Week: 06 Date: 10/02/2025

15-110 Recitation Week 6

Reminders

- HW3 due Monday, Oct. 6 at noon
- How was the exam?
- Code reviews this weekend!
- Recitation feedback form

Overview

- Aliasing Code Trace
- Recursive Code Tracing
- Recursive Code Writing Practice
- Linear and Binary Search

Problems

CODE TRACING WITH ALIASING

At the end of this set of operations, what list value will each variable hold?

```
x = [ "15110", "Spring" ]
z = [ "15110", "Spring" ]
y = x
x.append("CMU")
y = y + [ "Reci" ]
z.append("CMU")
y.pop(0)
```

x = _____

y = _____

z = _____

Are there any aliases at the end of the code?

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):
     for i in range(len(s)):
          front = s[i]
          back = s[len(s) - 1 - i]
          if front != back:
               return False
     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 number in the original list (n) is repeated n times in the new list.

`listMultiply([1,2,3])` should return `[1, 2, 2, 3, 3, 3]`
`listMultiply([6])` should return `[6, 6, 6, 6, 6, 6]`

```
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