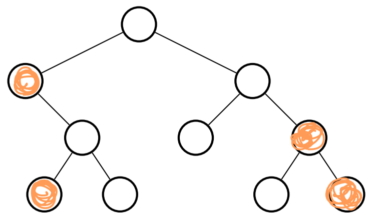
Quiz 4 covers everything up until and including the latest lecture. To prepare for the quiz, you should review your assignment(s) and the lecture notes. For further practice, we're providing some extra problems below. We're also giving you condensed solutions at the end of this handout. You should attempt these problems prior to looking at the solutions.

1 Structure Of A Binary Search Tree

The tree figure below is a binary search tree, though their keys are not shown.



(1) Mark the nodes that are the two largest keys. Indicate which is which.

(2) Mark the nodes that are the two smallest keys. Indicate which is which.

2 Second Key

```
class TreeNode {
  int key;
  TreeNode left;
  TreeNode right;
}
```

Like the tree nodes you have seen, this `TreeNode` class keeps its key in the field `key`, and the left child and right child in `left` and `right`, respectively.

Where's the second smallest key in a binary search tree? You'll answer this question by implementing a function (in Java/pseudo-Java) `int secondKey(TreeNode t)`, where `t` contains at least 2 keys, that returns the second-smallest key in `t`. Write helper functions as necessary. You can assume that there is a `firstKey(t)` function that works perfectly and can refer to it in your code.

3 Unconventional Binary Search

The standard binary search algorithm compares the search key with the middle element to narrow down the range. Someone came up with the following unconventional variant of binary search, which compares the search key with a random element from the range.

```
boolean containsH(long[] A, long key, int lo, int hi) {
    if (lo < hi) {
        int m = draw a random number uniformly from lo, ..., hi - 1
        if (A[m] == key) return true;
        if (A[m] > key)  return containsH(A, key, lo, m);
        else             return containsH(A, key, m+1, hi);
    }
    else return false;
}

boolean contains(long[] A, long k) { return containsH(A, k, 0, A.length); }
```

Our goal is to understand the running time on this variant. We'll do this in several steps:

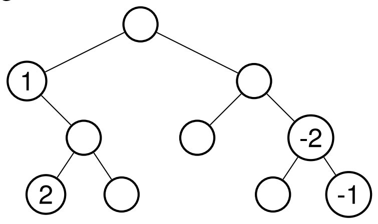
(1) We'll measure the problem size by $n = \texttt{hi} - \texttt{lo}$. For your convenience, define random variables $L = m - \texttt{lo}$ and $R = \texttt{hi} - m - 1$. Write a recurrence that provides an upper bound on the running time of `containsH` in terms of n, L, and R. For example, $T(n) \leqslant \ldots$. Be sure to explain your answer. (*Hint*: The max function.)

(2) Compute $\mathbf{Pr}[\max(L, R) \leqslant 3n/4]$. Don't worry about being slightly off. Show your work.

(3) Using what we know, find the expected running time of `contains`. To save yourself some writing, use $\overline{T}(\cdot)$ to mean $\mathbf{E}[T(\cdot)]$. Show the steps.

4 Condensed Solutions

1. *BST Structure:* The two smallest keys are labeled 1 and 2, and the two largest keys are -2 and -1 (the largest).



2. *Second Key:* Where exactly is the second smallest key in a binary search tree? A moment's thought reveals that it is either **(i)** the parent of the minimum-key node or **(ii)** the minimum key in the right subtree of the minimum-key node. Play with a couple of binary search trees to get a sense of this. Once we know this, it's just a matter of writing a program that walks the tree according to this recipe. Remember that the smallest key can be found by walking left in the tree until it is no longer possible.
3. *Unconventional Binary Search:* This is essentially the same derivation that we did in class (see also the lecture notes). Notice that the recurrence becomes $T(n) = T(\max(L, R)) + O(1)$. The probability that $\mathbf{Pr}[\max(L, R) \leqslant 3n/4]$ is $1/2$ because the array is sorted, so the event happens as long as the chosen m is between the $n/4$-th and $3n/4$-th index. Hence, we have $\bar{T}(n) \leqslant \frac{1}{2}(c + \bar{T}(3n/4)) + \frac{1}{2}(c + \bar{T}(n))$, where c is a constant. This recurrence, by the way, solves to $O(\log n)$. Hence, it has a $O(\log n)$ running time in expectation.