For each of the following problems, describe the algorithm, give students about 10 minutes to think about it, and then provide the solution. Make sure they get the algorithms since the next tutorial will again be about them.

1. Let $T(n)$ denote the worst-case running time of the algorithm below on inputs of size n.

```
# **Precondition**: $A$ is a non-empty list of integers, $i$ is a natural number.
def recSS($A, i$):
    if $i < \text{len}(A) - 1$:
        $small = i$
        for $j$ in range($i+1, \text{len}(A)$):
            if $A[j] < A[small]$:
                $small = j$
        $temp = A[i]$
        $A[i] = A[small]$
        $A[small] = temp$
        recSS($A, i+1$)
```

Note that the above algorithm has an implicit base case, for which it does nothing.

Write a recurrence relation satisfied by T. Make sure to define n precisely (as a function of the algorithm's parameters) and justify that your recurrence is correct (by referring to the algorithm to describe how you obtained each term in your answer).

Solution Elements

For any natural number n, let $T(n)$ denote the maximum number of steps executed by a call to recSS(A, i), where $n = \text{len}(A) - i$.

If $n = 1$, then it does nothing except evaluating the if-condition, which take constant time, represented by a constant value a (note that $n \geqslant 1$ as in all recursive calls i is less than or equal to $\text{len}(A) - 1$).
Otherwise, lines 2 to 9 execute. The for-loop in line 3 executes $n - 1$ times. Therefore, lines 3–5 take $b * (n-1)$, where b is a constant value. The recursive call in line 9 increments i by one, and therefore decrements $n = len(A) - i$ by one. Therefore, line 9 takes $T(n-1)$.
Lines 6 – 8, and all other instructions in lines 3–9 take constant time, represented by a constant value c. Putting all together, we get the following definition for $T(n)$:

$$T(n) = \begin{cases} O(1), & n \leq 1 \\ T(n-1) + O(n), & n \geqslant 2 \end{cases}$$

2. The following algorithm performs *breadth-first search* on a *perfect binary search trees*. A *binary search tree* is a binary tree which stores a value $u.val$ for every node u. The values satisfy: for all nodes s in the subtree of u rooted at its left-child, $s.val < u.val$, and for all nodes t in the subtree of u rooted at its right-child. $t.val > u.val$. *You can assume that the values are all unique.* Further, such a tree is *perfect* if each internal node has both a left *and* a right child and all leaves are of the same height (with the maximum number of leaves possible). *We want to know if x appears as a value on some node of the tree.*

Draw an example of a perfect binary search tree on the board.

Let $T(n)$ denote the worst-case running time of the algorithm on trees with n nodes.

```
   # **Precondition**: $r$ is the root of a perfect balanced binary search tree.
   **def** BFS($r, x$):
1.     **if** $r$ is None:
2.         **return** False
3.     **if** $r.val == x$:
4.         **return** True
5.     **if** $x < r.val$:
6.         **return** BFS($r.left, x$)
7.     **else**:
8.         **return** BFS($r.rigth, x$)
```

Write a recurrence relation satisfied by T. Make sure to define n precisely (as a function of the algorithm's parameters) and justify that your recurrence is correct (by referring to the algorithm to describe how you obtained each term in your answer).

Solution Elements

For any natural number n, let $T(n)$ denote the maximum number of steps executed by BFS(r, x), where r is the root of a perfect binary search tree with n nodes.

If the tree is empty, then x is not in the tree and we return false. If the value stored at the root is *exactly* the value that we are looking for, we return true. Otherwise one of the recursive lines 6 or 8 will execute (but not both). Note that the subtree rooted at each of r's children will have about half of the number of nodes that the tree has since the tree is perfect.

Putting all together, we get the following definition for $T(n)$:

$$T(n) = \begin{cases} O(1), & n = 0 \\ T(n/2) + O(1), & n \geqslant 1 \end{cases}.$$