Problem: Estimate the time complexity of finding a key in a B^+-Tree of order *m*. Recall that interior nodes of a B+-Tree of order m have from $\left\lceil \frac{m}{2} \right\rceil$ to *m* children.

Let us change variables.

$d = \left\lceil \frac{m}{2} \right\rceil$ and $2d = m$

Let us say that interior nodes a B^+-Tree of order *d* have from *d* to 2*d* children.

Let *F*(*n*) be the timing function for finding a key in a B^+-Tree of order *d*. Let *n* be the number of unique keys in the B^+-Tree.

The time required to find a key is the product of the time required to search a node, $N(d)$, multiplied by the number of nodes in the path from the root to a leaf $(h+1)$. The number of nodes in the path from the root to a leaf is *h+1* where *h* is the height of the B^+-Tree.

$$F(n) = O(N(d)(h+1))$$

A binary search is used to search a particular node. A node in a particular tree contains at most 2d keys.

$$N(d) = O(\log_2 2d) = O(\log_2 d + 1)$$

Unique keys are contained the leaves of a B^+-Tree. Interior nodes of the B+-Tree having the greatest height have *d* children and *d* keys in the leaves. There are d^l nodes at level *l*. Let n be the number of unique keys in a B^+-Tree of order *d* and height *h*.

$$n = d^h d = d^{h+1}$$

$$h + 1 = \log_d n$$

Thus

$$F(n) = O(\log_d n(\log_2 d + 1))$$
$$F(n) = O(\log_d (n) \log_2 (2d))$$

Problem: Estimate the time complexity of inserting a key in a B^+-tree of order *d*.

In the worst case, a key is inserted into every node on the path to where the new key is inserted. Inserting the new key causes a chain reaction. Every node on the path to the new key is split and a parent hoisted to the next higher level. The length of the path from root to leaf is *h*, the height of the tree. As a result of inserting a new key, the height of the tree could be increased by one. Let *n* be the number of unique keys in the B^+-tree *after* the new key is inserted.

The number of nodes for which a new key can be inserted is one more than the height of the tree.

$$h + 1 = \log_d n$$

The time required to insert a node is $O(2d)$

$$I(n) = O(2d \log_d n)$$

The time required to delete a node from a B^+-tree is the same as the time required to insert a node.

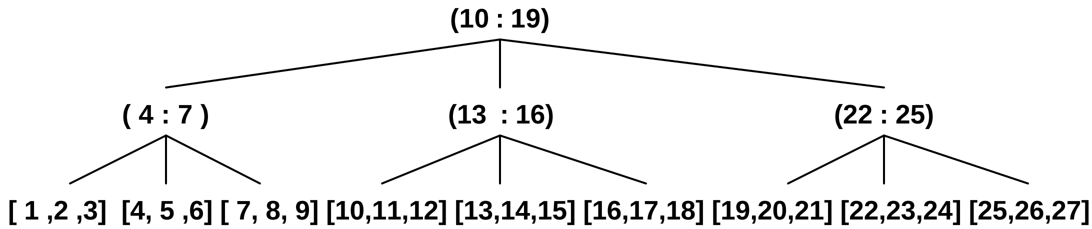

Figure 1. B+Tree, order 3, completely filled

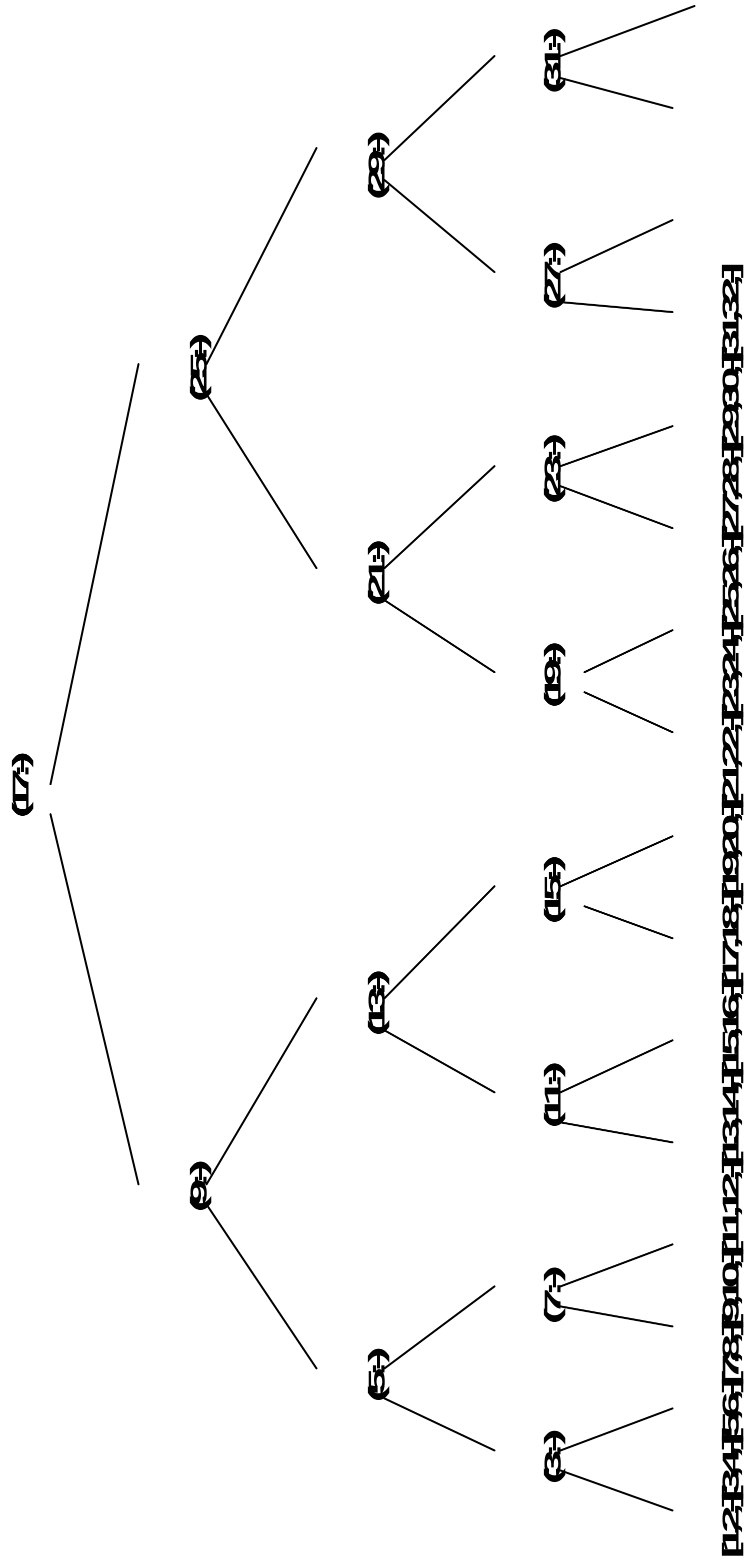

Figure 2. B+Tree, order 3, mimimally filled