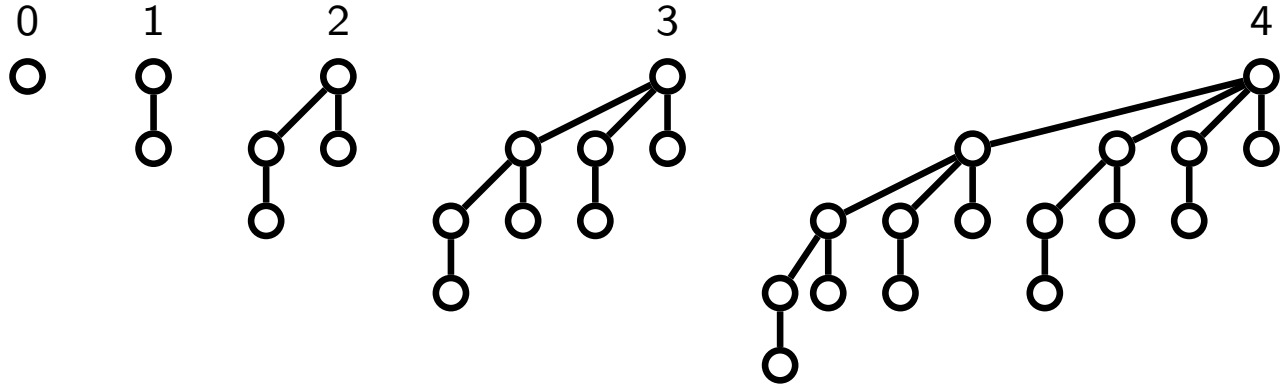
Binomial and Fibonacci Heaps

Binomial Trees

Definition
A *binomial tree* is defined recursively as follows:

- A binomial tree of order 0 is a single node.
- A binomial tree of order k has a root node with children that are roots of binomial trees of orders $k-1, k-2, \ldots, 2, 1, 0$



Note:

- A Binomial Tree of order k has exactly 2^k nodes
- A Binomial Tree of order k can be constructed from 2 Binomial Trees of order $k-1$ by attaching one of them as a new leftmost child of the root of the other: this is the basis of the efficient *merge* operation for Binomial Heaps.

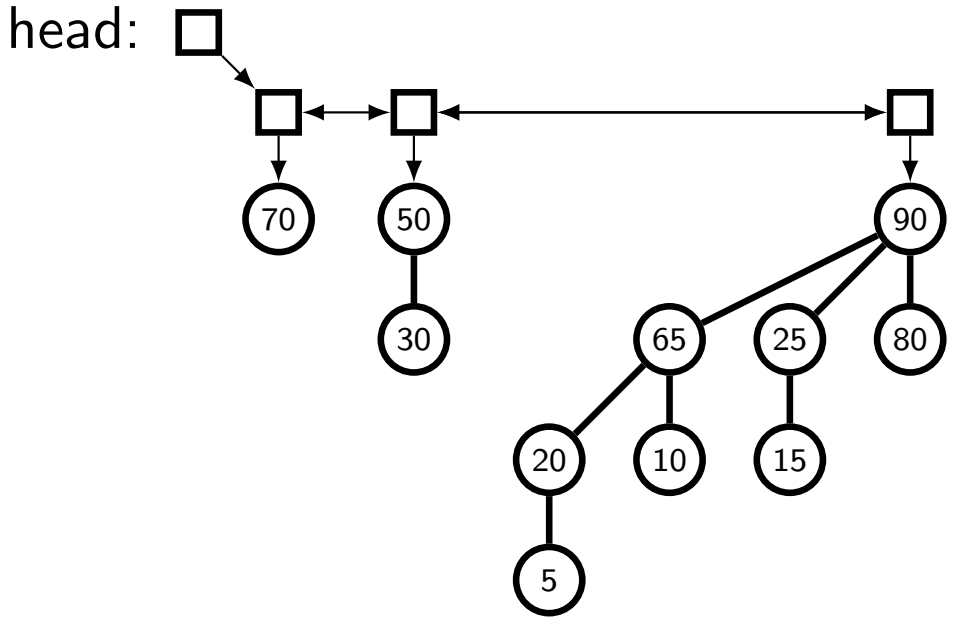
Binomial Heaps

A *Binomial Heap* is a list of *Binomial Trees* with the properties:

- There can be only zero or one *Binomial Trees* of each order
- Each *Binomial Tree* satisfies the priority ordering property: each node has priority less than or equal to its parent.

A typical implementation would use a doubly linked list of pointers to the root of each component Binomial Tree kept in order of Binomial Tree orders.

Binomial Heap Example, Find Max



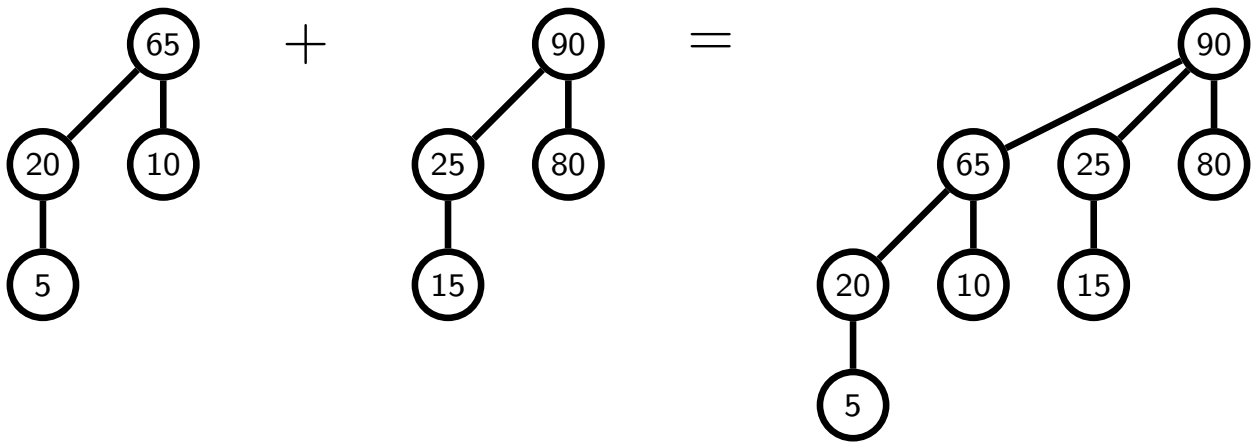
Note that there is no ordering between the keys in the **different** component Binomial Trees

Finding the node with highest priority is a linear search through the trees comparing the root values. Since, in the worst case, the number of nodes double in each consecutive tree, there will be $\log n$ trees required to store n values, so this is $O(\log n)$

Binomial Heap Merge

The key operation is merge, which merges two binomial heaps into one. Inserting a new value into a binomial tree works by merging a simple one node heap with the new value into the existing heap.

Note that if we have two binomial trees of the same order k, we can merge them into a single binomial tree of order $k+1$ by adding one of the trees as the leftmost child of the root of the other:

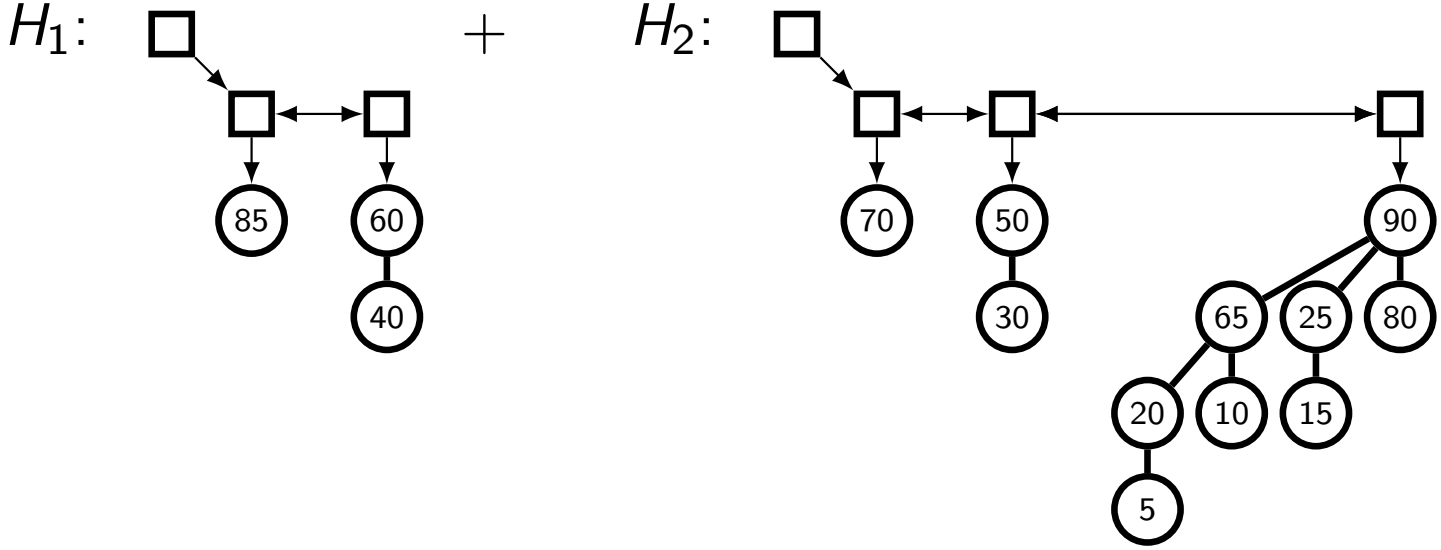


Binomial Heap Merge works in a way that is analogous to elementary addition of integers: Think of each Binomial Tree as a single binary digit in the addition.

Binomial Heap Merge

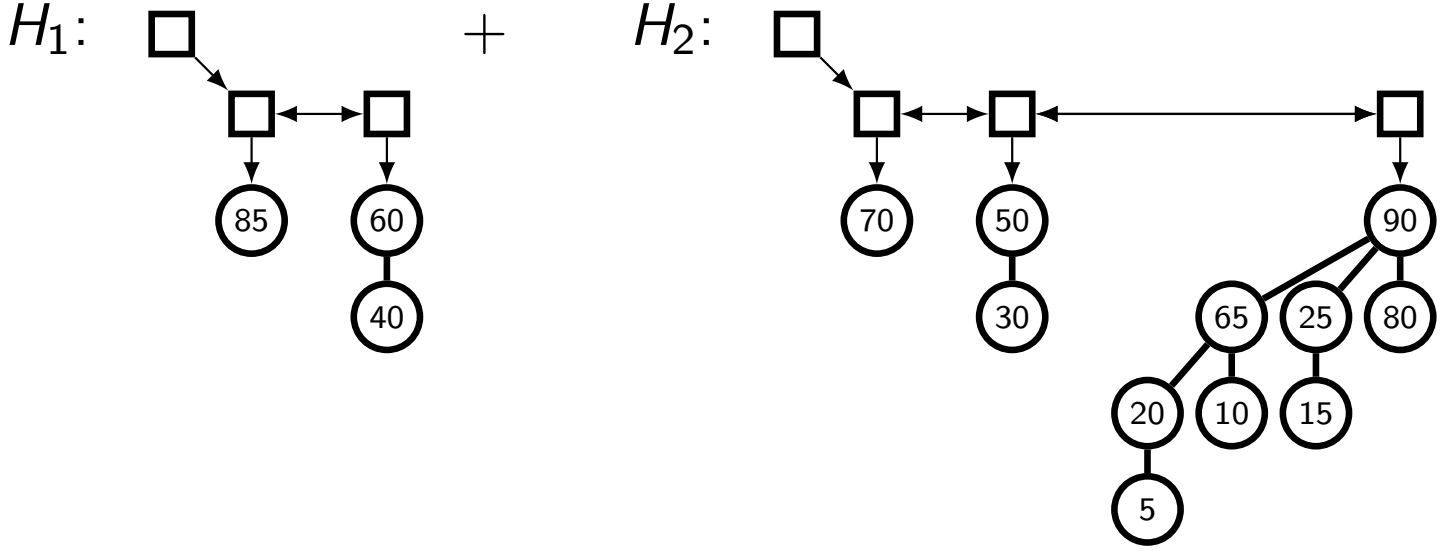
- "Adding" two trees of the same order k produces a single tree of order $k+1$, and is treated as a "*carry out*"
- Iterate through the tree orders $0, 1, 2, \ldots$. For each order, set the resulting tree and carry out to be the merge of any carry in and the trees of that order in the two heaps
- For any particular order k, there may be between 0 and 3 input trees to be merged, with the following effect on the result heap:
  - 0: no output tree of order k and there is no carry out.
  - 1: the output tree of order k is the input tree and there is no carry out.
  - 2: no output tree of order k and the carry out (of order $k+1$) is the merge of the two input trees
  - 3: the output tree of order k is one of the input trees, and the carry out (of order $k+1$) is the merge of the other two input trees

Binomial Heap Merge Example



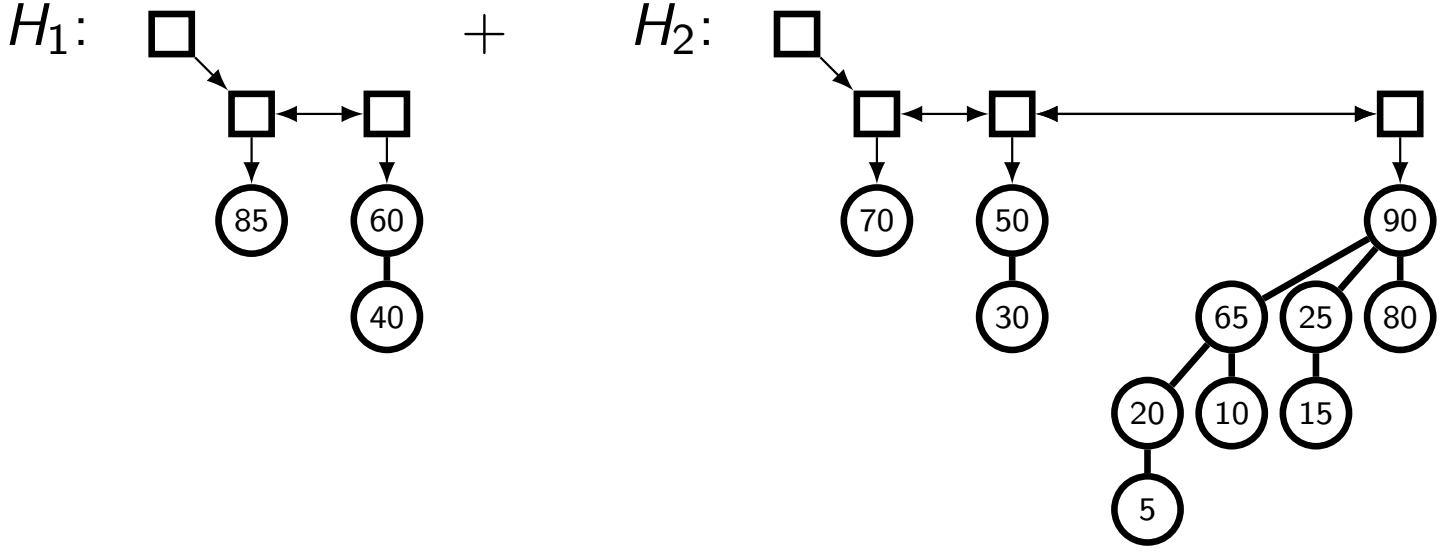
Order	Carry-in	H_1	H_2	Out	Carry-out
0		85	70		85 70

Binomial Heap Merge Example



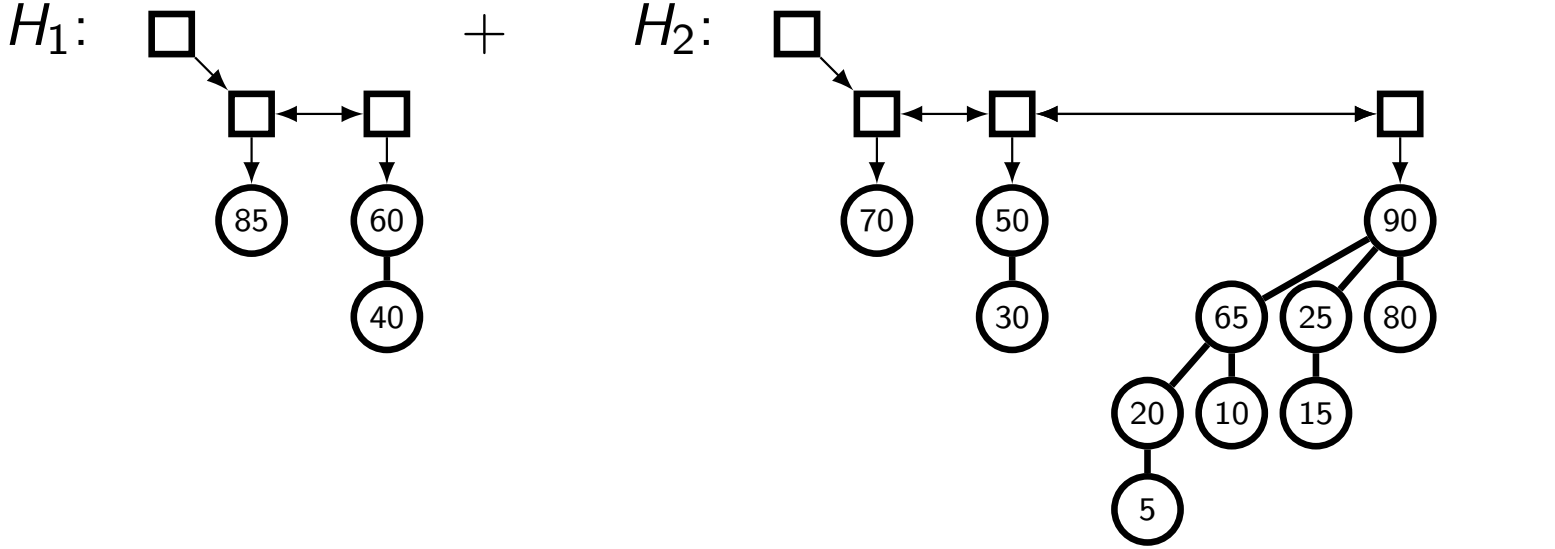
Order	Carry-in	H_1	H_2	Out	Carry-out
0		85	70		85 70
1	85 70	60 40	50 30	85 70	60 50 40 30

Binomial Heap Merge Example

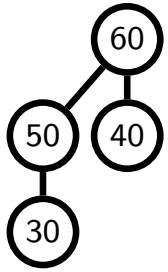


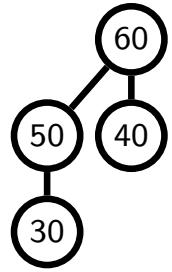
Order	Carry-in	H_1	H_2	Out	Carry-out
1	85, 70	60, 40	50, 30	85, 70	60, 50, 40, 30
2	60, 50, 40, 30			60, 50, 40, 30	

Binomial Heap Merge Example

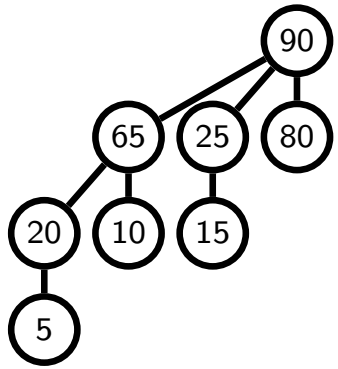


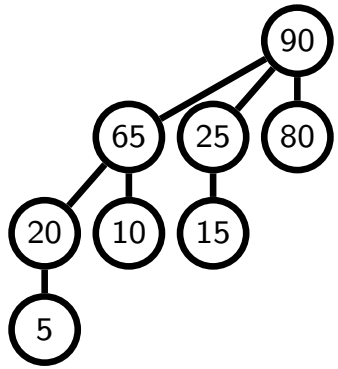
Order	Carry-in	H_1	H_2	Out	Carry-out
2					



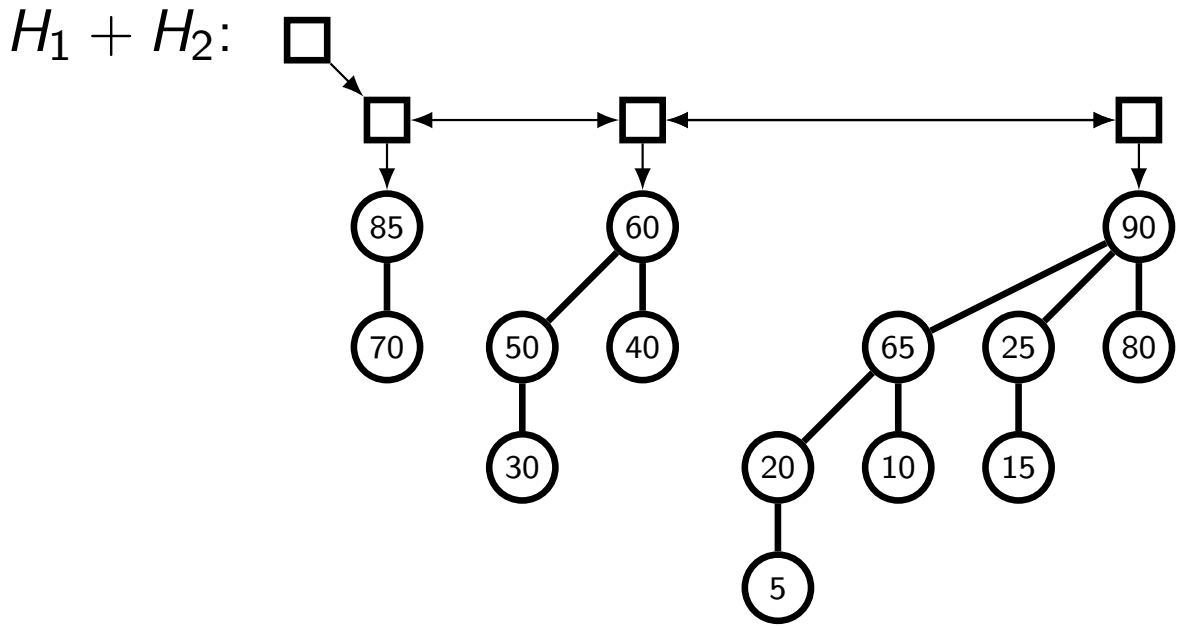


3





Binomial Heap Merge Example: Final Result



Complexity

Merging two Binomial Trees is $O(1)$

Merging two Binomial Heaps requires, in general, the merging of $O(\log n)$ Binomial Trees, hence $O(\log n)$

Inserting merges two Binomial Heaps: one with a single Binomial Tree of order 0. There is a probability of

- $\frac{1}{2}$ that the other heap has a Binomial tree of order 0 and so requires merging of order 0 trees and a carry-out
- $\left(\frac{1}{2}\right)^2$ that the other heap has a Binomial tree of order 1 and requires a merge
- $\left(\frac{1}{2}\right)^3$ etc.

Full average (amortised) cost of insertion is

$$O(1) \times \sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^i = O(1)$$

Binomial Heap: Other operations

There is no fast way to build a Binomial Heap from a collection of n key values, so simply insert n times: $O(n)$

To change the priority of a node in a Binomial Heap, we can use a bubble up/down process similar to that of Binary Heaps: $O(\log n)$

Deleting the highest priority node requires finding it: a linear search through the roots of the Binomial trees: $O(\log n)$, and removing the root node of that tree and merging the subtrees back into the binomial heap: $O(\log n)$. Total cost:

$$O(\log n) + O(\log n) = O(\log n)$$

Deleting non-root nodes can be done by setting the node's priority to ∞, bubbling it up to the root and then deleting it as in the previous case: $O(\log n)$

Fibonacci Heaps

Fibonacci Heaps are similar to Binomial Heaps in that they are also a collection of trees, but with different constraints on their structure. They are considerably more complex than Binomial Heaps, and make use of lazy modifications to keep themselves organised.

Their advantage over Binomial Heaps are that they achieve complexity of $O(1)$ for merge, and updating the priority of a node has amortised complexity of $O(1)$