Trees and Orders of Growth

Computer Science Mentors 61A

October 2 to October 6, 2017

1 Trees

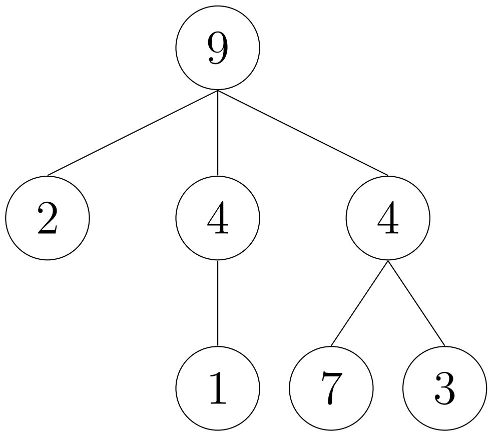
Things to remember:

```
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:

```
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`.



3. What would this output?

```
>>> label(t)

>>> branches(t)[2]

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

4. Write the Python expression to return the integer 2 from t.
5. Write the function `sum_of_nodes` which takes in a tree and outputs the sum of all the elements in the tree.

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

2 Orders of Growth

6. In big-Θ notation, what is the runtime for `foo`?
   (a)
```
def foo(n):
    for i in range(n):
        print('hello')
```

   (b) What's the runtime of `foo` if we change `range(n)`:
      i. To `range(n / 2)`?
      ii. To `range(10)`?
      iii. To `range(10000000)`?

7. What is the order of growth in time for the following functions? Use big-Θ notation.
   (a)
```
def strange_add(n):
    if n == 0:
        return 1
    else:
        return strange_add(n - 1) + strange_add(n - 1)
```

   (b)
```
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
```

(c)
```
def waffle(n):
    i = 0
    total = 0
    while i < n:
        for j in range(50 * n):
            total += 1
        i += 1
    return total
```

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

(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)
```

(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
```

8. Consider the following functions:

```
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-Θ notation, describe the runtime for the following:

(a) `foo(10, hailstone)`

(b) `foo(3000, fib)`

9. **Orders of Growth and Trees:** Assume we are using the non-mutable tree implementation introduced in discussion. Consider the following function:

```
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?

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