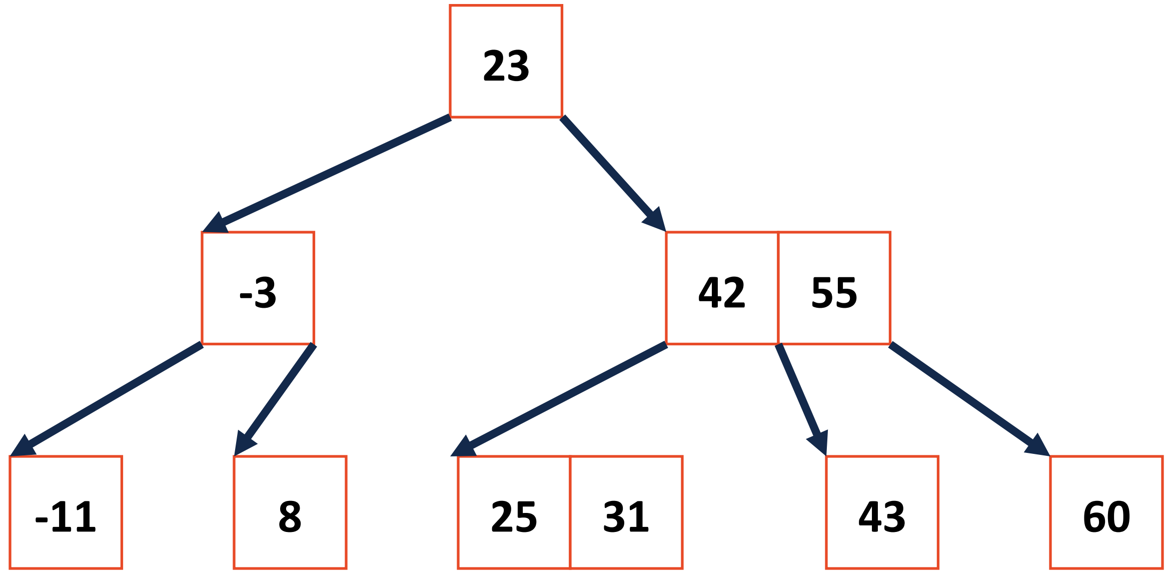
CS 400

B-Tree Search

ID: 08-03

BTree Search



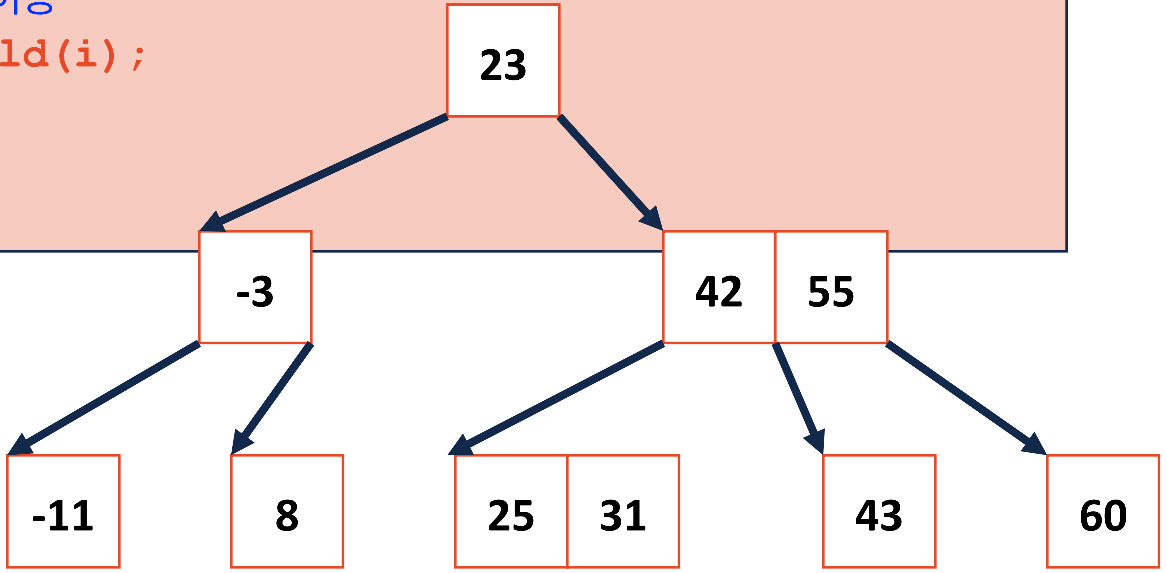
BTree Search

whether or not this key exists in our BTree

```
bool Btree::_exists(BTreeNode & node, const K & key) {
                                          recursive definition
  unsigned i;
  for ( i = 0; i < node.keys_ct_ && key < node.keys_[i]; i++) { }

  if ( i < node.keys_ct_ && key == node.keys_[i] ) {
    return true;                      우리가 찾는 그 키면 return true
  }

  if ( node.isLeaf() ) {
    return false;                 지금 기준이 되는 노드가 leaf이면 끝냄.
  } else {                        아니면 그 다음 child로 이동
    BTreeNode nextChild = node._fetchChild(i);
    return _exists(nextChild, key);
  }
}
```



BTree Analysis

The height of the BTree determines maximum number of ____seeks____ possible in search data.

…and the height of the structure is: ____logm(n)____.

Therefore: The number of seeks is no more than ____logm(n)____.

So, a BTree is an optimized algorithm to optimize around the idea of doing as minimal disk seeks as possible when all of our data is can't fit in main memory.

It's a little bit of a hybrid between a classical tree and a pure disk algorithm. So, we have the best of both worlds by combining two different algorithms we know. So, we'll do a little bit more about BTrees as we look at more algorithms that involve seeking on disks. But largely, we wanted to make sure that you have an idea of exactly if one algorithm that works really well are under conditions that we don't normally account for under Big O notations.