

15-110 Recitation Week 6

Reminders

- HW3 due Monday, Feb. 23 at noon
- How was the exam?
- Code reviews this weekend!
- [Recitation feedback form](#)

Overview

- Recursive Code Tracing
- Recursive Code Writing Practice
- Linear and Binary Search
- Bonus: Recursive Catch the Error

Problems

RECURSIVE CODE TRACING

Consider this recursive function:

```
def f(a, b):  
    if a == []:  
        return []  
    else:  
        return [a[0]] + [b[0]] + f(a[1:], b[1:])
```

If we call the function with these values:

```
print(f([1,2,3], [4,5,6]))
```

Trace through the code to determine what will be printed.

RECURSIVE CODE WRITING

Recall **isPalindrome** from week 4 recitation, which we wrote using a loop:

```
def isPalindrome(s):
    leftPointer = 0
    rightPointer = len(s)-1
    while leftPointer < rightPointer:
        front = s[leftPointer]
        back = s[rightPointer]
        if front != back:
            return False
        leftPointer += 1
        rightPointer -= 1
    return True
```

Now, write the **recursive** function **isPalindrome** that takes in a string and returns **True** if the string is a palindrome and **False** otherwise. For example:

isPalindrome("abba") should return **True**

isPalindrome("Dancing Queen") should return **False**

isPalindrome("123321") should return **True**

isPalindrome("") should return **True**

```
def isPalindrome(s):
```

Write the **recursive** function **listMultiply** that takes in a list of integers and returns a new list, where each **non-negative** number in the original list (n) is repeated n times in the new list. Negative numbers should not be included in the resulting list.

`listMultiply([1,2,3])` should return `[1, 2, 2, 3, 3, 3]`

`listMultiply([6])` should return `[6, 6, 6, 6, 6, 6]`

`listMultiply([1,2,3,-4])` should return `[1, 2, 2, 3, 3, 3]`

```
def listMultiply(lst):
```

LINEAR AND BINARY SEARCH

Binary search review notes:

```
def binarysearch(L, item):
    if len(L) == 0:
        return False
    else:
        mid = len(L)//2
        if L[mid] == item:
            return True
        elif L[mid] > item:
            return binarysearch(L[:mid], item)
        else:
            return binarysearch(L[mid+1:], item)
```

Run a visual trace on the list **[2, 4, 6, 7, 10, 11]** and find **1** and **7** using both linear and binary search. Which one, linear or binary search, runs in the smallest number of steps? Will this always be the case?

Linear Search

Binary Search

BONUS: RECURSIVE CATCH THE ERROR

Look at the recursive function below, `countEvens`, which takes in any string `s` and recursively returns the number of **even numbers** present in that string.

```
def countEvens(s):  
    if s == '':  
        return ''  
    else:  
        char = s[0]  
        if char.isdigit() and int(char) % 2 == 0:  
            return 1 + countEvens(s)  
        else:  
            return countEvens(s[1:])  
  
assert(countEvens('12345678') == 4)  
assert(countEvens('0') == 1)  
assert(countEvens('abcd4ef-56') == 2)
```

What two errors are present in the code? How can you fix them so the function runs as intended?