BTree

The index leaf nodes are stored in an arbitrary orderSmithRobinsonin short: the B-tree.

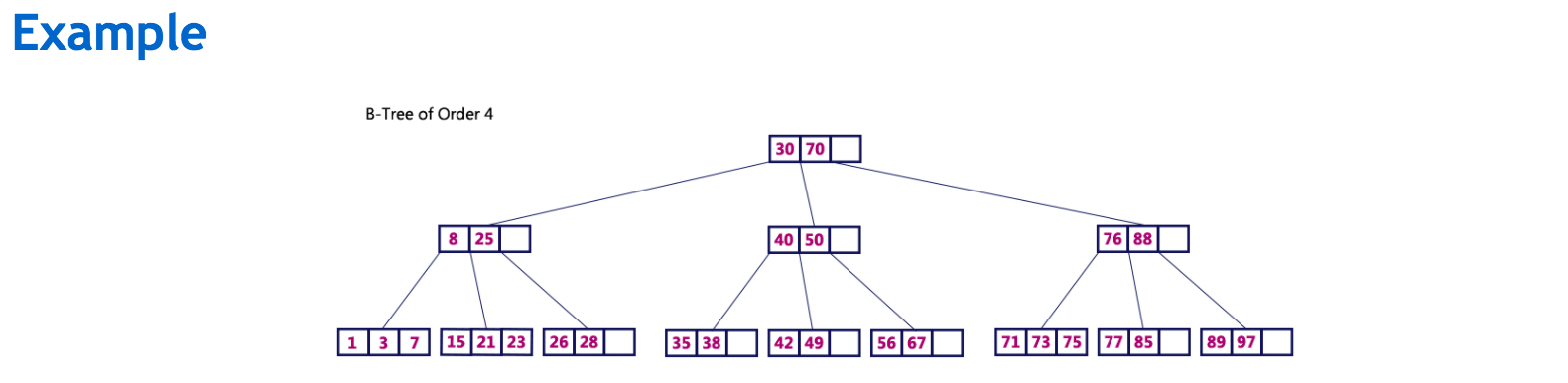
In a binary search tree, AVL Tree, Red-Black tree etc., every node can have only one value (key) and maximum of two children but there is another type of search tree called B-Tree in which a node can store more than one value (key) and it can have more than two children. B-Tree was developed in the year of 1972 by Bayer and McCreight with the name Height Balanced m-way Search Tree. Later it was named as B-Tree.

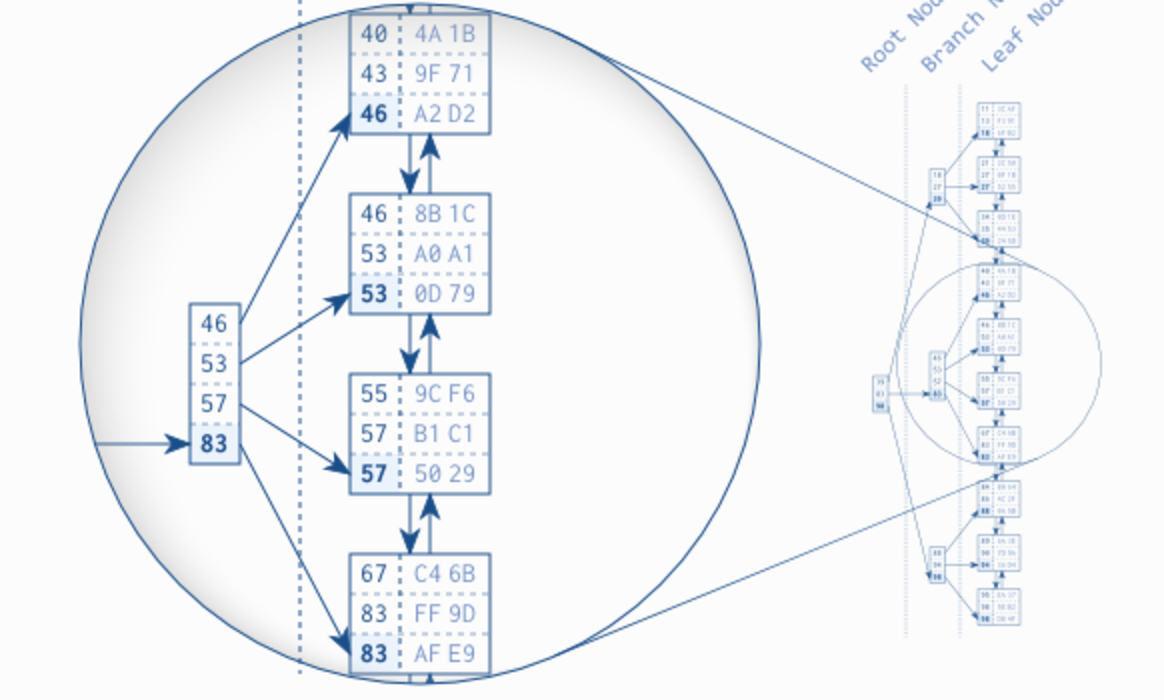
BTree is not binary tree, but a balanced tree. **B-Tree is a self-balanced search tree with multiple keys in every node and more than two children for every node.**

Here, number of keys in a node and number of children for a node is depend on the order of the B-Tree. Every B-Tree has order.  
  
B-Tree of Order m has the following properties...

* **Property #1** - All the leaf nodes must be at same level.
* **Property #2** - All nodes except root must have at least [m/2]-1 keys and maximum of m-1 keys.
* **Property #3** - All non leaf nodes except root (i.e. all internal nodes) must have at least m/2 children.
* **Property #4** - If the root node is a non leaf node, then it must have at least 2 children.
* **Property #5** - A non leaf node with n-1 keys must have n number of children.
* **Property #6** - All the key values within a node must be in Ascending Order.

For example, B-Tree of Order 4 contains maximum 3 key values in a node and maximum 4 children for a node.

****

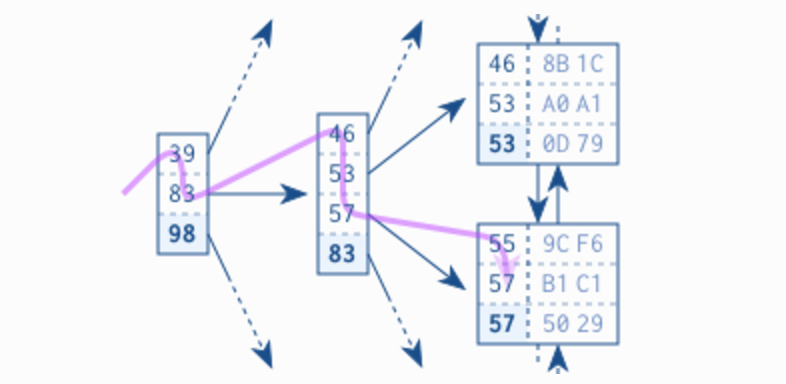


Each node is arbitrary number of values, and it is in accending order. The siblings is connected by doubly linkedlist.

The figure highlights a branch node and the leaf nodes it refers to. Each branch node entry corresponds to the biggest value in the respective leaf node. Take the first leaf node as an example: the biggest value in this node is 46, which is thus stored in the corresponding branch node entry. The same is true for the other leaf nodes so that in the end the branch node has the values 46, 53, 57 and 83. According to this scheme, a branch layer is built up until all the leaf nodes are covered by a branch node.

The next layer is built similarly, but on top of the first branch node level. The procedure repeats until all keys fit into a single node, the *root node*. The structure is a *balanced search tree* because the tree depth is equal at every position; the distance between root node and leaf nodes is the same everywhere.

## B-Tree Traversal



The tree traversal is a very efficient operation—so efficient that I refer to it as the *first power of indexing*. It works almost instantly—even on a huge data set. That is primarily because of the tree balance, which allows accessing all elements with the same number of steps, and secondly because of the logarithmic growth of the tree depth. That means that the tree depth grows very slowly compared to the number of leaf nodes. Real world indexes with millions of records have a tree depth of four or five. A tree depth of six is hardly ever seen. The box [“](http://use-the-index-luke.com/sql/anatomy/the-tree" \l "sb-log)*Logarithmic Scalability*” describes this in more detail.

The following operations are performed on a B-Tree...

1. Search
2. Insertion
3. Deletion

# Search Operation in B-Tree

In a B-Ttree, the search operation is similar to that of Binary Search Tree. In a Binary search tree, the search process starts from the root node and every time we make a 2-way decision (we go to either left subtree or right subtree). In B-Tree also search process starts from the root node but every time we make n-way decision where n is the total number of children that node has. In a B-Ttree, the search operation is performed with O(log n) time complexity. The search operation is performed as follows...

* Step 1: Read the search element from the user
* Step 2: Compare, the search element with first key value of root node in the tree.
* Step 3: If both are matching, then display "Given node found!!!" and terminate the function
* Step 4: If both are not matching, then check whether search element is smaller or larger than that key value.
* Step 5: If search element is smaller, then continue the search process in left subtree.
* Step 6: If search element is larger, then compare with next key value in the same node and repeate step 3, 4, 5 and 6 until we found exact match or comparision completed with last key value in a leaf node.
* Step 7: If we completed with last key value in a leaf node, then display "Element is not found" and terminate the function.

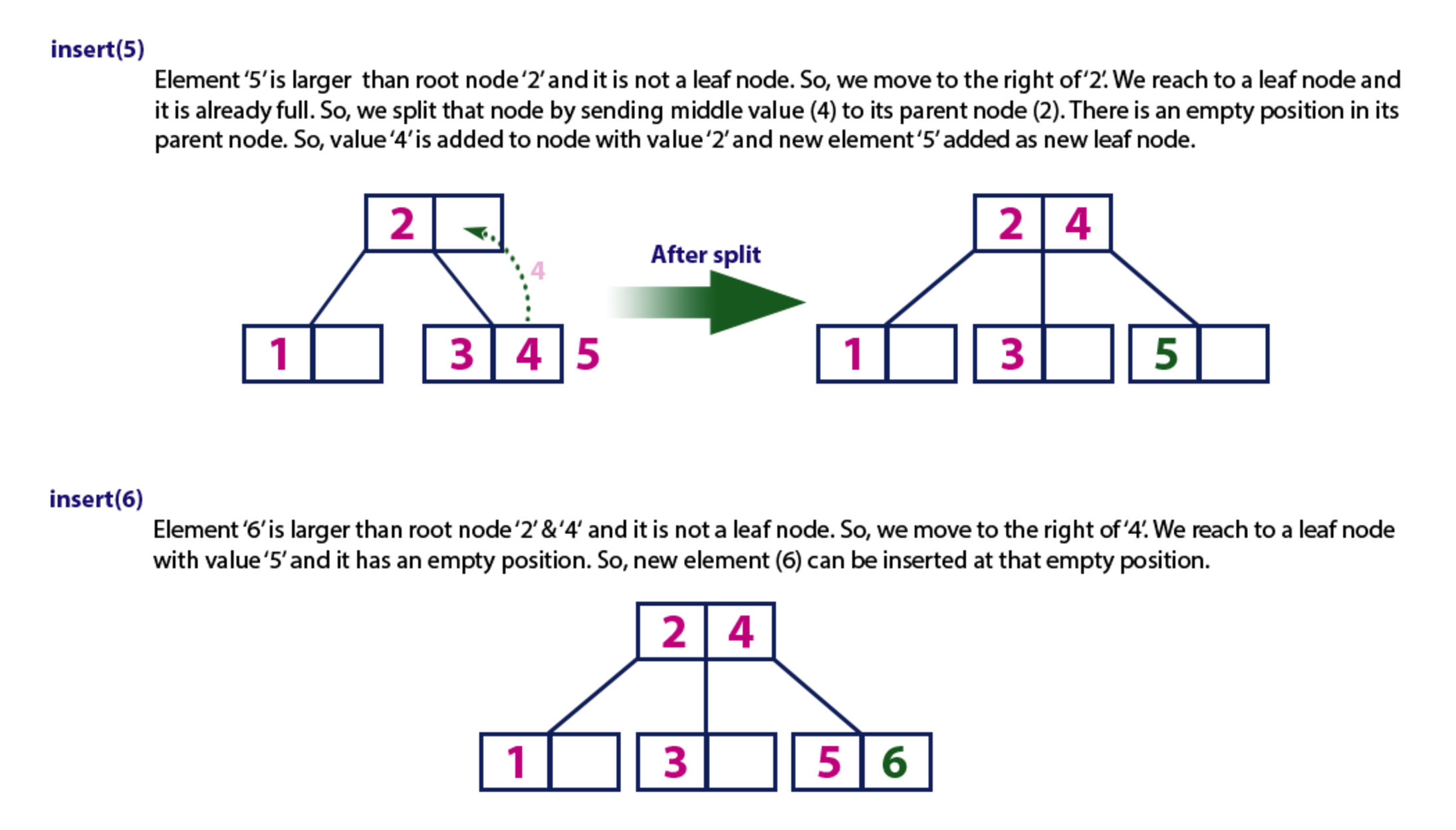
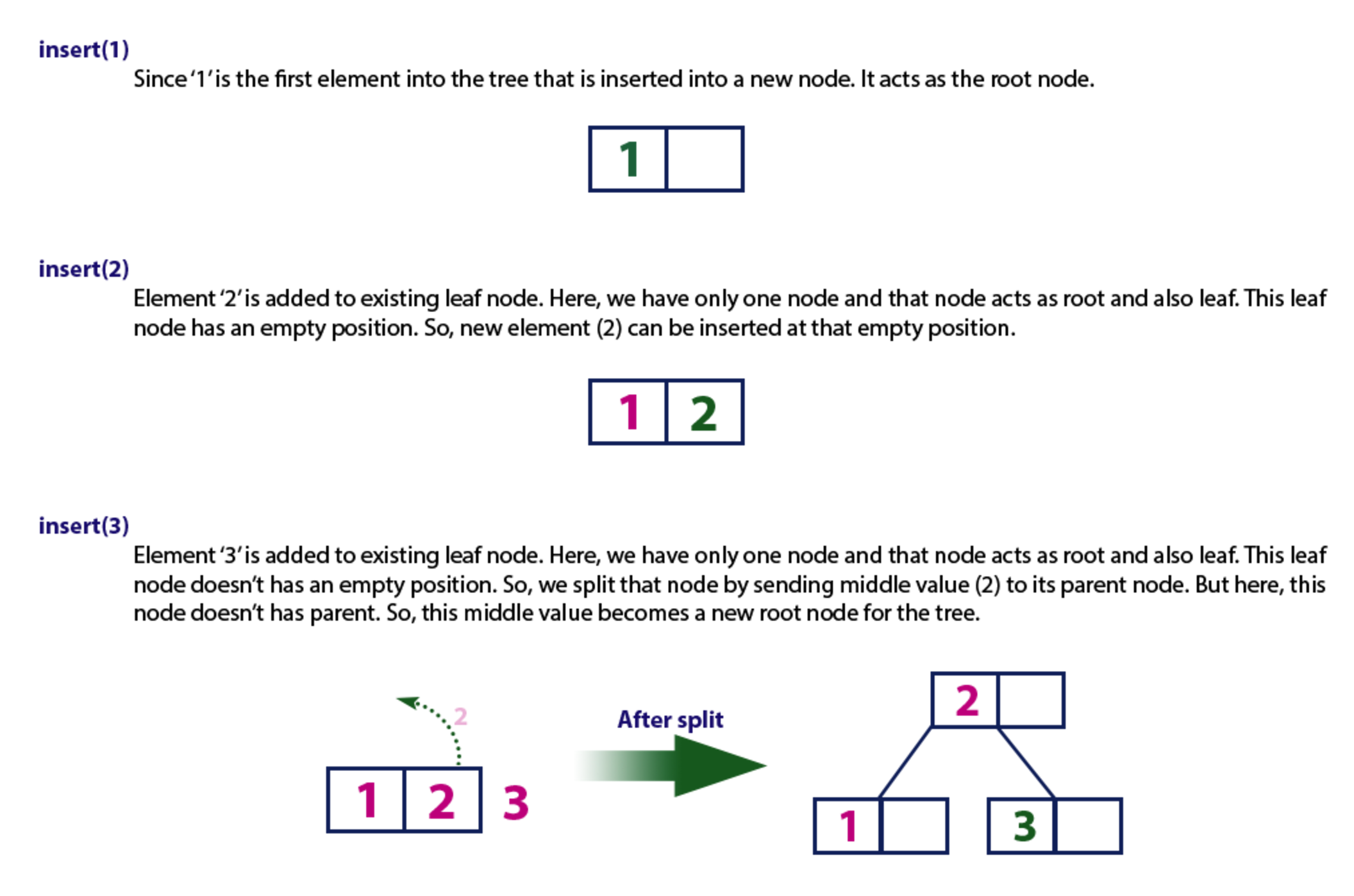
# Insertion Operation in B-Tree

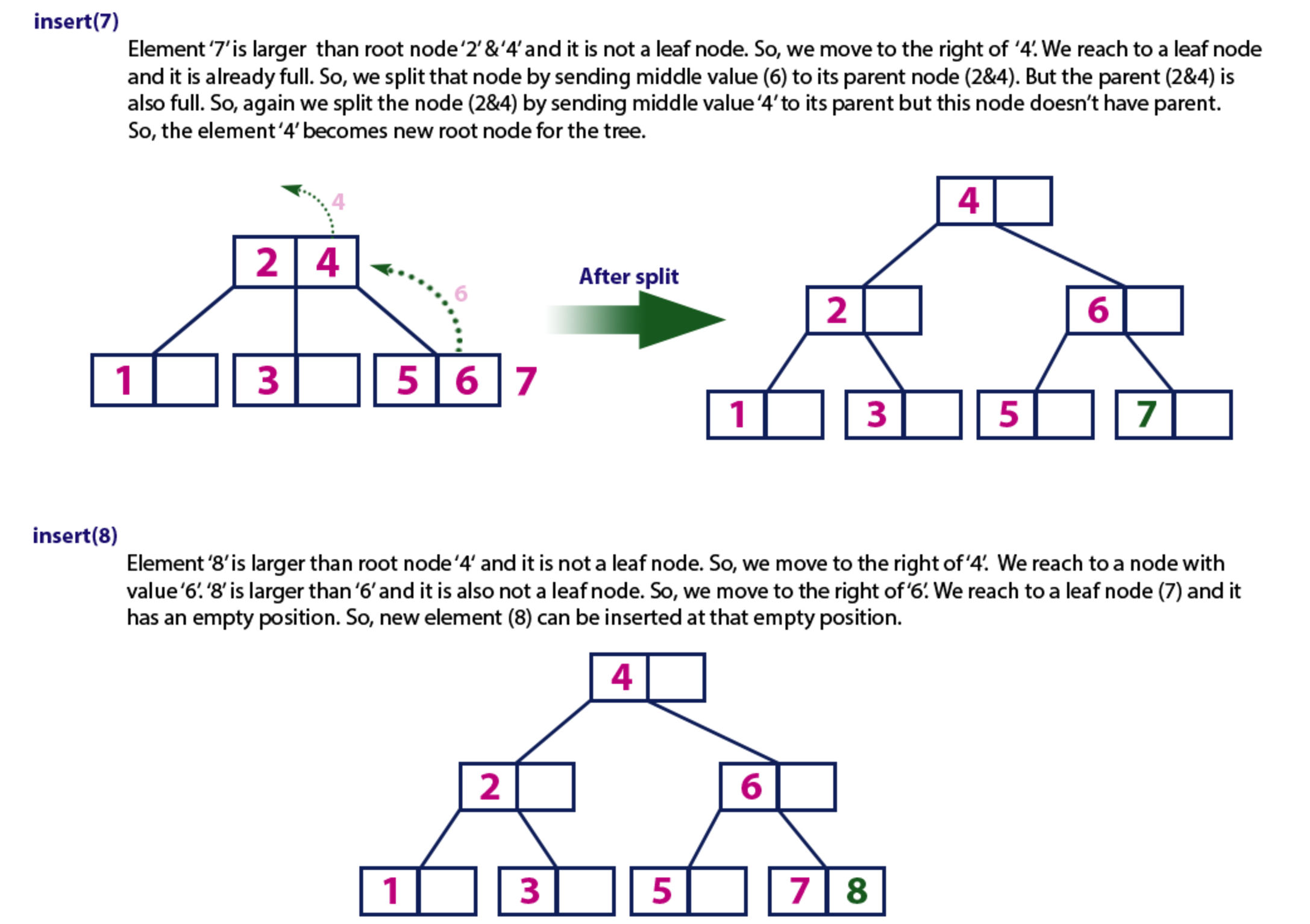
In a B-Tree, the new element must be added only at leaf node. That means, always the new keyValue is attached to leaf node only. The insertion operation is performed as follows...

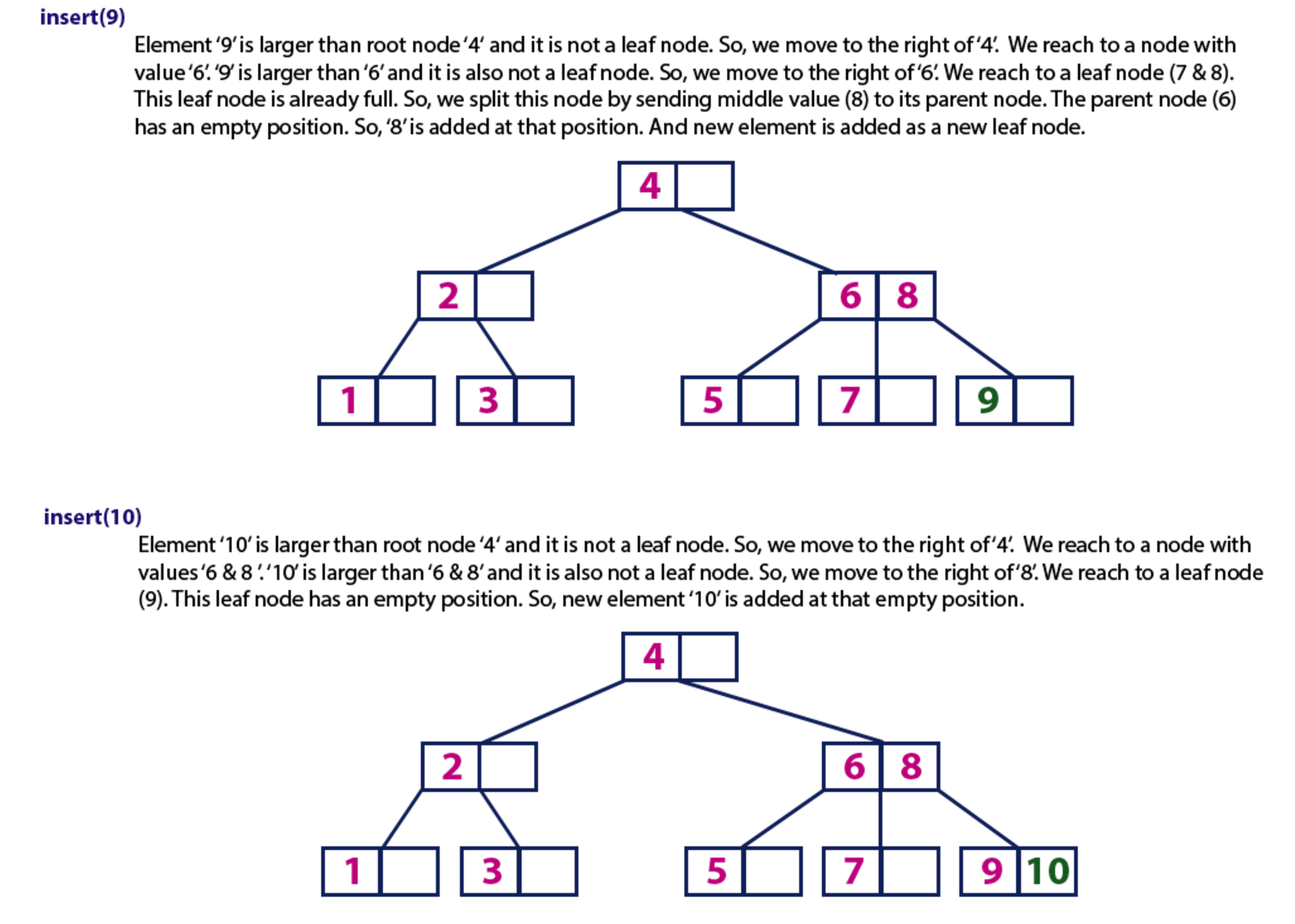
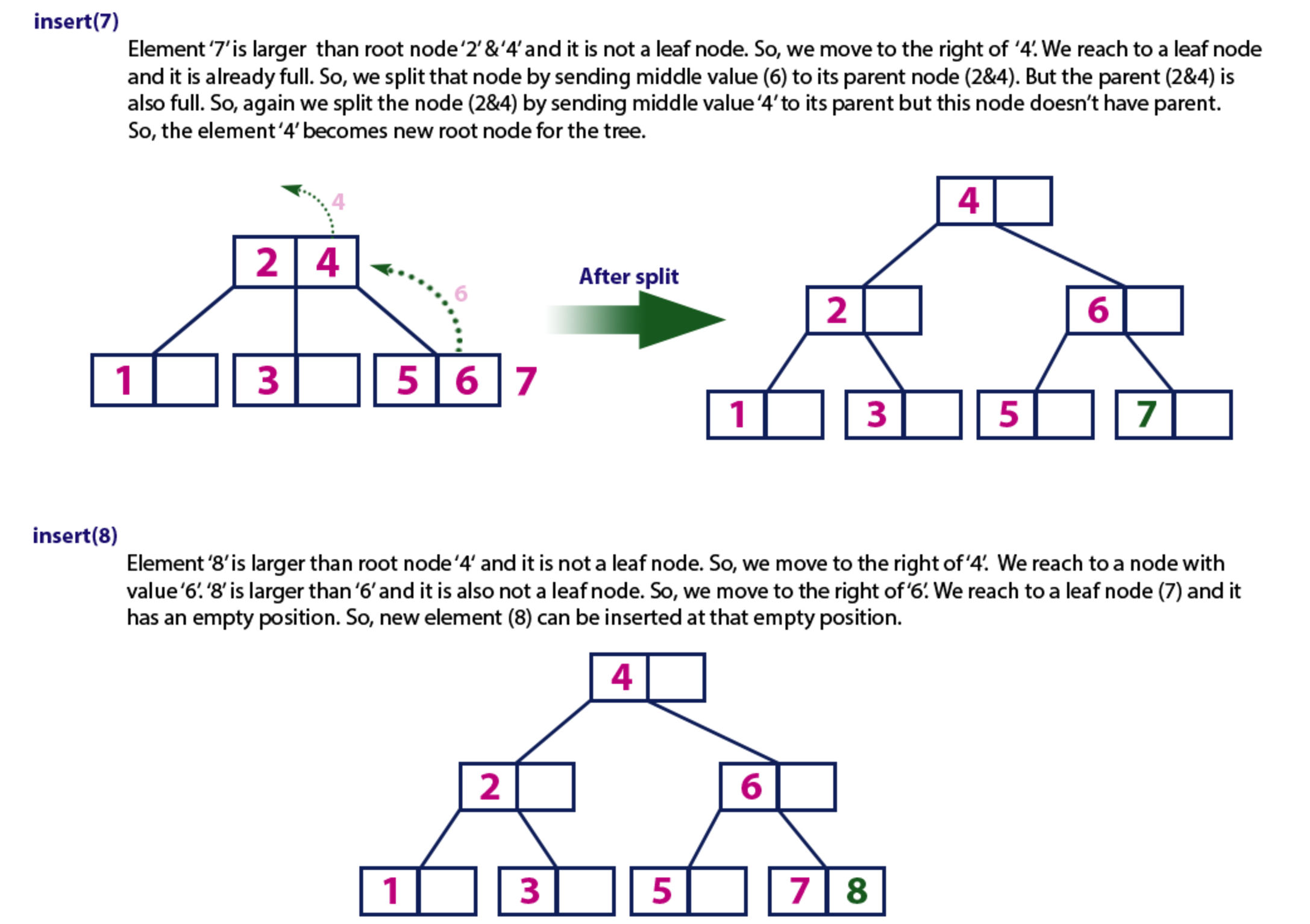
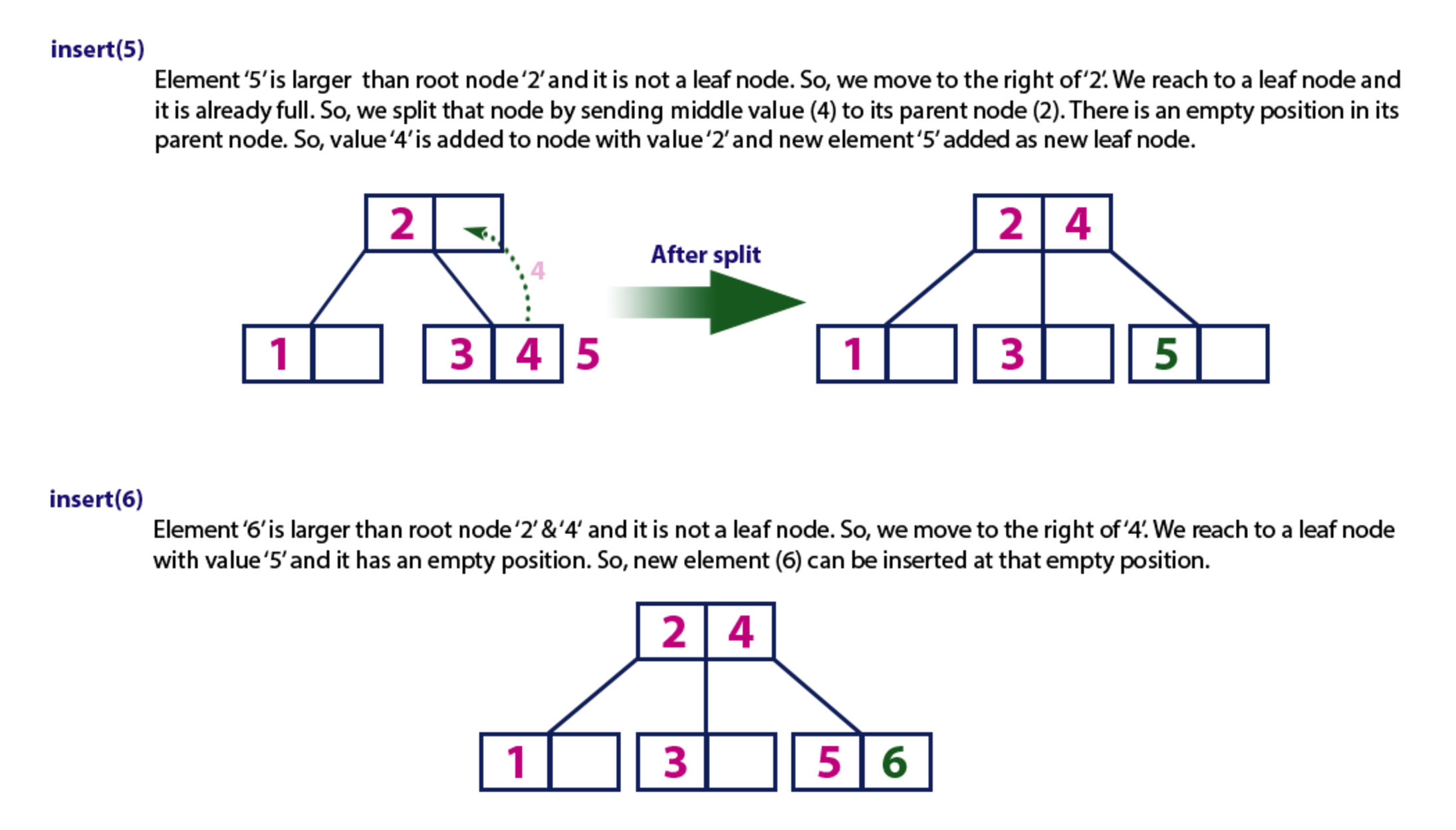
* Step 1: Check whether tree is Empty.
* Step 2: If tree is Empty, then create a new node with new key value and insert into the tree as a root node.
* Step 3: If tree is Not Empty, then find a leaf node to which the new key value cab be added using Binary Search Tree logic.
* Step 4: If that leaf node has an empty position, then add the new key value to that leaf node by maintaining ascending order of key value within the node.
* Step 5: If that leaf node is already full, then split that leaf node by sending middle value to its parent node. Repeat tha same until sending value is fixed into a node.
* Step 6: If the spilting is occuring to the root node, then the middle value becomes new root node for the tree and the height of the tree is increased by one.

# Example

Construct a B-Tree of Order 3 by inserting numbers from 1 to 10. Note that the following just take special set of data which is accending, it does not guarantee it is sorted or looks like bst.







Ref:

<http://use-the-index-luke.com/sql/anatomy/the-tree>

<http://btechsmartclass.com/DS/U5_T3.html>

<http://www.cs.cornell.edu/courses/cs3110/2011sp/recitations/rec24-B-trees/B-trees.htm><http://www.cs.yale.edu/homes/aspnes/pinewiki/BTrees.html>