IOrdered

```
package ordered;

public interface IOrdered {
    public static final int LESS    = -1;
    public static final int EQUAL   =  0;
    public static final int GREATER =  1;

    public int compare(IOrdered other);
}
```

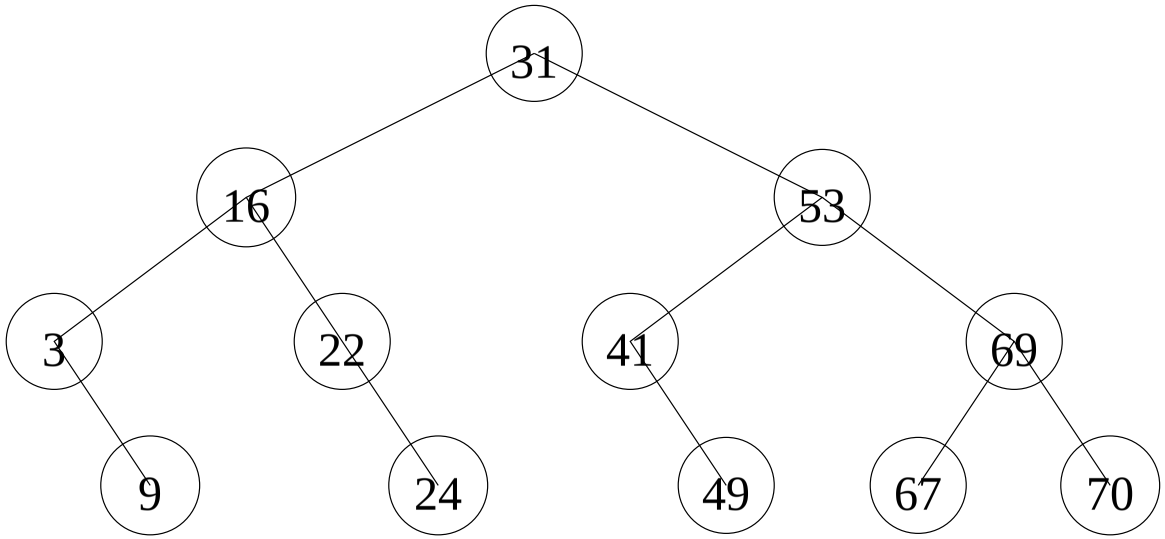
An Ordered Array-based Container

```
private int findIndex(IOrdered key)
{
    int lo = -1;
    int hi = _firstEmptyPair;
    while (lo + 1 != hi) {
        int mid = (lo + hi)/2;
        switch (_pairs[mid].getKey().compare(key)) {
        case IOrdered.EQUAL:   return mid;
        case IOrdered.GREATER: hi = mid;   break;
        case IOrdered.LESS:    lo = mid;   break;
        }
    }
    return lo;
}
```

Calculating the Computational Cost of `findIndex()`

- Consider the following array and its possible traversals by `findIndex`.

3	9	16	22	24	31	41	49	53	67	69	70



- The longest traversal is $\lceil log_2(n+1) \rceil$ where n is the length of the array.

find()

```
public Object find(IOrdered key)
{
    int index = findIndex(key);

    if ((index >= 0) &&
        (_pairs[index].getKey().compare(key) == IOrdered.EQUAL))
        return _pairs[index].getValue();

    return null;
}
```

remove()

```
public Object remove(IOrdered key)
{
    int index = findIndex(key);

    if ((index >= 0) &&
        (_pairs[index].getKey().compare(key) == IOrdered.EQUAL)) {
        Object value = _pairs[index].getValue();

        int i = index;
        for (_firstEmptyPair--; i < _firstEmptyPair; i++)
            _pairs[i] = _pairs[i + 1];
        _pairs[i] = null;

        return value;
    }
    return null;
}
```

insert()

```
public void insert(IOrdered key, Object value)
{
    int index = findIndex(key);

    if ((index >= 0) &&
        (_pairs[index].getKey().compare(key) == IOrdered.EQUAL)) {
        _pairs[index] = new OrderedKeyValuePair(key, value);
        return;
    }
    if (_firstEmptyPair == _pairs.length) {
        OrderedKeyValuePair[] newPairs =
            new OrderedKeyValuePair[2*_pairs.length];

        for (int i = 0; i < _pairs.length; i++)
            newPairs[i] = _pairs[i];

        _pairs = newPairs;
    }
```

`insert()` (cont.)

```
    int i = _firstEmptyPair;

    for (index++; i > index; i--)
        _pairs[i] = _pairs[i - 1];
    _pairs[i] = new OrderedKeyValuePair(key, value);

    _firstEmptyPair++;
}
```

Summary

Operations	Cost
`find()`	O($log\ n$)
`remove()`	O(n)
`insert()`	O(n)

where n is the size of the container