```
CS 519-005, Algorithms (MS/MEng-level), Winter 2018
HW1 - Python 3, qsort, BST, and qselect
Due electronically on flip on Monday Jan 15, 11:59pm.
No late submission will be accepted.

Need to submit on flip: report.txt, qsort.py, and qselect.py.
qselect.py will be automatically graded for correctness (1%).

flip $ /nfs/farm/classes/eecs/winter2018/cs519-010/submit hw1 qselect.py qsort.py
report.txt

Note:

1. You can ssh to flip machines from your own machine by:
   $ ssh access.engr.oregonstate.edu

2. You can add /nfs/farm/classes/eecs/winter2018/cs519-010/ to your $PATH:
   $ export PATH=$PATH:/nfs/farm/classes/eecs/winter2018/cs519-010/
   so that you don't need to type it every time.

3. You can choose to submit each file separately, or submit them together.

Textbooks for References:
[1] CLRS Ch. 9.2 and Ch. 12

0. Q: What's the best-case, worst-case, and average-case time complexities of quicksort.
   Briefly explain each case.

1. [WILL BE GRADED]
   Quickselect with Randomized Pivot (CLRS Ch. 9.2).

   >>> from qselect import *
   >>> qselect(2, [3, 10, 4, 7, 19])
   4
   >>> qselect(4, [11, 2, 8, 3])
   11

   Q: What's the best-case, worst-case, and average-case time complexities? Briefly
explain.

   Filename: qselect.py


2. Buggy Qsort Revisited

   In the slides we showed a buggy version of qsort which is weird in an interesting way:
   it actually returns a binary search tree for the given array, rooted at the pivot:

   >>> from qsort import *
   >>> tree = sort([4,2,6,3,5,7,1,9])
   >>> tree
   [[[[], 1, []], 2, [[], 3, []]], 4, [[[], 5, []], 6, [[], 7, [[], 9, []]]]]

   which encodes a binary search tree:

                   4
                 /   \
               2       6
              / \     / \
             1   3   5   7
                          \
                           9

   Now on top of that piece of code, add three functions:
   * sorted(t): returns the sorted order (infix traversal)
   * search(t, x): returns whether x is in t
   * insert(t, x): inserts x into t (in-place) if it is missing, otherwise does nothing.

   >>> sorted(tree)
```

```
    [1, 2, 3, 4, 5, 6, 7, 9]
    >>> search(tree, 6)
    True
    >>> search(tree, 6.5)
    False
    >>> insert(tree, 6.5)
    >>> tree
    [[[[], 1, []], 2, [[], 3, []]], 4, [[[], 5, []], 6, [[[], 6.5, []], 7, [[], 9, []]]]]
    >>> insert(tree, 3)
    >>> tree
    [[[[], 1, []], 2, [[], 3, []]], 4, [[[], 5, []], 6, [[[], 6.5, []], 7, [[], 9, []]]]]

    Hint: both search and insert should depend on a helper function _search(tree, x) which
    returns the subtree (a list) rooted at x when x is found, or the [] where x should
    be inserted.

    e.g.,
    >>> tree = sort([4,2,6,3,5,7,1,9])        # starting from the initial tree
    >>> _search(tree, 3)
    [[], 3, []]
    >>> _search(tree, 0)
    []
    >>> _search(tree, 6.5)
    []
    >>> _search(tree, 0) is _search(tree, 6.5)
    False
    >>> _search(tree, 0) == _search(tree, 6.5)
    True

    Note the last two []'s are different nodes (with different memory addresses):
    the first one is the left child of 1, while the second one is the left child of 7
    (so that insert is very easy).

    Filename: qsort.py

    Q: What are the time complexities for the operations implemented?

 Debriefing (required!): --------------------------

 1. Approximately how many hours did you spend on this assignment?
 2. Would you rate it as easy, moderate, or difficult?
 3. Did you work on it mostly alone, or mostly with other people?
 4. How deeply do you feel you understand the material it covers (0%–100%)?
 5. Any other comments?

 This section is intended to help us calibrate the homework assignments.
 Your answers to this section will *not* affect your grade; however, skipping it
 will certainly do.
```