A Detailed Hand Note on Binomial Heaps

Fahim Faiaz*

Abstract

A Binomial Heap is a sophisticated data structure designed for implementing a Priority Queue. It is a collection (or forest) of Binomial Trees that supports fast merging of two heaps. This note covers the fundamental concepts of Binomial Trees, the structure and properties of Binomial Heaps, and a detailed analysis of their operations with pictorial representations.

Contents

1 Binomial Trees (B_k) 2
- 1.1 Definition . . . 2
- 1.2 Properties of a B_k Tree . . . 2
- 1.3 Pictorial Representation of B_k Construction . . . 2

2 Binomial Heap Structure 2
- 2.1 Definition . . . 2
- 2.2 Heap Property and Binary Representation . . . 2
- 2.3 Pictorial Representation of a Binomial Heap . . . 3

3 Operations on Binomial Heaps 3
- 3.1 Find-Minimum . . . 3
- 3.2 Union (The Core Operation) . . . 3
- 3.3 Insert . . . 4
- 3.4 Extract-Minimum . . . 4
- 3.5 Decrease-Key . . . 4
- 3.6 Delete . . . 4

4 Complexity Analysis 4
- 4.1 Worst-Case Time Complexity Summary . . . 4
- 4.2 Amortized Analysis Highlight . . . 4

5 Conclusion 5

*July 29, 2025

1 Binomial Trees (B_k)

A Binomial Heap is built from a specific type of tree called a Binomial Tree. Therefore, understanding Binomial Trees is the first step.

1.1 Definition

A Binomial Tree of order k, denoted B_k, is defined recursively:

- A Binomial Tree B_0 is a single node.
- A Binomial Tree B_k is formed by linking two Binomial Trees B_{k-1} together. The root of one B_{k-1} becomes a child of the root of the other B_{k-1}.

1.2 Properties of a B_k Tree

A Binomial Tree B_k has several important properties:

1. **Number of Nodes:** It has exactly 2^k nodes.
2. **Height:** The height of the tree is k.
3. **Root Degree:** The root node has a degree of k (i.e., k children).
4. **Children Structure:** The children of the root of B_k, from left to right, are the roots of Binomial Trees $B_{k-1}, B_{k-2}, \ldots, B_1, B_0$.

1.3 Pictorial Representation of B_k Construction

The recursive construction of Binomial Trees is visualized below.

2 Binomial Heap Structure

2.1 Definition

A Binomial Heap H is a collection of Binomial Trees that satisfies the following properties:

1. **Unique Orders:** For any non-negative integer k, there is at most one Binomial Tree B_k in the heap.
2. **Min-Heap Property:** Each Binomial Tree in the heap is min-heap ordered. The key of a node is greater than or equal to the key of its parent. Consequently, the root of each Binomial Tree contains the smallest key in that tree.

The roots of the Binomial Trees are connected in a **root list**, which is typically a linked list sorted by the order of the trees (from smallest to largest).

2.2 Heap Property and Binary Representation

An interesting property of a Binomial Heap with n nodes is that its structure corresponds to the binary representation of n. If the i-th bit in the binary representation of n is 1, then the heap contains a Binomial Tree B_i. For example, a heap with 13 nodes:

$$13 = 8 + 4 + 1 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

So, binary representation is 1101_2. This heap will consist of trees B_3, B_2, and B_0.

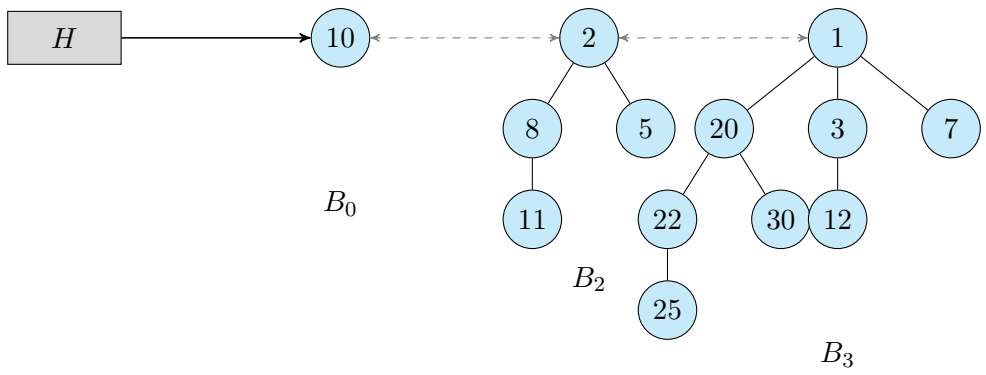

Figure 2: A Binomial Heap with 13 nodes, consisting of B_0, B_2, and B_3 trees.

2.3 Pictorial Representation of a Binomial Heap

Below is an example of a Binomial Heap with 13 nodes. The roots are linked together.

3 Operations on Binomial Heaps

The power of Binomial Heaps comes from the efficiency of their operations, especially the `union` operation.

3.1 Find-Minimum

To find the minimum key, we scan the root list. Since each tree is min-heap ordered, the minimum key in the entire heap must be one of the roots.

- **Procedure:** Iterate through the roots in the root list and find the one with the minimum key.
- **Complexity:** The number of trees in a heap with n nodes is at most $\lfloor \log_2 n \rfloor + 1$. Thus, the complexity is $O(\log n)$.

3.2 Union (The Core Operation)

The union operation merges two Binomial Heaps, H_1 and H_2, into a single heap. The procedure is analogous to binary addition.

1. **Merge Root Lists:** Merge the root lists of H_1 and H_2 into a single list, sorted by tree degree.
2. **Combine Trees of Same Order:** Iterate through the merged list. If you find two trees of the same order, B_k, link them to form a B_{k+1} tree. The root with the smaller key becomes the root of the new B_{k+1} tree.
3. **Handle Carries:** This linking might create a new B_{k+1} that needs to be merged with an existing B_{k+1} in the list. This is handled just like a "carry" in binary addition. Repeat until no two trees have the same order.

Complexity: $O(\log n_1 + \log n_2) = O(\log n)$, where $n = n_1 + n_2$.

3.3 Insert

Inserting a new element into a heap H is a special case of the union operation.

- **Procedure:** Create a new Binomial Heap H' containing only the new element (as a B_0 tree). Then, union H and H'.
- **Complexity:** Worst-case is $O(\log n)$, but the amortized complexity is $O(1)$.

3.4 Extract-Minimum

This operation removes the node with the minimum key from the heap.

1. **Find Min-Root:** Find the root with the minimum key, let this be the root of a B_k tree.
2. **Remove Tree:** Remove this B_k tree from the heap's root list.
3. **Create New Heap:** The children of the removed root form a new Binomial Heap H'. The children are roots of trees $B_{k-1}, B_{k-2}, \ldots, B_0$.
4. **Union:** Union the original heap H (now without the min-tree) and the new heap H'.

Complexity: This involves a find-min ($O(\log n)$) and a union ($O(\log n)$), so the total complexity is $O(\log n)$.

3.5 Decrease-Key

To decrease the key of a node x to a new value k.

- **Procedure:** Set the key of node x to k. This might violate the min-heap property. If the new key is smaller than its parent's key, repeatedly swap the node with its parent (bubble it up) until the min-heap property is restored or the node becomes a root.
- **Complexity:** The maximum height of any tree is $\log n$, so the complexity is $O(\log n)$.

3.6 Delete

To delete an arbitrary node x from the heap.

- **Procedure:** This is achieved by combining `decrease-key` and `extract-min`. First, decrease the key of node x to negative infinity ($-\infty$). This will bubble x up to become the root of its Binomial Tree. Then, call `extract-min` to remove it.
- **Complexity:** $O(\log n)$ for decrease-key + $O(\log n)$ for extract-min = $O(\log n)$.

4 Complexity Analysis

The efficiency of Binomial Heaps makes them a good choice for applications requiring frequent merges.

4.1 Worst-Case Time Complexity Summary

4.2 Amortized Analysis Highlight

While the worst-case for an `insert` is $O(\log n)$, its amortized cost is much better. An insert costs $O(1)$ most of the time, only becoming expensive when a series of "carries" occurs during the union. Using the potential method of amortized analysis, where the potential function $\Phi(H)$ is the number of trees in the root list of H, the amortized cost of an insert can be shown to be $O(1)$.

Table 1: Comparison of Heap Data Structures

Operation	**Binary Heap**	**Binomial Heap**	**Fibonacci Heap**
Make-Heap	$O(1)$	$O(1)$	$O(1)$
Insert	$O(\log n)$	$O(\log n)$	$O(1)$
Find-Minimum	$O(1)$	$O(\log n)$	$O(1)$
Extract-Minimum	$O(\log n)$	$O(\log n)$	$O(\log n)*$
Union (Merge)	$O(n)$	$O(\log n)$	$O(1)$
Decrease-Key	$O(\log n)$	$O(\log n)$	$O(1)*$
Delete	$O(\log n)$	$O(\log n)$	$O(\log n)*$

$*$ Denotes amortized time complexity.

5 Conclusion

Binomial Heaps are a versatile and efficient mergeable heap structure. Their primary advantage over standard Binary Heaps is the highly efficient $O(\log n)$ `union` operation. While Fibonacci Heaps offer better amortized complexities for some operations, Binomial Heaps are conceptually simpler and perform well in practice, especially in applications that rely heavily on merging priority queues.