Binomial Heaps

to support fast union of heaps in time $O(\log(n))$ where

- $n_1 =$ # elements in Heap 1
- $n_2 =$ # elements in Heap 2

Set of Binomial Trees

$B_i$ : contains $2^i$ nodes

Any set (any number) can be expressed as a union of $\log(n)$ Binomial Trees (binary representation)

$B_0$, $B_1$, $B_2$, ...

$H_1$

$H_2$

$H$

$B_0$, $B_1$, $B_2$, ...

$B_{in}$
The total number of operations (cutting and joining of binomial trees) is \( O\left( \max\left(\log(n_1), \log(n_2)\right) \right) \) which is \( O\left( \log(n_1+n_2) \right) \).

Total cost is logarithmic for creating a union of two binomial heaps / melding of heaps.

---

**Make heap (S)**: First figure out which order of heaps will be required and partition the elements accordingly (in any arbitrary order): \( O(n) \)

For any specific \( B_i \), create \( B_{i-1}(\Delta) \) - then make the larger root the leftmost child of the smaller root.

Total \( O(|S_1| \cdot \log n) \)
**Insertion:** Create a Binomial Heap with the node to be inserted. Union the two heaps. Time: $O(\log(n))$.

**Deletion:** A simple case for deletion happens when we delete the root of some Binomial Tree.

Create another heap with the children of the deleted node, say $H'$. Union $(H, H')$. Time: $O(\log(n))$.

**Case:** Arbitrary node:

Decrease the value $a$ to say $-\infty$. 
Decrease key takes $O(\log n)$.
(by swapping with parent as long as necessary).

Total time: length of path + delete root
$\Rightarrow O(\log n)$

Dictionary Data structure

Supports search, updates (insert, delete)

**Semi-dynamic**

Suppose we have at some juncture $n$ elements, say $n = 11$
$n = 2^3 + 2^1 + 2^0$

We maintain 3 sorted arrays of elements, say $A_3, A_1, A_0$

$|A_3| : 2^3$ \hspace{0.5cm} $|A_1| : 2^1$ \hspace{0.5cm} $|A_0| : 2^0$

Within each array we store the elements in sorted order
Search \( x \): Search for \( x \) in \( A_3, A_1, A_0 \).

\[ A_3: [50 \ldots 5] \]
\[ A_1: [100 \ldots 2] \]

\( x = 10 \) In general for a set with \( n \) elements, we have to search \( \log n \) sorted arrays.

\[ \Rightarrow \text{Time: } O(\log^2 n) = (\log n)^2 \]

**Insertion**: \( n \to n+1 \)

\[ 11 \to 12 \]

\[ A_3, A_1, A_0 \]

\[ A_3, A_2 \]

In general, it would lead to creating/destruction of sorted arrays.

\[ 7 6 5 4 3 2 1 x_0 \]

\[ 1 0 \ 0 \ 0 \ 0 \]

\[ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \]

---

\[ \text{destroyed} \]

\[ \text{created} \]
\[0111111 \rightarrow 100000^{2^j} - 1^{2^j}\]

What is Time to create \( A_j \)?

If we sort \( \Rightarrow O(2^j \log(2^j)) = j \cdot 2^j \)

Claim: Can be done in \( O(2^j) \)

Quiz (in class) on 22nd May (Th)

20 mins