## 15-110 Recitation Week 7

## **Reminders**

- 10/22 Tues Check3/HW3 revisions due (Tuesday after break)
- Recitation feedback form
- Have a restful and rejuvenating break!

### Overview

- Big-O Exercise
- For-Iterable Loop Review
- Dictionary Code Writing
- Tree Code Writing

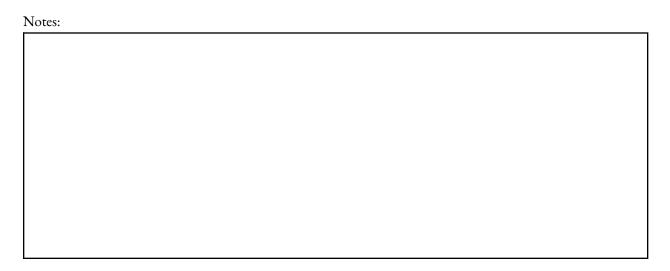
# Problems

#### **BIG-O EXERCISE**

Calculate the Big-O for the following examples:

```
Returning the last character in a string
# input size = n
def powersOfTwo(n):
    m = 1
    while m <= n:
        print(m)
        m = m * 2
# .index(), .pop() are O(n) worst
case!
\# input size = len(L) = n
def foo(L):
    if L == []:
        return 0
    else:
        L.append(L[0])
        n = L.index(10)
        L.pop(0)
        return n
# L is a n by n 2D list
# input size = n
def tripleLoop(L):
    for i in range(20):
        for row in L:
                            0(1)
             for elem in row:
                 print(elem)
```

#### FOR-ITERABLE LOOP REVIEW



Use this code to answer the following questions:

```
s = "15-110"
for i in range(len(s)):
    print(i)
for i in s:
    print(i)
```

What does each loop print?

What is the data type of i in each loop?

#### **DICTIONARY CODE WRITING**

We're given a dictionary that maps some number of football teams (e.g. CMU, Pitt, OSU, PennState) to the number of wins and losses they have (represented as [wins, losses]), and an integer representing the minimum number of games to be considered. We want to return the team with the highest percentage of wins and that has played the minimum number of games. There will be no ties.

<u>E.g.</u> b	estTeam({ "CMU" : [1, 10], "Pitt" : [7, 10], "OSU" : [10, 6], "PennState" : [2, 1] }, 5) returns "OSU
def	<pre>bestTeam(winsLosses, minGames):</pre>
	bestTeam =
	bestPercent =
	for team in winsLosses:
	wins =
	losses =
	gamesPlayed = +
	<pre># check if team played enough games</pre>
	if >= minGames:
	winPercent = / gamesPlayed
	<pre>if &gt; bestPercent:</pre>
	bestPercent =
	bestTeam =
	return

#### TREE CODE WRITING

Write the function addEvenLeaves(t) that takes in a dictionary representation of a tree (you can assume it will have at least 1 node) and returns a sum of **only** the even values held by leaves.

```
def addEvenLeaves(tree):
   # base case: empty tree
  if ____:
      # what should we return?
     return ____
   # recursive case
   else:
      result = 0
      # check if we're at a leaf
      if _____ and ____:
         # check if its value is even
         if ____:
            result += _____
      # recursively add the even leaves of the subtrees
      result +=
      return result
```