UNIVERSITY OF TORONTO
Faculty of Arts & Science

April 2022 MOCK EXAM

CSC 148 H1S

Duration: 45 mins (The real one is 3 hours)

Aids Allowed: Provided aid sheet

*Do **not** turn this page*
until you have received the signal to start.
In the meantime,
please fill out the section below,
and carefully read all instructions on this page.

Given Name(s):

Family Name(s):

Student Number (9 or 10 digits):

UTORid (e.g., `pitfra12`):

- Turn off and place all cell phones, smart watches, electronic devices, and unauthorized study materials in your bag under your desk. If it is left in your pocket, it may be an academic offence.
- This MOCK examination consists of 4 questions on 6 pages (including this one), printed on both sides of the paper. When you receive the signal to start, please make sure that your copy of the examination is complete.
- Answer each question directly on the examination paper, in the space provided. There are several blank pages at the end for rough work.
- Comments are not required in your code, however they may help us give you part marks if your answer is not completely correct.
- There is an aid sheet on a separate piece of paper. **Nothing you write on the aid sheet will be marked.**
- Remember that, in order to pass the course, you must achieve a grade of at least **40%** on the real final examination. (This mock one is just for practice, and not graded! :))
- As a student, you help create a fair and inclusive writing environment. If you possess an unauthorized aid during an exam, you may be charged with an academic offence.

Marking Guide

Nº 1: ______/ 6

Nº 2: ______/ 4

Nº 3: ______/ 4

Nº 4: ______/ 6

TOTAL: ______/20

It's been a real pleasure teaching you this term. We want you to do well on the exam and show us all that you learned! :)

1. [**6 marks**] **Exceptions** Consider the following function:

```
def solve(x, y):
    try:
        if x > 50:
            raise ValueError
        a = x / y
        solution = 2 * a
        print(solution)
    except ZeroDivisionError:
        print('Cannot divide by zero')
    except NameError: # this error is raised when a variable that has not been defined is used
        print('Name not defined inside')
    except:
        print('Something is wrong')
    finally:
        print('End of program')
```

What would be the output produced by each of the segments of code below?

```
>>> solve(20, 2)
```

```
>>> solve(100, 50)
```

```
>>> solve('science', 1)
```

TRY AT HOME!

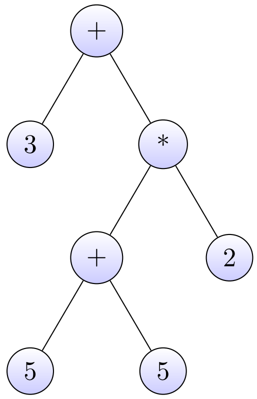
```
>>> try:
...     solve(x, 1)
... except NameError:
...     print('Name not defined outside')
```

```
>>> try:
...     print(solve(5, 1) * 2)
... except:
...     print('Something is wrong outside')
```

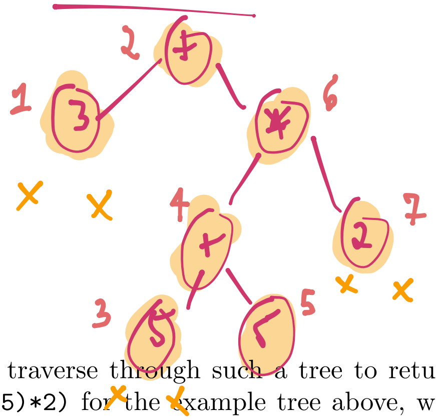
2. [**4 marks**] **Tree Traversals**

For this question, we will be dealing with binary trees in which each internal node corresponds to a simple mathematical operator (one of +, -, * or /) and each leaf node corresponds to an integer operand.

For example, the tree representation for `3 + ((5+5)*2)` would be:



IN-ORDER → 3 +((5 + 5) * 2)

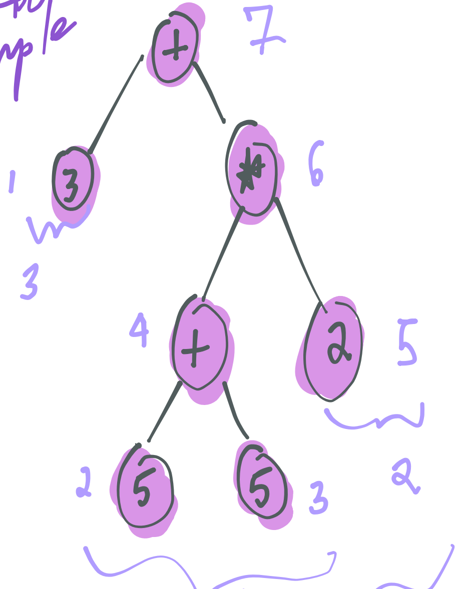


(a) [**2 marks**] If I wanted to traverse through such a tree to return the expression it represents in the form of a string such as `3 + ((5+5)*2)` for the example tree above, which of the following would be most suitable? Explain your answer. **No explanation = no marks**

i. In-order traversal → L, root, right
ii. Pre-order traversal → Root, children (L ... R)
iii. Post-order traversal → children (L ... R), Root
iv. Level order traversal → level by level (by depth)

(b) [**2 marks**] If I wanted to traverse through such a tree to evaluate the expression and return the solution, such as for the example tree above, we would return the integer 23, then which of the above traversal algorithms would be most suitable? Explain your answer. **No explanation = no marks**

POSTORDER example



Example: Solve 3+((5+5)*2)

① solve L
② solve R
③ Then, apply the root operator

3. [**4 marks**] Complete the following `BinarySearchTree` method according to the docstring (each docstring example includes an associated image beside it of the tree with all items that need to be removed highlighted in blue).

Hint 1: Don't forget to ensure that all of your representation invariants are satisfied before the method returns.

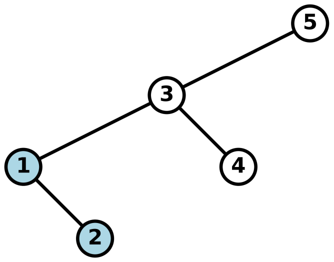
Hint 2: You should find the "promotion" strategy from when we covered BST deletion to be helpful.

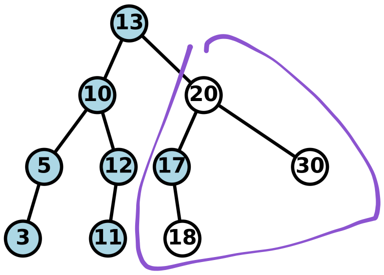
Note: See Aid sheet for BST documentation.

```
def prune(self, low: int) -> None:
    """
    Remove all values from this BST that are less than <low>

    Precondition: all values in this BST are integers

    >>> root = BinarySearchTree(None)
    >>> for x in [5, 3, 4, 1, 2]:
    ...     root.insert(x)
    >>> root.prune(3)
    >>> assert root.items() == [3, 4, 5]
    >>> me = BinarySearchTree(None)
    >>> for x in [13, 20, 17, 30, 18, 10, 5, 12, 11, 3]:
    ...     me.insert(x)
    >>> me.prune(18)
    >>> assert me.items() == [18, 20, 30]
    """
```





```
if self.is_empty():
    return    #do nothing
elif self._root < low:
    self._right.prune(low)
    self._root = self._right.root
    self._left = self._right._left
    self._right = self._right._right
else:
    self._left.prune(low)
```

→ what if this empties my tree?

Think about if this works when ALL of self._right's values are less than low

4. **[6 marks] Stacks / Recursion**

   (a) [**4 marks**] Complete the following function which takes an item and inserts it at the bottom of a stack.
   Note: See Aid sheet for Stack API.
   RESTRICTIONS (violating any of these points will result in a grade of 0):

   i. Your function must be **recursive**.

   ii. You may not use any loops.

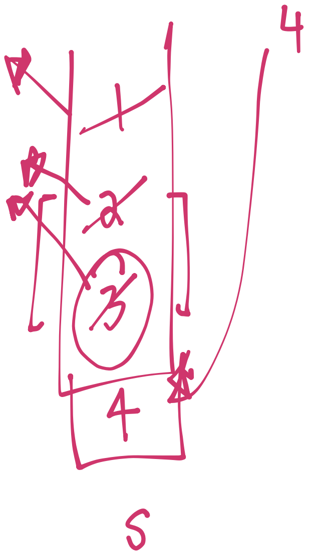
   iii. The Stack class only has the methods is_empty, push and pop.

   iv. You may not create any new objects (no new stacks, lists, etc., including temporary ones).

```
def insert_at_bottom(s: Stack, item: Any) -> None:
    """Insert the given <item> at the bottom of the given Stack <s>."""

    if s.is_empty():
        s.push(item)

    else:
        temp = s.pop()
        insert_at_bottom(s, item)
        s.push(temp)
```



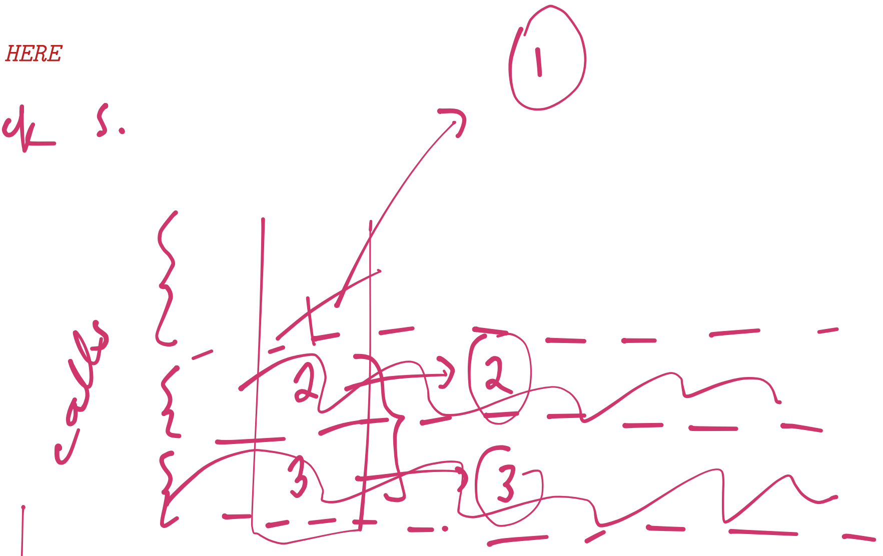
   (b) [**2 marks**] Complete the type contract by filling in the return type, and then write a proper docstring description of the function below (which uses `insert_at_bottom` from the previous question as a helper).
   Your description must be clear and concise, and follow proper docstring style (do **not** give a line-by-line description of the code).

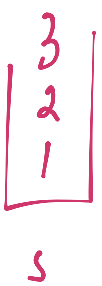
```
def mystery(s: Stack) -> None: # FILL IN RETURN TYPE HERE
    """
    # YOUR DOCSTRING DESCRIPTION HERE

    Reverse the stack s.

    """

    if not s.is_empty():
        temp = s.pop()
        mystery(s)
        insert_at_bottom(s, temp)
```





Basic operators

```
True and False, True or False, not True
1 + 3, 1 - 3, 1 * 3
5 / 2 == 2.5, 5 // 2 == 2, 5 % 2 == 1
'hi' + 'bye'           # 'hibye'
[1, 2, 3] + [4, 5, 6]  # [1, 2, 3, 4, 5, 6]
```

Stacks and Queues

```
s = Stack()
s.is_empty()
s.push(10)
s.pop()  # Raises an EmptyStackError if stack is empty.

q = Queue()
q.is_empty()
q.enqueue(10)
q.dequeue()  # Returns None if queue is empty.
```

Binary Search Trees

```
class BinarySearchTree:
    # === Private Attributes ===
    # _root: The item stored at the root of the tree, or None
    #        if the tree is empty.
    _root: Optional[Any]
    # _left: The left subtree, or None if the tree is empty.
    _left: Optional[BinarySearchTree]
    # _right: The right subtree, or None if the tree is empty.
    _right: Optional[BinarySearchTree]

    # === Representation Invariants ===
    # - If self._root is None, then so are self._left and
    #   self._right. This represents an empty BST.
    # - If self._root is not None, then self._left and
    #   self._right are BinarySearchTrees.
    # - (BST Property) If self is not empty, then
    #   all items in self._left are <= self._root, and
    #   all items in self._right are >= self._root.

    def __init__(self, root: Optional[Any],
           left: Optional[BinarySearchTree] = None,
           right: Optional[BinarySearchTree] = None) -> None:
        """Initialize this BST with root, left and right. If <root> is None, initialize an empty tree.
        """

    def is_empty(self) -> bool:
        """Return whether this BST is empty."""

    def items(self) -> list:
        """Return all the values in this BST in sorted order."""

    def insert(self, item: Any) -> None:
        """Insert <item> into this tree.
        Do not change positions of any other values.
        """
```