Data Structures
CSCI C343, Fall 2014

Midterm

Name: ______________________

This exam has 10 questions, for a total of 100 points.

1. 9 points What is the output of the following Python program?

```
import math

def abs(n):
    return n
class Point1D:
    def __init__(self, x):
        self.x = x
    def distance(self, other):
        return abs(self.x - other.x)
    def abs(self, n):
        return n if n >= 0 else -n
def transform(L, f):
  return [f(x) for x in L]

p1 = Point1D(3)
p2 = Point1D(5)
print(p1.distance(p2))
L = [Point1D(3), Point1D(5)]
print(transform(L, lambda p: p.x * p.x))
D = {3: 'x', 'x': 3}
print(D['x'], D[3])
```

Solution: (3 points each)

```
-2
[9, 25]
(3, 'x')
```

2. 10 points Define a function in Python named `swap_range` with four parameters. The function should swap the elements of two sub-ranges within an array given by parameter `A`. The beginning of the first subrange is given by parameter `begin1` and the beginning of the second subrange is given by parameter `begin2`. The two sub-ranges have the same length, given by paramter `n`. You may assume that `begin1 + n <= begin2`.

Example use of `swap_range`:

```
A = [1,2,3,4]
swap_range(A, 0, 2, 2)
assert A == [3,4,1,2]
```

Solution:

```
def swap(A, i, j):
    tmp = A[i]
```

```
        A[i] = A[j]
        A[j] = tmp

    def swap_range(A, begin1, begin2, n):
        for i in range(0, n):
            swap(A, begin1 + i, begin2 + i)
```

4 points for correct logic for swapping
4 points for looping over the array correctly
2 points for swapping the right elements

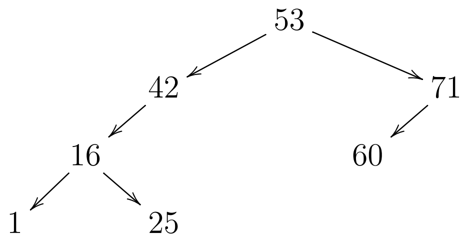
3. $\boxed{\text{15 points}}$ The following code defines a linked-list data structure comprised of a `List` and `Node` class. Fill in the blanks of the implementation so that the unit test passes.

```
class Node:
    def __init__(self, d, n = None):
        self.data = d
        self.next = n

class List:
    def __init__(___(a)___, array = []):
        self.head = None
        for x in reversed(array):
            self.head = ___(b)___

    def to_array(self):
        n = ___(c)___
        A = []
        while n:
            A.append(___(d)___)
            n = ___(e)___
        return A

if __name__ == "__main__":
    L = List([1,2,3])
    assert L.to_array() == [1,2,3]
```

Solution: (3 points each)

```
(a) self
(b) Node(x, self.head)
(c) self.head
(d) n.data
(e) n.next
```

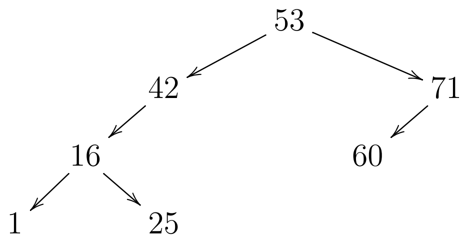
4. $\boxed{\text{8 points}}$ For each of the nodes 25, 42, 53, and 71, list the successor and predecessors of the node (if they exist).



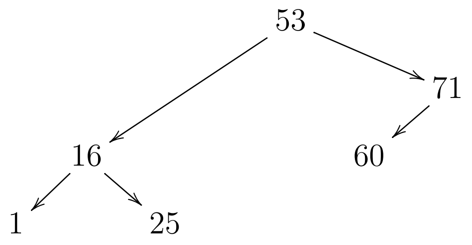
Solution: (2 points each)

- For 25, 42 is the successor and 16 is the predecessor.
- For 42, 53 is the successor and 25 is the predecessor.
- For 53, 60 is the successor and 42 is the predecessor.
- For 71, there is no successor and 60 is the predecessor.

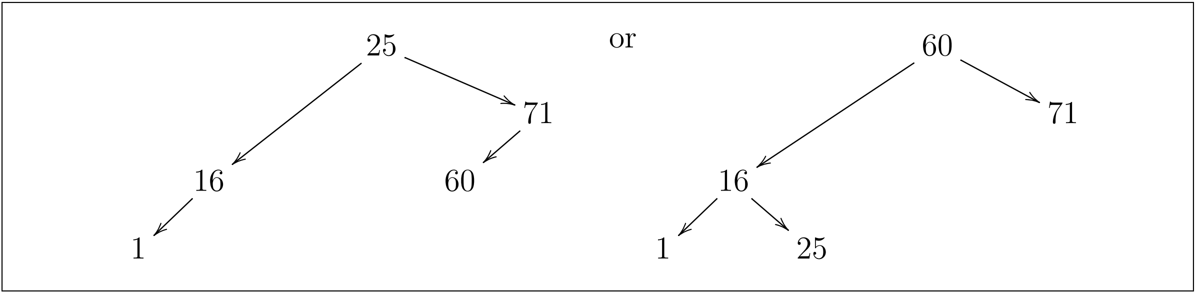
5. $\boxed{\text{10 points}}$ Draw the result of deleting the node with key 42, followed by deleting the node with key 53, from the below Binary Search Tree.



Solution: The node with key 42 has only one child, so we can attach its child to its parent. (5 points)



The node with key 53 has two children, so we can replace with with either its predeccessor 25 or its successor 60. (5 points) Both solutions are shown below.



6. 10 points Implement the `in_order` methods of the `Tree` and `Node` classes. They should return a list of the keys in the tree, ordered by the "in-order" traversal. For example:

```
x = Node(8,
         Node(3, Node(1), Node(6, Node(4), Node(7))),
         Node(10, None, Node(14, Node(13))))
T = Tree(x)
assert T.in_order() == [1,3,4,6,7,8,10,13,14]

class Tree:
    def __init__(self, r):
        self.root = r

    def in_order(self):



class Node:
    def __init__(self, k, l=None, r=None):
        self.key = k
        self.left = l
        self.right = r

    def in_order(self):
```

Solution:

```
class Tree:
    def __init__(self, r):
        self.root = r

    def in_order(self):
        return self.root.in_order()

class Node:
    def __init__(self, k, l=None, r=None):
        self.key = k
        self.left = l
        self.right = r

    def in_order(self):
        L = self.left.in_order() if self.left else []
        R = self.right.in_order() if self.right else []
        return L + [self.key] + R
```

7. 10 points What is a best DNA sequence aligment for the sequences TAG and ATG? When computing the score, use +2 for a match, -2 for a mismatch, and -1 for the gap penalty. Show your work by filling in the below dynamic programming table.

		T	A	G
A				
T				
G				

Solution: One of several solutions is

```
TA_G
_ATG
```

		T	A	G
	0	←-1	←-2	←-3
A	↑-1	↖ -2	↖ 1	←0
T	↑-2	↖ 1	↑0	↖-1
G	↑-3	↑ 0	←-1	↖ 2

(2 points for correct initialization of the table. 5 points for the rest of the table. 3 points for correct result from traceback.)

8. 12 points Suppose that `L` is a Python `list` (array) of length n. Categorize the worst-case execution time of the below expressions as either

1. constant time (takes the same amount of time no matter what n is).
2. logarithmic time (takes time proportional to $\lg n$).
3. linear time (takes time proportional to n)
4. quadratic (takes time proportional to n^2)

Label each operation with the above item number.

- `n in L`
- `L.remove(0)`
- `L[n/2]`
- `L.pop()`

Solution: (3 points each)

- (3), `n in L` is linear time,
- (3), `L.remove(0)` is linear time,
- (1), `L[n/2]` is constant time,
- (1), `L.pop()` is constant time.

9. $\boxed{\text{6 points}}$ What is the big-O worst-case time complexity of the following function, which raises `x` to the power `n`, in terms of the input `n`? Assume that all the primitive number operations (multiplication, division, etc.) are constant time. Explain why your answer is correct.

```
def exponent(x, n):
    if n == 0:
        return 1
    elif n == 1:
        return x
    elif n % 2 == 0:
        return exponent(x * x, n / 2)
    else:
        return x * exponent(x, n - 1)
```

Solution: In the worst-case, each time we divide in half we get an odd number, and have to make an extra call to exponent. So we have $2 \lg n$ calls, but the factor of 2 is ignored for the big-O, so the time complexity is $O(\lg n)$.

10. $\boxed{\text{10 points}}$ Give the definition of Θ. Prove that if $f(n) \in \Theta(g(n))$ then $g(n) \in \Theta(f(n))$.

Solution: The definition of Θ is: (2 points)

$$\Theta(g(n)) = \{f(n) \mid \exists n_0. \forall n \geq n_0. \exists c_1 c_2.\ 0 \leq c_1 g(n) \leq f(n) \leq c_2\, g(n)\}$$

So we can assume

$$\forall n \geq n_0.\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

and we need to show that

$$\forall n \geq n'_0.\ 0 \leq c'_1 f(n) \leq g(n) \leq c'_2 f(n)$$

for some suitable choice for n'_0, c'_1, c'_2. From $f(n) \leq c_2 g(n)$ we have

$$\forall n \geq n_0.\ \frac{1}{c_2} f(n) \leq g(n) \qquad \text{(2 points)}$$

and from $c_1 g(n) \leq f(n)$ we have

$$\forall n \geq n_0.\ g(n) \leq \frac{1}{c_2} f(n) \qquad \text{(2 points)}$$

Also, from $0 \leq f(n)$ we have

$$\forall n \geq n_0.\ 0 \leq \frac{1}{c_2} f(n) \qquad \text{(1 point)}$$

Putting these together we have

$$\forall n \geq n_0.\ 0 \leq \frac{1}{c_2} f(n) \leq g(n) \leq \frac{1}{c_1} f(n)$$

So we choose $n'_0 = n_0$ (1 point), $c'_1 = \frac{1}{c_2}$ (1 point), and $c'_2 = \frac{1}{c_1}$ (1 point).