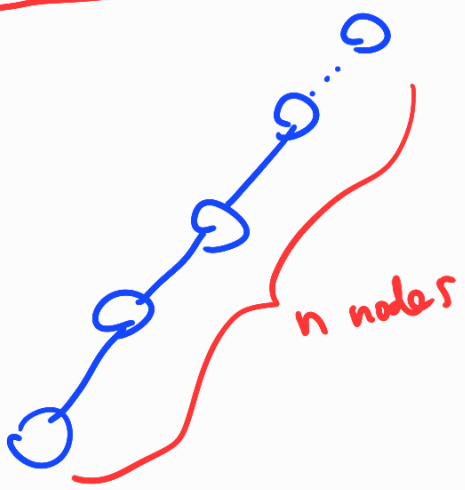
Binary Search Tree (BST)

The running time of the SEARCH operation is $O(h)$ where h is the "height" of the BST.

A BST is <u>not</u> necessarily a nearly complete binary tree.

<u>Worst-Case Scenario</u> [Caveat]



$h = n-1$

$* \quad O(h) = O(n)$

```
1:  procedure BST-SEARCH(root, key)
2:      if root == NULL then
3:          return NULL
4:      else
5:          if key < root.key then
6:              return BST-SEARCH(root.left, key)
7:          else
8:              if key > root.key then
9:                  return BST-SEARCH(root.right, key)
10:             else
11:                 return root
12:
```

Correctness Proof:

Base Case: We need to show that BST-SEARCH works correctly for a BST with height $h = -1$ and $h = 0$.

Case $h = -1$ (Empty BST)

This is trivially true. (Line 1 and 2) ✓

Case $h = 0$ (a BST with one node)

If key is at the root, Line 11 handles this case correctly ✓

If key is not in this BST, either line 6 or line 9 deals with this.

So BST-SEARCH is called with root.left = NULL or root.right = NULL.

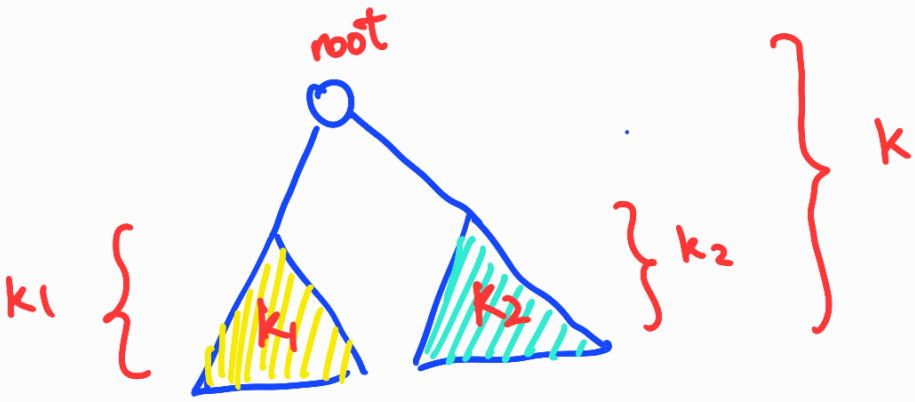
Here, Line 2 will correctly deal with this ✓

Induction Hypothesis:

Assume BST-SEARCH works correctl for a BST with height of $\underbrace{-1, 0, 2, \ldots k-1}_{\text{strong induction.}}$.

Inductive Step:

Let $\max(k_1, k_2) = k-1$.



We will show true for k:

Case I: The root has the key

Line 11 deals with this case cometly.

Case II: The left subtree  contains the key.

Line 6 deals with this case,

Since $k_1 \leq k-1$, we can invoke I.H.

BST-SEARCH works correctly on 

<u>Case III</u>: The right subtree 

contains the key.

Line 9 deals with this case

Since $k_2 \leq k-1$, we can invoke I.H.

BST-SEARCH works correctly on 

Hence, BST-SEARCH works correctly for a BST with height K. ■