}$ ) or  $p \leq S.\text{summary}.\text{min}$  then
10:    return  $S.\text{min}$ 
11:    $p \leftarrow \text{predecessor}_{\lfloor w/2 \rfloor}(p, S.\text{summary})$ 
12:   return indexw( $p, S.\text{clusters}[p].\text{max}$ )
```

Theorem

predecessor_w(x, S) takes worst case $\mathcal{O}(d(w)) = \mathcal{O}(\log \log |U|)$ time.

Proof.

It makes at most one recursive call.



vEB: worst case $\mathcal{O}(\log \log |U|)$ time, $\mathcal{O}(|U|)$ space

```
1: function insertw(x, S)                                ▷ Assumes  $x \notin S$ 
2:   if empty(S) then
3:     S.min  $\leftarrow$  x, S.max  $\leftarrow$  x, return
4:   if S.min = S.max then
5:     if  $x < S.min$  then S.min  $\leftarrow$  x
6:     if  $x > S.max$  then S.max  $\leftarrow$  x
7:     return
8:   if  $x < S.min$  then S.min  $\leftrightarrow$  x
9:   if  $x > S.max$  then S.max  $\leftrightarrow$  x
10:  p  $\leftarrow$  hiw(x), s  $\leftarrow$  low(x)
11:  if empty(S.clusters[p]) then
12:    insert $\lfloor w/2 \rfloor$ (p, S.summary)
13:  insert $\lceil w/2 \rceil$ (s, S.clusters[p])
```

Theorem

insert_w(x, S) takes worst case $\mathcal{O}(d(w)) = \mathcal{O}(\log \log |U|)$ time.

Proof.

It makes at most one recursive call on a non-empty substructure, and inserting in an empty substructure takes constant time. □

RS-vEB: expected $\mathcal{O}(\log \log |U|)$ time, $\mathcal{O}(n \log \log |U|)$ space

Idea: Use a hash table instead of an array to store clusters $[\cdot]$, and don't store empty substructures.

Why does this change updates from worst case $\mathcal{O}(\log \log |U|)$ to expected $\mathcal{O}(\log \log |U|)$ time? Because updates to a hash table take expected $\mathcal{O}(1)$ time rather than worst case.

Why does this only use $\mathcal{O}(n \cdot d(w)) = \mathcal{O}(n \log \log |U|)$ space? Because the empty structure uses $\mathcal{O}(1)$ space and insert_w only creates or updates $\mathcal{O}(d(w))$ substructures in the worst case. Each of these costs at most an additional $\mathcal{O}(1)$ space.

R^2S -vEB: expected $\mathcal{O}(\log \log |U|)$ time, $\mathcal{O}(n)$ space

Idea: Partition S into “chunks” of size $\Theta(\min\{n, \log \log |U|\})$, and store only one element representing each chunk in the RS-vEB structure. I.e. RS-vEB stores only $\mathcal{O}(\max\{1, n/\log \log |U|\})$ elements, using $\mathcal{O}(n)$ space.

We can store each chunk as a sorted linked list, and keep a hash table mapping each representative to its chunk. This also takes $\mathcal{O}(n)$ space.

predecessor and successor uses the RS-vEB structure to find the nearest two representatives in $\mathcal{O}(\log \log |U|)$ time, and can then spend linear time in the size of the two chunks to find the result.

insert may have to split a chunk that becomes too large and insert a new representative in the RS-vEB structure. Splitting the chunk can take linear time in the size of the chunk, and together with inserting the new representative into the RS-vEB structure, this still only takes expected $\mathcal{O}(\log \log |U|)$ time.

Similarly, delete may have to join two chunks and delete a representative, but again this only takes expected $\mathcal{O}(\log \log |U|)$ time.

Bonus: vEB is optimal for $w = \Theta(\log n)$

In fact, $\mathcal{O}(\log w)$ is the optimal query time when $w \in \Theta(\log n)$, or equivalently when $n \in 2^{\Theta(w)}$. This was shown in:

Time-space trade-offs for predecessor search,
Mihai Pătraşcu and Mikkel Thorup,
STOC'06: Proceedings of the thirty-eighth annual ACM symposium on
Theory of Computing, pages 232–240.
<https://doi.org/10.1145/1132516.1132551>

Bonus: Integer sorting in expected $\mathcal{O}(n \log \log |U|)$ time.

Idea: Insert all elements in R^2S -vEB structure, use $\min(S)$ then repeatedly $\text{successor}(x, S)$.

This takes expected $\mathcal{O}(n \log \log |U|) = \mathcal{O}(n \log w)$ time.

Compare this to other famous sorting methods:

QuickSort Takes expected $\mathcal{O}(n \log n)$ time.

CountingSort Takes $\mathcal{O}(n + |U|)$ time.

RadixSort Takes $\mathcal{O}((n + r) \log_r |U|) = \mathcal{O}((n + r) \frac{w}{\log r})$ time for radix r , e.g. $\mathcal{O}(\frac{n}{\log n} \log |U|) = \mathcal{O}(\frac{nw}{\log n})$ for $r = n$.

We observe that for $\log n \in \Omega(\log w) \cap \mathcal{O}(\frac{w}{\log w})$, or equivalently $n \in w^{\Omega(1)} \cap 2^{\mathcal{O}(\frac{w}{\log w})}$, sorting using vEB should be better than these alternatives.

Summary

Today's topic was van Emde Boas Trees, which is a data structure for sets of bounded integers. We have covered

- ▶ A naive implementation of ordered sets.
- ▶ A slightly less naive two-level implementation.
- ▶ A recursive implementation with good query time (proto-vEB).
- ▶ The actual van Emde Boas Tree (vEB).
- ▶ An extension to vEB using hashing to save space (RS-vEB).
- ▶ A further space-saving extension using “indirection” (R^2S -vEB).
- ▶ Some notes about the optimality of vEB and on integer sorting.
- ▶ Next time: Polygonal Triangulation