Binomial Heap

Objective: In this lecture we discuss binomial heap, basic operations on a binomial heap such as insert, delete, extract-min, merge and decrease key followed by their asymptotic analysis, and also the relation of binomial heap with *binomial co-efficients*.

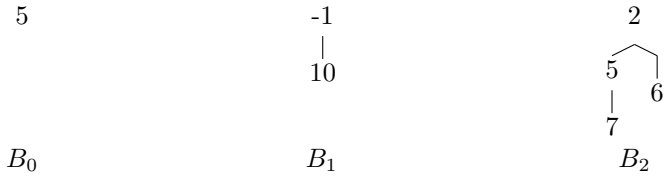
Motivation: Is there a data structure that supports operations insert, delete, extract-min, merge and decrease key efficiently. Classical min-heap incurs $O(n)$ for merge and $O(\log n)$ for the rest of the operations. Is it possible to perform merge in $O(\log n)$ time.

1 Binomial Tree

We shall begin our discussion with binomial trees. Further, we study structural properties of binomial trees in detail and its relation to binomial heaps. Binomial tree is recursively defined as follows;

1. A single node is a binomial tree, which is denoted as B_0
2. The binomial tree B_k consists of two binomial trees B_{k-1}, $k \geq 1$.
3. Since we work with min binomial trees, when two B_{k-1}'s are combined to get one B_k, the B_{k-1} having minimum value at the root will be the root of B_k, the other B_{k-1} will become the child node.

Eg:



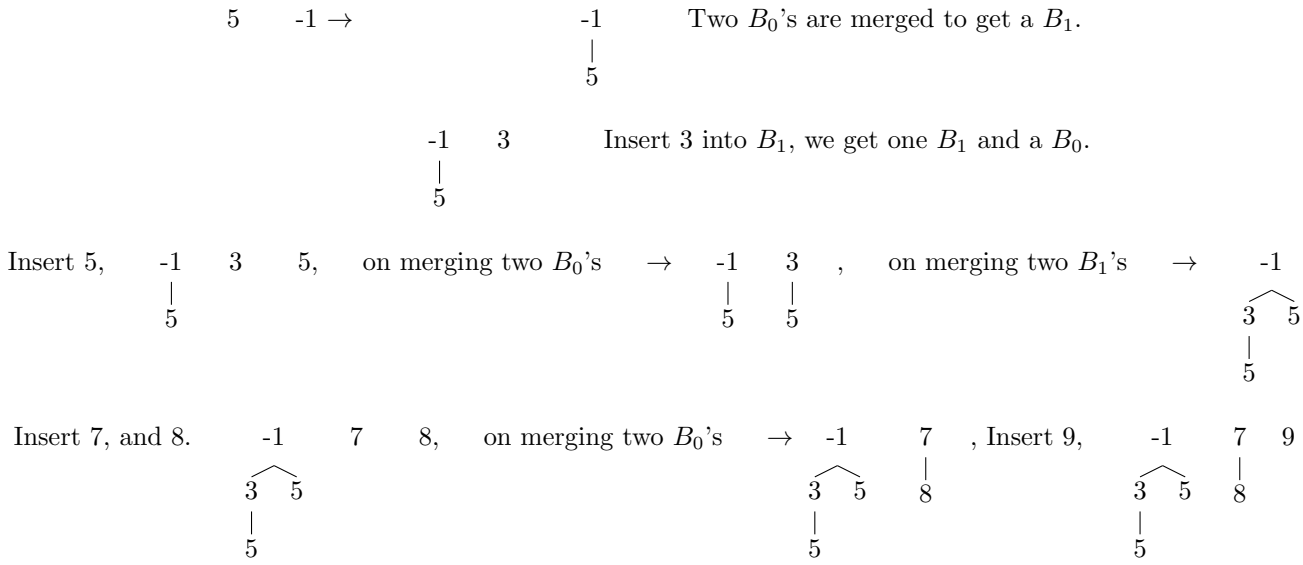
Structural Properties:

For the binomial tree B_k,

1. There are 2^k nodes.
2. The height of the binomial tree is k.
3. There are exactly $\binom{k}{i}$ nodes at depth $i = 0, 1, \ldots, k$.
4. The root has degree k, which is greater than that of any other node, moreover if the children of the root are numbered from left to right by $k-1, k-2, \ldots, 0$, child i is the root of the subtree B_i.

Note: Due to Property 3, it gets the name binomial tree (heap).
Eg:

$$\left[\mathbf{5} \quad \mathbf{-1} \quad \mathbf{3} \quad \mathbf{5} \quad \mathbf{7} \quad \mathbf{8} \quad \mathbf{9}\right]$$



Binomial Tree Construction

Proof of Property 1: Mathematical induction on k. The binomial tree B_0 is the base binomial tree for $k = 0$. Clearly, by definition, B_0 is a single node. Consider a binomial tree B_k, $k \geq 1$. Since B_k is constructed using two copies of B_{k-1}, by the hypothesis, each B_{k-1} has 2^{k-1} nodes. Thus, B_k has $2^{k-1} + 2^{k-1} = 2^k$ nodes. Hence the claim.

Proof of Property 2: Mathematical induction on k. Clearly, B_0 has height '0'. By the hypothesis, B_{k-1} has height $k-1$. For B_k, one of the B_{k-1}'s becomes the root and hence the height increases by one when the other B_{k-1} is attached. Thus, the height of B_k is $k - 1 + 1 = k$.

Proof of Property 3: Let $D(k, i)$ be the number of nodes at depth i for a binomial tree of degree k. Since B_k is constructed using two copies of B_{k-1}, the nodes at depth $(i-1)$ of B_{k-1} becomes the nodes at depth i for B_k. Therefore,

$$
\begin{aligned}
D(k,i) &= D(k-1,i) + D(k-1,i-1); \\
&= {}^{k-1}C_i + {}^{k-1}C_{i-1}; \\
&= \frac{(k-1)!}{i!(k-1-i)!} + \frac{(k-1)!}{(i-1)!(k-1-i+1)!}; \\
&= \frac{(k-1)!}{(k-i-1)!(i-1)!}\left[\frac{1}{i} + \frac{1}{k-i}\right]; \\
&= \frac{k!}{i!(k-i)!}; \\
&= {}^{k}C_i
\end{aligned}
$$

Proof of Property 4: Follows from the recursive definition of B_k.

2 Binomial Heap

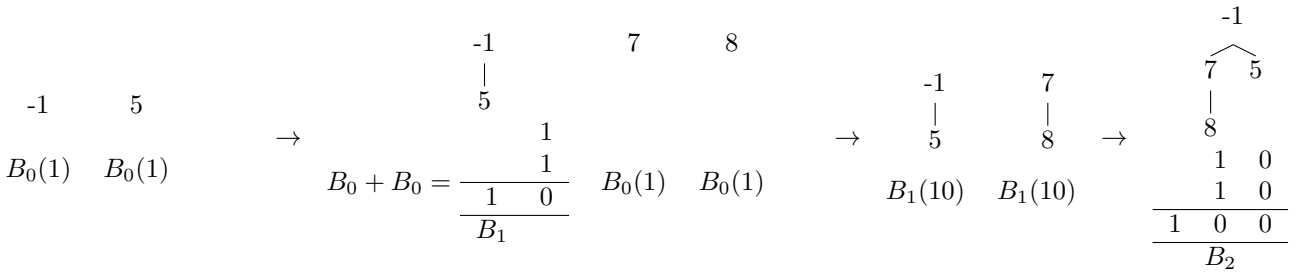
In this section, we shall discuss the construction of min binomial heap, time-complexity analysis, and various operations that can be performed on a binomial heap along with its analysis.
A **Min Binomial Heap** H is a collection of distinct min binomial trees. For each $k \geq 0$, there is at most one min binomial tree in H whose root has degree k.

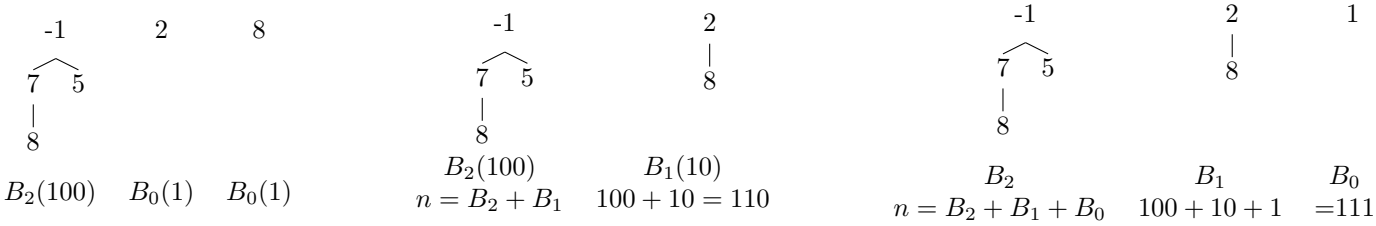
Observation 1: An n-node min binomial heap consists of at most $\lfloor \log n \rfloor + 1$ binomial trees.

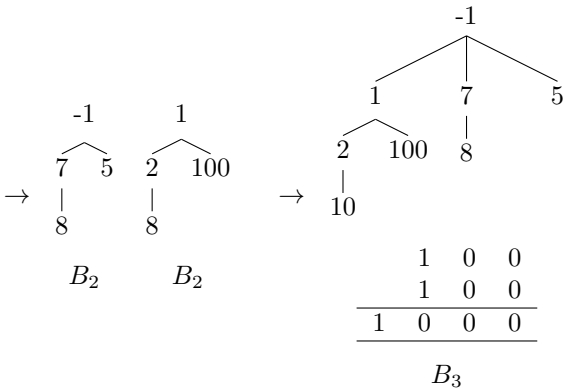
Observation 2: A binomial heap on n nodes and a binary representation of n has a relation. Binary representation of n requires $\lfloor \log n \rfloor + 1$ bits. Adding a node into a binomial heap H is equivalent to adding a binary '1' to the binary representation of H.

We now present an example illustrating the construction of binomial heap and its relation to binary representation. For B_i, the value given in parenthesis is the binary representation of the number of nodes ($n = 2^i$) in B_i.

$$\left[\mathbf{-1} \quad \mathbf{5} \quad \mathbf{7} \quad \mathbf{8} \quad \mathbf{2} \quad \mathbf{8} \quad \mathbf{100} \quad \mathbf{1}\right]$$







Note:

- In the above example, for $n = 8$, the final binomial heap has $B_3 = 1$ and $B_2 = B_1 = B_0 = 0$ which is 1 0 0 0, the binary representation of 8.
- For $n = 17$, the binomial heap consists of one B_4 and B_0, which corresponds to the binary representation of 1 0 0 0 0 1.

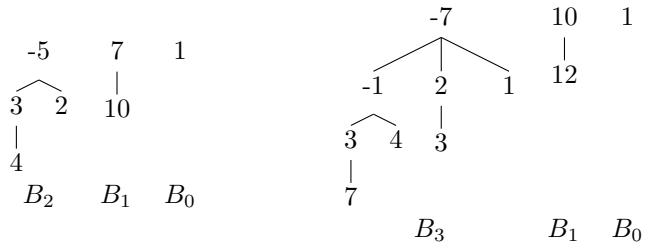
Insertion

Inserting a node into a binomial heap H is equivalent to adding a binary '1' to the binary representation of H. In the worst case, the newly inserted node B_0 triggers merge at each iteration, i.e., inserting B_0 creates a new B_1 which inturn creates a new B_2 and so on. Thus, insert requires $O(\log n)$ operations.

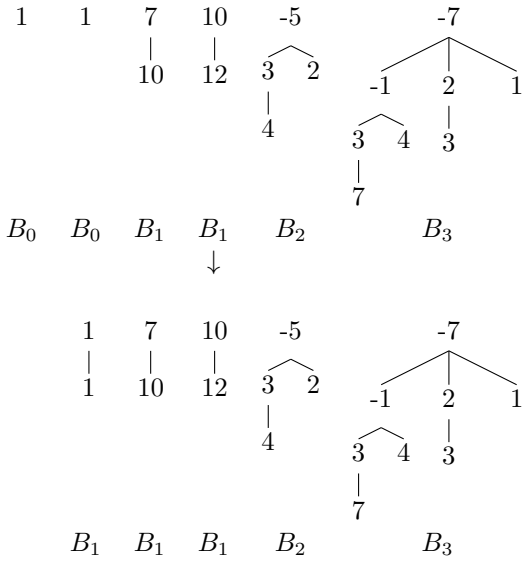
Merge

Merging two binomial heaps H_1 and H_2 is equivalent to adding two binary numbers. In particular, adding the binary representation of $|H_1|$ and $|H_2|$. In the worst case, every bit addition generates a carry which is equivalent to creating a new B_i while merging a copy of B_{i-1} in H_1 and a copy of B_{i-1} in H_2. Thus, merge incurs $O(\log n)$ in the worst case, where $n = |H_1| + |H_2|$. An example is illustrated below:

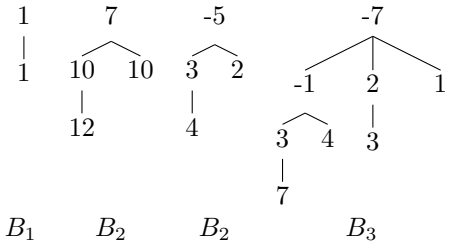
Binomial heap H_1 Binomial heap H_2



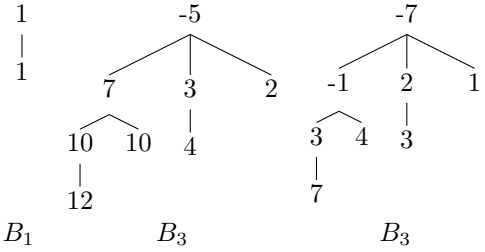
1. If B_0 is present in one of the heaps, then do nothing. Otherwise, merge two copies of B_0 and create one B_1. In general, merge two copies of B_i and create a copy of B_{i+1}.



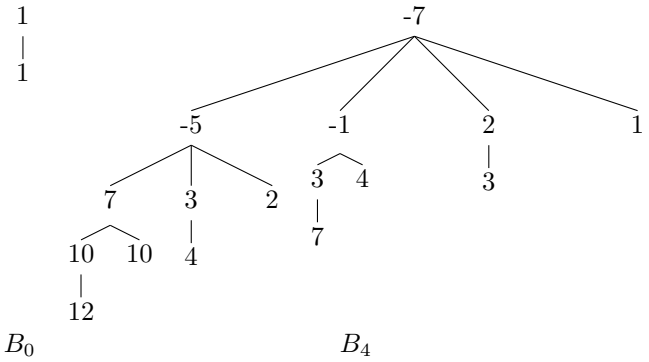
2. On merging we may get three copies of B_1, leave the first B_1 and merge the last two to obtain one B_2.



3. Now, two B_2 exists. Whenever, more than two copies of B_i exists, leave the first one and merge the last two.



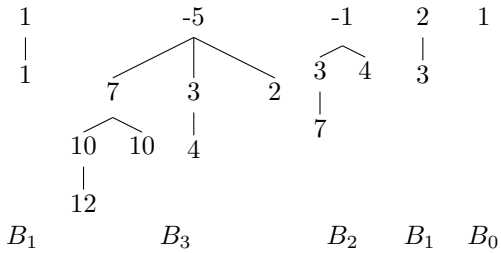
4. Now, merge two B_3.



There are 18 nodes, binary representation = $\begin{matrix} 1 & 0 & 0 & 1 & 0 \\ B_4 & B_3 & B_2 & B_1 & B_0 \end{matrix}$

Extract Min

Let B_k be the node containing the minimum of a binomial heap H. By construction, B_k contains $B_{k-1}, \ldots, B_0$ as its children. On extracting minimum, we invoke Merge() routine with H_1 being $B_{k-1}, \ldots, B_0$ and H_2 being the remaining nodes in H (except B_k). Thus, extract minimum incurs $O(\log n)$ in the worst case. Suppose, we perform extract min on the above 18-node binomial heap, we get
After extracting -7 ,



Now we perform merge on the above binomial heap so that each B_i occurs at most once.

Decrease Key

For decrease key, the value of the node pointed by the pointer x is decreased to the desired value y. If y is smaller than its parent, i.e., on performing decrease key min binomial heap property is still maintained, then no further modification is required. Otherwise, min-heapify() routine is called to set right the min-heap property. Since the height of the binomial heap is $k = \log n$, the decrease key in worst case takes $O(\log n)$ comparisons.

Delete

To perform delete we make use decrease key and extract min subroutines. The node to be deleted is decreased to $-\infty$ (or choose a value which is smaller than the current minimum), followed by extract min. Clearly, this incurs $O(\log n)$.

Summary

In this lecture, we have discussed in detail a variant of min-heap, namely, min binomial heap using which one can perform the following operations efficiently.

- Insert and extract min can be done in $O(\log n)$ time.
- Merging of two heaps can be done in $O(\log n)$ in worst case, whereas classical heap incurs $O(n)$.
- Decrease key and delete can be performed in $O(\log n)$ time.