Problem Set 2

This problem set is due **Wednesday, March 7** at **11:59PM**.

Solutions should be turned in through the course website. **You must enter your solutions by modifying the solution template (in Python) which is also available on the course website.** The grading for this problem set will be largely automated, so it is important that you follow the specific directions for answering each question.

For multiple-choice and true/false questions, no explanations are necessary: your grade will be based only on the correctness of your answer. For all other non-programming questions, full credit will be given only to correct solutions which are described clearly and concisely.

Programming questions will be graded on a collection of test cases. Your grade will be based on the number of test cases for which your algorithm outputs a correct answer within time and space bounds which we will impose for the case. Please do not attempt to trick the grading software or otherwise circumvent the assigned task.

---

1. **Del or no del? (35 points, 5 points per part)**

   Consider the following correct Python implementation for deleting a node from a binary search tree. This function is analogous to the `delete` method of the `BSTNode` class found on the website, except that it assumes that all `BSTNode` instances have a parent pointer. (Since the implementation on the website does not include parent pointers, you will not be able to test this code by replacing the `delete` method in that class.)

   This `delete` function takes a node, `self`, and a value, `val`. It deletes the node with that value from the subtree rooted at `self`, if it exists and if the tree has at least one other node. The function returns `True` if some node was deleted.

   Assume each node has five properties: its `val`, `count`, `left`, `right`, and `parent`. The `left`, `right`, and `parent` pointers are either other instances of this class or `None`. Also, assume that the `search` method is implemented exactly as in the `BSTNode` class.

```
 1 def delete(self, val):
 2     # Find the node to delete.
 3     node = self.search(val)
 4     if node.val != val:
 5         return False
 6
 7     # If there were multiple occurrences of this value, we're done.
 8     node.count -= 1
 9     if node.count > 0:
10         return True
11
12     if node.right is None:
13         if node.left is None:
14             # This node is a leaf. Delete its reference from its parent.
15             if node.parent is not None:
16                 if node.parent.left == node:
17                     node.parent.left = None
18                 else:
19                     node.parent.right = None
20                 return True
21             else:
22                 # We are the only node. Deletion is not allowed.
23                 return False
24         else:
25             # Move the old left child to our place.
26             node.val = node.left.val
27             node.count = node.left.count
28             node.right = node.left.right
29             node.right.parent = node
30             node.left = node.left.left
31             node.left.parent = node
32             return True
33     else:
34         # We have a right child. Replace this node with its successor
35         # in the right subtree.
36         next = node.right.search(val)
37         if next is not None:
38             node.val = next.val
39             node.count = next.count
40             next.count = 1
41         node.right.delete(next.val)
42         return True
```

Answer the following questions with `True` or `False`.

a. The code is correct if lines 25 – 32 are replaced with the lines

```
25              # Move the old left child to our place.
26              node.left.parent = node.parent
27              if node.parent is not None:
28                  if node.parent.left == node:
29                      node.parent.left = node.left
30                  else:
31                      node.parent.right = node.left
32              return True
```

b. The code is correct if line 36 is replaced with

```
36          next = node.right
```

c. The code is correct if line 37 is removed (and lines 38 – 40 are unindented).

d. The code is correct if all instances of `left` and `right` are interchanged.

e. Lines 34 – 42 can be replaced with the lines

```
34          # We have a right child. Replace this node with its successor
35          # in the right subtree.
36          next = node.right.search(val)
37          if next.right is not None:
38              next.right.parent = next.parent
39          if next.parent.left == next:
40              next.parent.left = next.right
41          else:
42              next.parent.right = next.right
43          return True
```

f. The code is correct if line 41 is moved to just before line 37.

g. The code is correct if lines 41 and 42 are combined into

```
40          return node.right.delete(next.val)
```

Solution Format:

You should answer this problem with a boolean value for each part. For example, if you thought the answer to part y) was `True` and the answer to part z) was `False`, then your answer should be:

```
answer_for_problem_1_part_y = True
answer_for_problem_1_part_z = False
```

2. **Binary search tree, of sorts (20 points)**

   Consider the following code for a sorting algorithm. Here, the `BST` class is an implementation of a self-balancing binary search tree. This class supports the `insert`, `get_min`, and `delete` operations in $O(\log n)$ time, where n is the number of elements in the tree.

```
def bst_sort(list):
    bst = BST()
    for val in list:
        bst.insert(val)
    ans = []
    for i in range(len(list)):
        min = bst.get_min()
        ans.append(min)
        bst.delete(min)
    return ans
```

   a. **(5 points)** This function sorts the list: `True` or `False`?

   b. **(5 points)** On a list of n elements, the runtime of this algorithm is:

      1. $O(n)$
      2. $O(n \log n)$
      3. $O(n \log^2 n)$
      4. $O(n^2)$
      5. $O(n^2 \log n)$
      6. $O(n^2 \log^2 n)$

   c. **(10 points)** Assuming that (comparison) sort is impossible in better than $\Theta(n \log n)$, give a short argument that it is impossible to construct a data structure which stores arbitrary ordered values and supports `insert`, `get_min` and `delete`, each in $o(\log n)$.

   Solution Format:

   Your answer for part a) should be a boolean. Your answer for part b) should be an integer between 1 and 6, and your answer for part c) should be a (short) string.

3. **An awkward sort of party (20 points)**

   There are n people who attend a party, labeled 1 through n. Person i arrives at time a_i and departs at time d_i. The $2n$ arrival / departure times are all distinct.

   None of the partygoers knew each other before the event. Afterwards, each person goes on Twitter and follows the people who were there when they arrived at the party, but who left before they did.

   Find an efficient algorithm to determine the total number of new Twitter followings formed, given the the n pairs of the arrival and departure times of each person. Prove that your algorithm is correct and find its running time. For full credit, your algorithm should run in $O(n \log n)$ time.

   Solution Format:

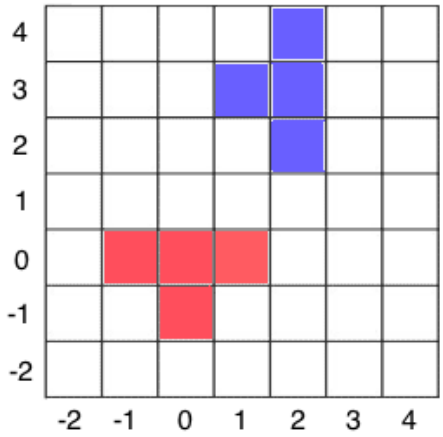
   Your answer for this problem should be a string, such as:

```
answer_to_problem_3 = """
I have a beautiful algorithm for this problem, but this tweet is not
long enough to contain it.
"""
```

4. **Polyomino time algorithms (50 points)**

Two biologists have independently documented the proteins found in two strains of bacteria, *E. foo* and *E. bar*. Each protein is a *polyomino*: a two-dimensional shape formed by attaching a number of unit squares along their edges. Two proteins are the same if one polyomino can be transformed into the other by a rotation and translation.

Each scientist represents a protein as a list of ordered pairs of integers, one pair for each unit square in the protein. Each pair represents the coordinates of the center of its unit square. For example, the T-protein (which looks much like the T piece in Tetris) might be represented by the list `[(0,0),(1,0),(-1,0),(0,-1)]` or by the list `[(2,3),(2,4),(2,2),(1,3)]`:



Two representations of the T-protein.

As a computer scientist working with the biology department, your job is to determine the number of proteins in common between the two strains of bacteria. Write a function `num_proteins_in_common` that efficiently computes the number of proteins in common, given two lists of proteins. You may assume that the proteins in each list are distinct. However, you may not assume a bound on the number of proteins or on the number of squares in a protein.

We have attached some code to help you get started with this problem. Specifically, we have provided three functions for your use:

- a `translate` function that translates a polyomino by a fixed offset
- a `rotate` function that rotates a polyomino by a quarter-turn counterclockwise
- and a `compare` function that determines if two polyominoes are equivalent after rotations and translations

(To see more examples of polyominoes, you may want to visit `ntris.mit.edu`. However, a high score will not get you any credit for this class.)