1 Trees

Things to remember:

```python
def tree(label, branches=[]):
    return [label] + list(branches)

def label(tree):
    return tree[0]

def branches(tree):
    return tree[1:]
```

1. Draw the tree that is created by the following statement:
   ```python
tree(4,
       [tree(5, []),
        tree(2,
            [tree(2, []),
             tree(1, [])],
        tree(1, []),
        tree(8,
            [tree(4, [])]))]
```
2. Construct the following tree and save it to the variable \( t \).

\[
\text{Solution:} \quad t = \text{tree}(9, [\text{tree}(2, []), \text{tree}(4, [\text{tree}(1, []), \text{tree}(7, []), \text{tree}(3, []))])}
\]
3. What would this output?

```python
>>> label(t)

Solution: 9
```

```python
>>> branches(t)[2]

Solution:
tree(4, [tree(7, []), tree(3, [])])
```

```python
>>> branches(branches(t)[2])[0]

Solution:
tree(7, [])
```

4. Write the Python expression to return the integer 2 from t.

```python
Solution:
label(branches(t)[0])
```

5. Write the function `sum_of_nodes` which takes in a tree and outputs the sum of all the elements in the tree.

```python
def sum_of_nodes(t):

    """
    >>> t = tree(...)  # Tree from question 2.
    >>> sum_of_nodes(t)  # 9 + 2 + 4 + 4 + 1 + 7 + 3 = 30
    30
    """

    total = label(t)
    for branch in branches(t):
        total += sum_of_nodes(branch)
    return total

    Alternative solution:
    return label(t) +
        sum([sum_of_nodes(b) for b in branches(t)])
```

---

Computer Science Mentors CS61A Fall 2017: Paul Bitutsky and Jason Goodman, with Colin Schoen, Kevin Lin, Thomas Zhang, Danelle Nachum, Jennie Chen, Nipun Ramakrishnan, Christopher Allsman, Nikunj Jain, Shreya Sahoo, Joshua Zeitsoff
2 Orders of Growth

6. In big-$\Theta$ notation, what is the runtime for $\text{foo}$?
   (a) \texttt{def foo(n):}
   \hspace{1em} for \texttt{i in range(n)}:
   \hspace{2em} \texttt{print('hello')}
   
   \textbf{Solution:} $O(n)$. This is simple loop that will run $n$ times.

   (b) What’s the runtime of $\text{foo}$ if we change $\text{range(n)}$:
       
       i. To $\text{range(n / 2)}$?

       \textbf{Solution:} $O(n)$. The loop runs $n/2$ times, but we ignore constant factors.

       ii. To $\text{range(10)}$?

       \textbf{Solution:} $O(1)$. No matter the size of $n$, we will run the loop the same number of times.

       iii. To $\text{range(10000000)}$?

       \textbf{Solution:} $O(1)$. No matter the size of $n$, we will run the loop the same number of times.

7. What is the order of growth in time for the following functions? Use big-$\Theta$ notation.
   (a) \texttt{def strange_add(n):}
       \hspace{1em} if \texttt{n == 0}:  
       \hspace{2em} \texttt{return 1}
       \hspace{1em} else:
       \hspace{2em} \texttt{return strange_add(n - 1) + strange_add(n - 1)}

   \textbf{Solution:} $\Theta(2^n)$. To see this, try drawing out the call tree. Each level will create two new calls to $\text{strange_add}$, and there are $n$ levels. Therefore, $2^n$ calls.

   (b) \texttt{def stranger_add(n):}
if n < 3:
    return n
elif n % 3 == 0:
    return stranger_add(n - 1) + stranger_add(n - 2) + stranger_add(n - 3)
else:
    return n

Solution: $\Theta(n)$ is $n$ is a multiple of 3, otherwise $\Theta(1)$. The case where $n$ is not a multiple of 3 is fairly obvious – we step into the else clause and immediately return. If $n$ is a multiple of 3, then neither $n-1$ nor $n-2$ are multiples of 3 so those calls will take constant time. Therefore, we just run stranger_add, decrementing the argument by 3 each time.
(c) def waffle(n):
    i = 0
    total = 0
    while i < n:
        for j in range(50 * n):
            total += 1
        i += 1
    return total

Solution: Θ(n^2). Ignore the constant term in 50 * n, and it because just two for loops.

(d) def belgian_waffle(n):
    i = 0
    total = 0
    while i < n:
        for j in range(n ** 2):
            total += 1
        i += 1
    return total

Solution: Θ(n^3). Inner loop runs n^2 times, and the outer loop runs n times. To get the total, multiply those together.

(e) def pancake(n):
    if n == 0 or n == 1:
        return n
    # Flip will always perform three operations and return -n.
    return flip(n) + pancake(n - 1) + pancake(n - 2)

Solution: Θ(2^n). Flip will run in constant time. Therefore, this call tree looks very similar to fib! (which is 2^n)

(f) def toast(n):
    i = 0
    j = 0
    stack = 0
    while i < n:
        stack += pancake(n)
        i += 1
while j < n:
    stack += 1
    j += 1
return stack

Solution: $\Theta(n2^n)$. There are two loops: the first runs $n$ times for $2^n$ calls each time (due to pancake), for a total of $n2^n$. The second loop runs $n$ times. When calculating orders of growth however, we focus on the dominating term – in this case, $n2^n$.

8. Consider the following functions:

```python
def hailstone(n):
    print(n)
    if n < 2:
        return
    if n % 2 == 0:
        hailstone(n // 2)
    else:
        hailstone((n * 3) + 1)

def fib(n):
    if n < 2:
        return n
    return fib(n - 1) + fib(n - 2)

def foo(n, f):
    return n + f(500)
```

In big-$\Theta$ notation, describe the runtime for the following:

(a) foo(10, hailstone)

Solution: $\Theta(1)$. $f(500)$ is independent of the size of the input $n$.

(b) foo(3000, fib)

Solution: $\Theta(1)$. See above.
9. **Orders of Growth and Trees**: Assume we are using the non-mutable tree implementation introduced in discussion. Consider the following function:

```python
def word_finder(t, p, word):
    if root(t) == word:
        p -= 1
        if p == 0:
            return True
    for branch in branches(t):
        if word_finder(branch, p, word):
            return True
    return False
```

(a) What does this function do?

**Solution:** This function takes a Tree `t`, an integer `n`, and a string `word` as input. Then, `word_finder` returns `True` if any paths from the root towards the leaves have at least `n` occurrences of the word and `False` otherwise.

(b) If a tree has `n` total nodes, what is the total runtime in big-$\Theta$ notation?

**Solution:** $\Theta(n)$. At worst, we must visit every node of the tree.