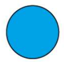
Binomial Tree X section

11 February 2025 09:00

A **Binomial Tree** B_k is an ordered tree defined recursively, where k represents the order of the binomial tree.

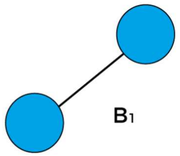
- If the binomial tree is of order **0** (B_0), it consists of a single node.
- In general, a binomial tree of order **k** (B_k) consists of **two binomial trees** of order $k-1$, where one is linked as the **left subtree** of the other.

If B_0, where k is 0, there would exist only one node in the tree.



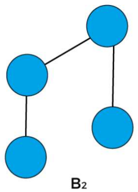
B_0

If B_1, where k is 1. Therefore, there would be two binomial trees of B_0 in which one B_0 becomes the left subtree of another B_0.



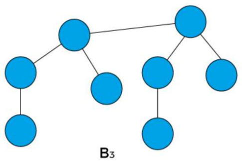
B_1

If B_2, where k is 2. Therefore, there would be two binomial trees of B_1 in which one B_1 becomes the left subtree of another B_1.

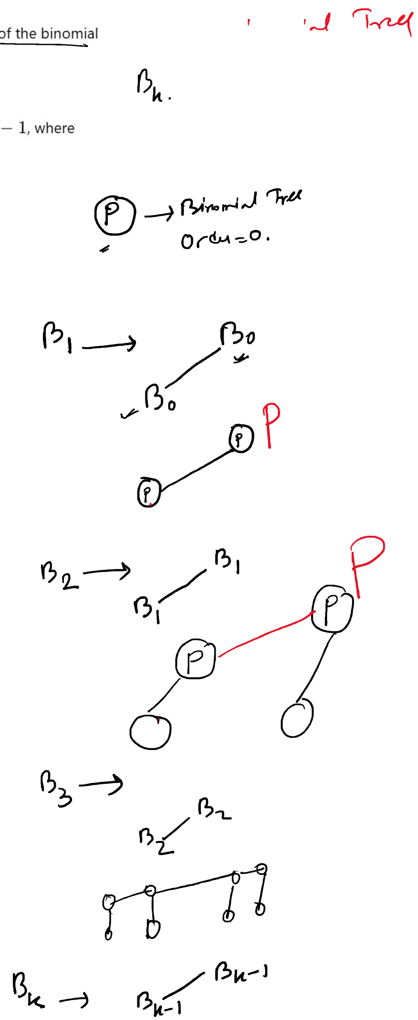


B_2

If B_3, where k is 3. Therefore, there would be two binomial trees of B_2 in which one B_2 becomes the left subtree of another B_2.



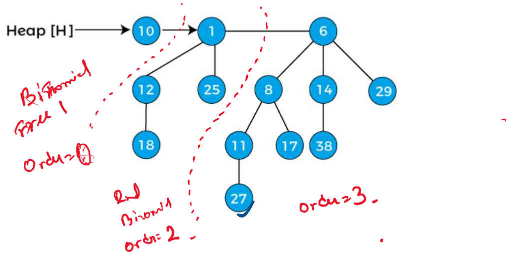
B_3



A binomial heap is a collection of binomial trees that satisfies the following binomial heap properties:

1. No two binomial trees in the collection have the same order.
2. Every binomial tree in the heap must follow the min-heap property, i.e., the value of a child node is greater than parent node.

2. Every binomial tree in the heap must follow the min-heap property, i.e., the value of a child node is greater than parent node.



Binomial Heap Union Operation

To perform the union of two binomial heaps, we have to consider the below cases -

Case 1: If degree[x] is not equal to degree[next x], then move pointer ahead.

Case 2: if degree[x] = degree[next x] = degree[sibling(next x)] then,

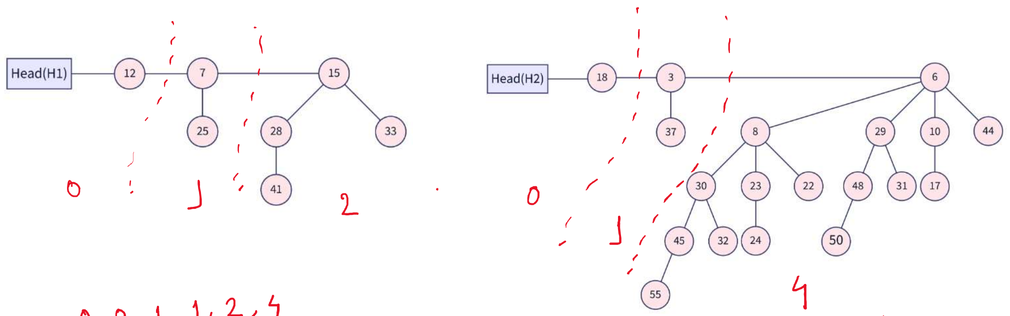
Move the pointer ahead.

Case 3: If degree[x] = degree[next x] but not equal to degree[sibling[next x]]

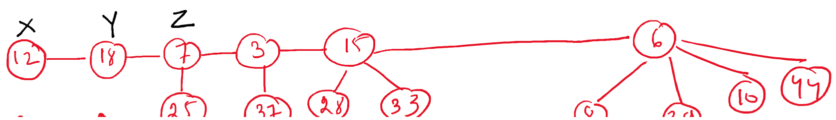
and key[x] < key[next x] then remove [next x] from root and attached to x.

Case 4: If degree[x] = degree[next x] but not equal to degree[sibling[next x]]

and key[x] > key[next x] then remove x from root and attached to [next x].



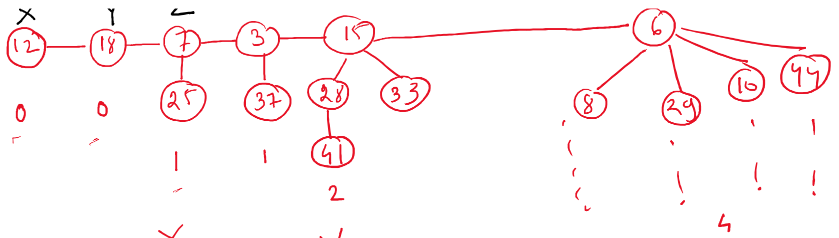
0, 0, 1, 1, 2, 4



Case-1,

~~X = Y = Z~~

order(X) = order(Y) = order(Z)



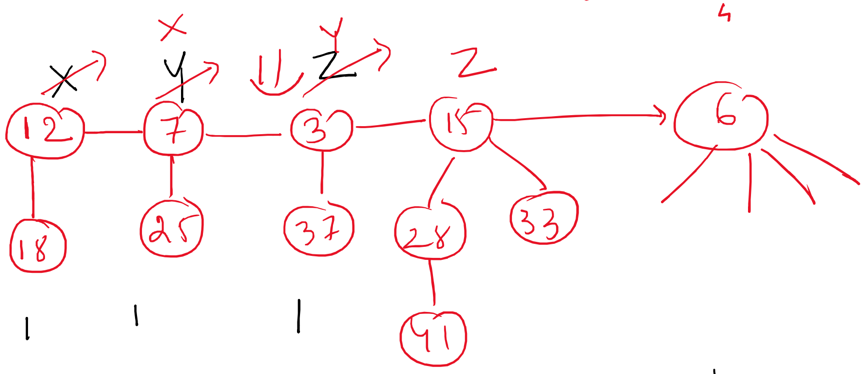
~~X = Y = Z~~

order(x) = order(Y) = order(z)

Move pointers ahead by 1

Case 2 - order(x) ≠ order(y)

Move pointers ahead by 1



Case 3:

order(x) = order(y) ≠ order(z)

key(x) <= key(y)

x → left = add y.