Implementation of BST:

1. Helper method: find

```
Entry find(x):          // Find entry containing x or
                        //   entry where search failed.
    s ← new stack       // to store ancestors of current node
    s.push(null)        // parent of root node
    return find(root, x)

Entry find(t, x):       // search for x in subtree rooted at t
                        // LI: s has path of ancestors from root to t.
    if t = null or t.element = x then return t
    while true do       // LI: t ≠ null
        if x < t.element then
            if t.left = null break
            else {s.push(t); t ← t.left}
        else if x = t.element then break
        else                // x > t.element
            if t.right = null then break
            else {s.push(t); t ← t.right}
    return t
```

RT = O(h)
h = height of tree
= O(depth of node returned by find)

2. contains(x): // does x appear in tree.

```
t ← find(x)
if t = null or t.element ≠ x then
        return false
else return true
```

```
= return t != null and t.element = x
```

---

3. boolean add(x):

```
if root = null then
        root = new Entry(x)
        size ← 1 ; return true
else
        t ← find(x)
        if t.element = x then
                return false // duplicate x is rejected
        else if x < t.element then
                t.left ← new Entry(x)
        else // x > t.element
                t.right ← new Entry(x)

        size++
        return true
```

min() - max() - try at home.

```
remove(x):   // remove and return x.
    t ← find(x)
    if root = null or t.element ≠ x then
        return null.
    result ← t.element   // stack contains ancestors of t
    if t.left = null or t.right = null then
        bypass(t)
    else // t has 2 children; t.element is replaced by its successor
        s.push(t)
        minRight ← find(t.right, x)
        t.element ← minRight.element
        bypass(minRight)
    size--
    return result
```

```
bypass(t):   // Pre: t has at most one child
             // Inv: stack has ancestors of t
    parent ← s.peek()
    child ← t.left == null ? t.right : t.left
    if parent.left = t then
        parent.left ← child
    else parent.right ← child
```