Binomial Heaps (With C++ Implementation)

Introduction

A binary heap (https://algorithmtutor.com/Data-Structures/Tree/Binary-Heaps/) has fast insert, delete-max (or delete-min), find maximum (or find minimum) operations. All of these operations run in O(log n) time. But if we want to merge two binary heaps, it takes at least a linear time ($\Omega(n)$). Therefore, binary heaps are inefficient in situations where we need to perform the merge operations frequently. There is another data structure which is as efficient as binary heaps in all above operations as well as supports fast merge or union operation. This data structure is called a Binomial Heap. A binomial heap is also called a mergeable heap or meldable heap because it provides an efficient merge operation. The table below shows the worst case complexity for the binary and binomial heap operations.

Operations	Binary Heap	Binomial Heap
Make-Heap	O(1)	O(1)
Insert	$\Theta(\log n)$	$O(\log n)$
Maximum (or Minimum)	$\Theta(1)$	$O(\log n)$
Delete Max (or Delete Min)	$\Theta(\log n)$	$\Theta(\log n)$
Merge (Union)	$\Theta(n)$	$O(\log n)$

The structure of the binomial heap allows the fast merge operation. It contains a collection of binomial trees. Before diving further into the binomial heap, let us first try to understand what a binomial tree is.

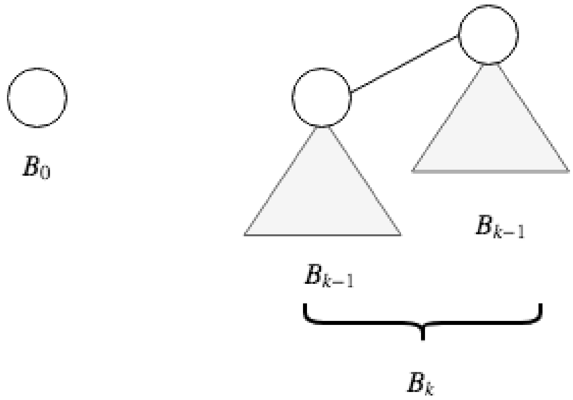
Binomial Tree

A binomial tree B_k is an ordered tree defined recursively.

The recursive definition is given below.

- The binomial tree of order 0 i.e. k = 0 is a single node.
- The binomial tree of order k is the two binomial trees of order $k - 1$ linked together: the root of one is the leftmost child of the root of another.

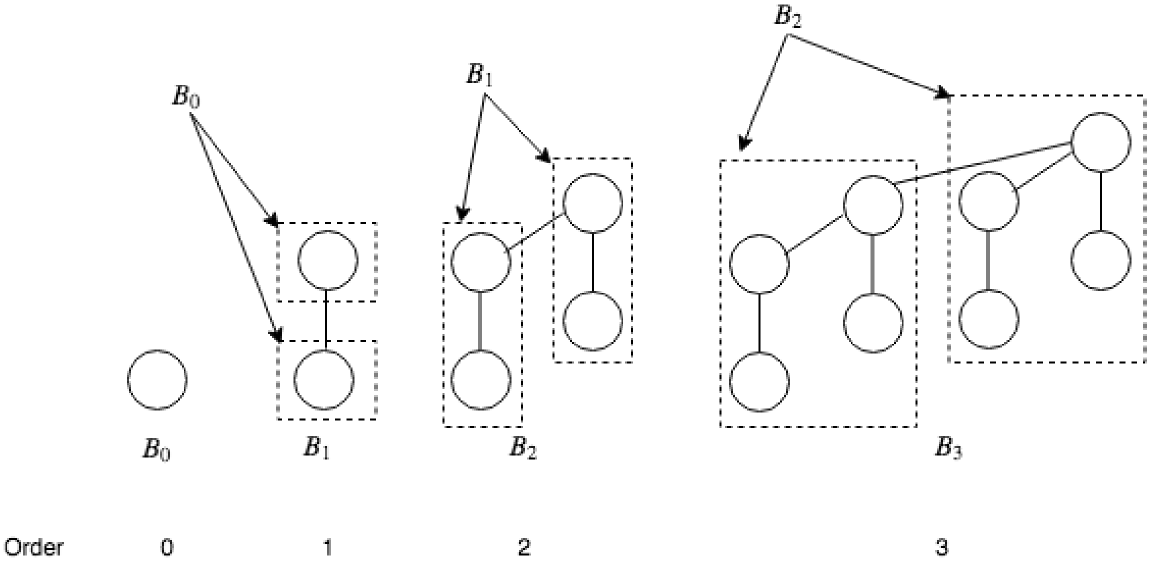
Figure 1 illustrates the recursive definition.



(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAASwAAAEsCAMA

Fig 1: Recursive definition of a binomial tree

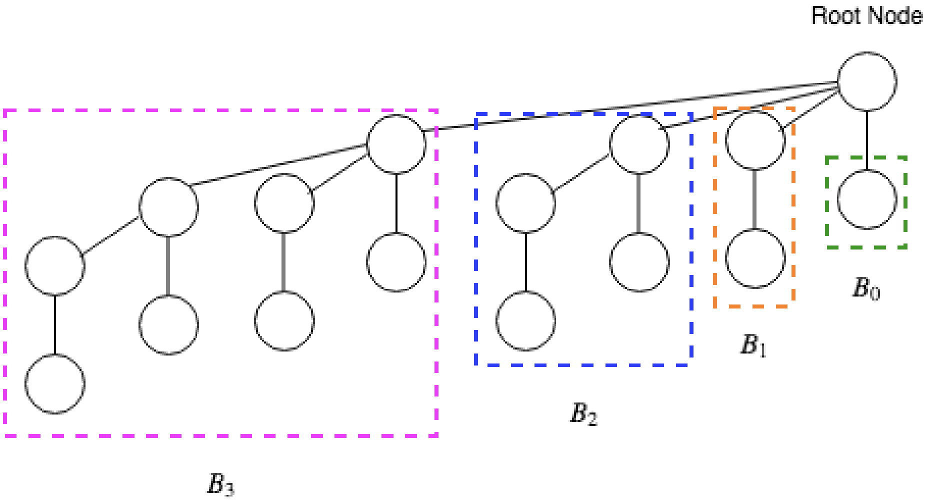
The binomial tree of order 1 is two binomial of trees of order 0 linked together. Similarly, the binomial tree of order 2 is two binomial trees of order 1 linked together and so on. Figure 2 shows the binomial trees of order 0, 1, 2, and 3.



(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAASwAAAEsCAMA

Fig 2: Binomial trees of order 0, 1, 2, and 3.

Alternatively, a binomial tree of order k is a tree whose children are the binomial trees of order $k-1, k-2, \ldots, 1, 0$. Figure 3 shows a binomial tree of order 4.



(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAAAEsCAMA

Fig 3: A binomial tree of order 4 whose children are the binomial trees of order 3, 2, 1, and 0.

Properties of Binomial tree

Following are the properties of a binomial tree.

For the binomial tree of degree k i.e. B_k

- There are 2^k nodes.
- The height of the tree is k.
- There are exactly $\binom{k}{i}$ nodes at depth i for $i = 0, 1, 2, \ldots. k$.
- If we delete the root node, we get $B_{k-1}, B_{k-2}, \ldots, B_1, B_0$ binomial trees (see fig 3).

Min-heap-ordered or Max-heap ordered binomial tree

A min-heap-ordered binomial tree is a binomial tree that obeys the min-heap property i.e. the parent node is smaller than or equal to its children nodes. Similarly, a max-heap-ordered binomial tree is a binomial tree that obeys the max-heap property i.e. the parent node is larger than or equal to its children nodes. Figure 4 shows an example of these binomial trees.

(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAAAEsCAMA

Fig 4: (a) An example of a min-ordered heap. (b) An example of a max-ordered heap.

Binomial Heap

A binomial heap H is a collection of binomial trees that satisfy the following binomial heap properties.

- Each binomial tree in H must be min-heap-ordered (or max-heap-ordered) binomial tree.
- For any non-negative integer k, there is at most one (either 0 or 1) binomial tree in H whose root has degree k.

The first property tells us that the root of a min-heap-ordered tree contains the smallest key in the tree. If there are m trees, then the smallest key can be found in $O(m)$ time.

The second property implies that an n-node binomial heap H consists of at most $\lfloor \log n \rfloor + 1$ binomial trees. We know that the number of nodes in a binomial tree of degree k is 2^k. If there are 3 binomial trees of degree 1, 4, and 7, the total number of nodes in a binomial tree consisting these three trees is $2^1 + 2^4 + 2^7$. We can write this as

$$0.2^0 + 1.2^1 + 0.2^2 + 0.2^3 + 1.2^4 + 0.2^5 + 0.2^6 + 1.2^7$$

This means the binomial heap has three trees whose roots are of degree 1, 4, and 7 and zero trees whose roots are other numbers than these three.

Figure 5 shows an example of a binomial heap consisting of three binomial trees of degree 0, 1 and 3.

(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAAAEsCAMA

Fig 5: A binomial heap containing three binomial trees of order 0, 1, and 3

The total number of nodes in the above binomial heap can be calculated as $2^0 + 2^1 + 2^3 = 11$.

Question 1: Find all the binomial trees in a binomial heap of 30 nodes
Solution: We need to find the combination of number in 2^k format that sums to 30.

$$30 = 16 + 8 + 4 + 2 = 2^4 + 2^3 + 2^2 + 2^1$$

This mean the heap contains 4 binomial trees of degree 4, 3, 2, 1.
Question 2: Find all the binomial trees in a binomial heap of 10 nodes.
Solution: Do yourself.

Representation

Each node in a binomial heap has 5 fields as follows.

1. `p` : A pointer to the parent node.
2. `data` : The key of the node.
3. `degree` : Number of children.
4. `child` : The pointer to the left-most child of the node.
5. `sibling` : The pointer to the right sibling of the node. In case of a root node, the sibling points to the root of another tree in the right.

Figure 6 shows a representation of a node in a binomial heap.

(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAAAEsCAMA

Fig 6: A representation of a node

Figure 7 shows the representation of a binomial tree given in Figure 5.

(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAAAEsCAMA

Fig 7: Representation of a tree given in Fig 5

Operations

In this section, all the operations in a binomial heap are described in detail.

Creating a new heap

This is the easiest operation. We simply point the `head` of the tree to `null`. The head always points to the leftmost tree of the heap. This operation takes $\Theta(1)$ time.

Finding the minimum

The root of a min-heap-ordered binomial tree contains the node with minimum data in it. If there are m such trees, we need to find the minimum of m items. If n is a total number of nodes in a binomial heap, there are at most $\lfloor \log n \rfloor + 1$ binomial trees. So the running time of this operation is $\Theta(\log n)$.

(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAAAEsCAMA

Fig 8: Finding the minimum of a binomial heap

In figure 8, the binomial heap has 3 binomial trees. The minimum value of the binomial heap is the smallest root among these three trees, which is 5.

Merging two binomial heaps

We repeatedly merge two binomial trees of the same degree until all the binomial trees have a unique degree. To merge two binomial trees of the same degree, we do the following.

1. Compare the roots of two trees (x and y). Find the smallest root. Let x is the tree with the smallest root.
2. Make the x’s root parent of y’s root.

Figure 9 illustrates this process.

(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAAAEsCAMA

Fig 9: Merging of two binomial trees with the same degree

This operation clearly takes constant time. Once we know how to merge two binomial trees with the same degree, we can merge two binomial heaps using the following steps.

1. Merge two binomial heaps without worrying about trees with the same degree. That means put the trees in the increasing order of degree.
2. Starting from the head, repeatedly merge trees with the same degree until all the trees in the heap have a unique degree.

Let me clarify these points with the help of examples. Consider two binomial heaps given in figure 10.

(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAAAEsCAMA

Fig 10: Two binomial heaps (a) and (b)

In the first step, we simply merge them without taking care of repeated degrees. After this merge, the trees in the heap must be arranged in an increasing order of degree. Figure 11 is the result of this process.

(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAAAEsCAMA

Fig 11: Merging of heaps in an increasing order of degree

Next, we start from the head and move towards the right direction. As soon as we see two trees with the same degree, we merge them. In figure 11, there are two trees of degree 2. Figure 12 shows the result of merging these two trees.

(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAAAEsCAMA

Fig 12: Merging of two trees of degree 2.

We repeat the similar process and merge two trees of degree 3. The final result of merging two binomial heaps is given in figure 13.

(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAAAEsCAMA

Fig 13: Final heap after merging

The complexity of merge operation is $O(\log n)$. There are at most $\lfloor \log n_1 + \log n_2 \rfloor + 2$ number of binomial trees (where n_1 is the number of nodes in the first heap and n_2 is the number of nodes in the second tree). We traverse the roots of these trees constant number of times. That gives the complexity of $O(\log n)$.

Insert a node

Inserting a node x into a heap H is a three steps process as follows.

1. Create a new empty binomial heap H'. It has a head pointer pointing to null.
2. Create a new node x with all the necessary fields. Point the head of the heap to x.
3. Merge this new heap H' with the existing heap H.

Figure 14 illustrates this process with an example.

(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAAAEsCAMA

Fig 14: Inserting a node

The complexity of inserting a new node is $O(\log n)$. Creating new heap takes only a constant amount of time and merging two heaps takes $O(\log n)$.

Delete Min (or Extract Min)

The steps for deleting the node from a heap H with the smallest key is given below.

1. Search through the roots of binomial trees and find the root with the smallest key and call it x. Remove the x from the tree.
2. Create a new empty heap H'.
3. Reverse the order of x's children and set the head of H' to point to the head of the resulting list.
4. Merge H' with H.

Figure 15 illustrates this process with an example.

(data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAASwAAAEsCAMA

Fig 15: Deleting node with the smallest key

The complexity of this operation is $O(\log n)$.

Implementation

I have implemented the code in C++. Because of its length, I find it inappropriate to paste here. It is available on GitHub (https://github.com/Bibeknam/algorithmtutorprograms/blob/master/data-structures/binomial-heaps/BinomialHeaps.cpp).

binomial heaps (/tags/binomial-heaps/) binomial trees (/tags/binomial-trees/) heaps (/tags/heaps/) mergeable heaps (/tags/mergeable-heaps/)

‹ Priority Queues (/Data-Structures/Tree/Priority-Queues/)

Binary Search Trees (/Data-Structures/Tree/Binary-Search-Trees/) ›

Contact Us (https://goo.gl/forms/qNqf8R99NZkKnKMD3) About Us