A Binary Tree is **perfect** or **height-balanced** if every leaf is at the same level, which is equivalent to every path from a given node to a leaf being the same length; it is **full** if every internal (non-leaf) node has 2 children(Not necessarily all leaves are at the same level, but every internal node must have two children.); it is **complete** (some time call **nearly complete )** if is perfect and all levels but the last are full, and in the last level leaves are as far left as possible (so any "vacancies" are to the right); it is **degenerate** if every non-leaf node has just one child (and as a graph it is a path from the root to the one leaf).

complete include perfect

Sure, let's work through an example of building a heap using the bottom-up method.

Given an array: **[2, 8, 6, 1, 10, 15, 3, 12, 11]**, let's create a max-heap using the bottom-up method.

**Step 1: Determine the last parent node**

For a heap, the last parent node is at index **(n - 2) // 2**, where **n** is the number of elements in the array.

* With **n = 9**, the last parent node is at index **(9 - 2) // 2 = 3**.

**Step 2: Apply heapify from the last parent to the root**

Now we apply heapify to each parent node, from the last parent to the root, to ensure that the max-heap property is maintained. Heapify involves ensuring that the current node is greater than its children. If not, we swap with the largest child and continue heapifying.

* **Node at Index 3** (**1**):
  + Children: Index 7 (**12**), Index 8 (**11**).
  + Swap with the larger child: Swap **1** with **12**.
  + New array: **[2, 8, 6, 12, 10, 15, 3, 1, 11]**.
* **Node at Index 2** (**6**):
  + Children: Index 5 (**15**), Index 6 (**3**).
  + Swap with the larger child: Swap **6** with **15**.
  + New array: **[2, 8, 15, 12, 10, 6, 3, 1, 11]**.
* **Node at Index 1** (**8**):
  + Children: Index 3 (**12**), Index 4 (**10**).
  + Swap with the larger child: Swap **8** with **12**.
  + New array: **[2, 12, 15, 8, 10, 6, 3, 1, 11]**.
* **Node at Index 0** (**2**):
  + Children: Index 1 (**12**), Index 2 (**15**).
  + Swap with the larger child: Swap **2** with **15**.
  + New array: **[15, 12, 2, 8, 10, 6, 3, 1, 11]**.
  + Since the node at index 2 (**2**) now might not be in order, we need to heapify it:
    - Children: Index 5 (**6**), Index 6 (**3**).
    - Swap with the larger child: Swap **2** with **6**.
    - New array: **[15, 12, 6, 8, 10, 2, 3, 1, 11]**.

Thus, the resulting array **[15, 12, 6, 8, 10, 2, 3, 1, 11]** is now a max-heap built using the bottom-up method.