CSCI-SHU 210 Data Structures

Recitation 3 Recursion

You have a series of tasks in front of you. Complete them! Everyone should code on their own computer, but you are encouraged to talk to others, and seek help from each other and from the Professor/TA/LA.

Important:

- **Analyzing the output** for recursive programs;
- **Determining the big O complexity** for recursive programs**;**
- Understand "**Break large problem into smaller problems + induction**";
- Understand what type of problem **branching** can solve.

Problem 1

Recursion output analysis

What is the output for the following recursive program? Don't run it, first try to guess.

```python
def f(n):
    if n > 0:
        f(n - 1)
        print(n, end = " ")
        f(n - 1)


f(4)
```

Problem 2

Determine big-O complexity for the following code snippets:

```python
def func1(N):
    if n < 1:
        return
    else:
        for i in range(N ** 0.5):
            print("hi")
        func1(N - 5)
```

```python
def func2(N):
    if n < 1:
        return
    else:
        func2(N - 1)
        func2(N - 1)
        for i in range(N):
            print("*")
```

Big-O for func1:

____

Big-O for func2:

____

Problem 3

Palindrome (Recursive version)

Implement function palindrome( ): this function assesses whether an input String is indeed a palindrome.

Important:

- Check the string letter by letter, no string.reverse( )
- Use recursion to break large problem into smaller problem.

Problem 4: Tower of Hanoi

Your task is to code the famous Tower of Hanoi problem.

Complete function hanoi( ) in hanoi.py, so the disks move correctly when you run the program.

Important:

- We have graphical demo for this question.
- Start, goal, mid parameters represent three poles.
- To move a disk, call function **game.move(num_disks, start, goal)**
- Use recursion to break large problem into smaller problem.

Problem 5: All Possible Combinations problem

Implement a recursive approach to show all the teams that can be created from a group (out of n things choose k at a time). Implement the recursive showTeams( ), given a group of players, and the size of the team, display all the possible combinations of players.

Important:

- Combination is different from permutation. This is a combination problem.
- There are $\frac{n!}{k!(n-k)!}$ combinations (Choose k out of n) [1, 2], [2, 1] are the same combinations.
- There are $\frac{n!}{(n-k)!}$ permutations (Choose k out of n) [1, 2], [2, 1] are different permutations.
- Understand what is a driver function.

Example Input:

```
players = [“Dey”, “Ruowen”, “Josh”, “Kinder”, “Mario”, “Rock”, “LOL”] # 7
 players

show_team_driver(players, 2) # Choose 2 from 7
```

Should output:

```
['Rock', 'LOL']
['Mario', 'LOL']
['Mario', 'Rock']
['Kinder', 'LOL']
['Kinder', 'Rock']
['Kinder', 'Mario']
['Josh', 'LOL']
['Josh', 'Rock']
['Josh', 'Mario']
['Josh', 'Kinder']
['Ruowen', 'LOL']
['Ruowen', 'Rock']
['Ruowen', 'Mario']
['Ruowen', 'Kinder']
['Ruowen', 'Josh']
['Dey', 'LOL']
['Dey', 'Rock']
['Dey', 'Mario']
['Dey', 'Kinder']
['Dey', 'Josh']
['Dey', 'Ruowen']
```

Another example Input:

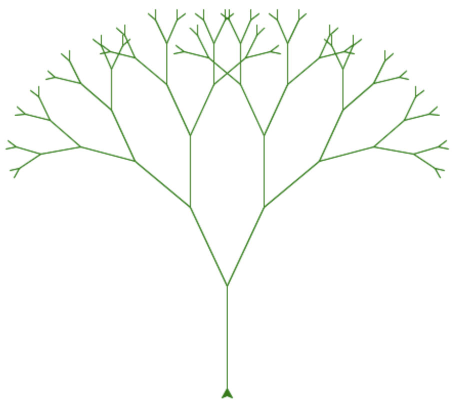
```
players = [“Dey”, “Ruowen”, “Josh”, “Kinder”, “Mario”, “Rock”, “LOL”] # 7
 players

show_team_driver(players, 4) # Choose 4 from 7
```

should output:

```
['Kinder', 'Mario', 'Rock', 'LOL']
['Josh', 'Mario', 'Rock', 'LOL']
['Josh', 'Kinder', 'Rock', 'LOL']
['Josh', 'Kinder', 'Mario', 'LOL']
['Josh', 'Kinder', 'Mario', 'Rock']
['Ruowen', 'Mario', 'Rock', 'LOL']
['Ruowen', 'Kinder', 'Rock', 'LOL']
['Ruowen', 'Kinder', 'Mario', 'LOL']
['Ruowen', 'Kinder', 'Mario', 'Rock']
['Ruowen', 'Josh', 'Rock', 'LOL']
['Ruowen', 'Josh', 'Mario', 'LOL']
['Ruowen', 'Josh', 'Mario', 'Rock']
['Ruowen', 'Josh', 'Kinder', 'LOL']
['Ruowen', 'Josh', 'Kinder', 'Rock']
['Ruowen', 'Josh', 'Kinder', 'Mario']
['Professor Day', 'Mario', 'Rock', 'LOL']
['Professor Day', 'Kinder', 'Rock', 'LOL']
['Professor Day', 'Kinder', 'Mario', 'LOL']
['Professor Day', 'Kinder', 'Mario', 'Rock']
['Professor Day', 'Josh', 'Rock', 'LOL']
['Professor Day', 'Josh', 'Mario', 'LOL']
['Professor Day', 'Josh', 'Mario', 'Rock']
['Professor Day', 'Josh', 'Kinder', 'LOL']
['Professor Day', 'Josh', 'Kinder', 'Rock']
['Professor Day', 'Josh', 'Kinder', 'Mario']
['Professor Day', 'Ruowen', 'Rock', 'LOL']
['Professor Day', 'Ruowen', 'Mario', 'LOL']
['Professor Day', 'Ruowen', 'Mario', 'Rock']
['Professor Day', 'Ruowen', 'Kinder', 'LOL']
['Professor Day', 'Ruowen', 'Kinder', 'Rock']
['Professor Day', 'Ruowen', 'Kinder', 'Mario']
['Professor Day', 'Ruowen', 'Josh', 'LOL']
['Professor Day', 'Ruowen', 'Josh', 'Rock']
['Professor Day', 'Ruowen', 'Josh', 'Mario']
['Professor Day', 'Ruowen', 'Josh', 'Kinder']
```

Problem 6: Use Turtle module draw a Tree (Use recursion)



Turtle module is a python built in module. Turtle module draws lines by moving the cursor.

```
turtle functions explained:
import turtle

t = turtle.Turtle()    # Initialize the turtle

t.left(30)             # The turtle turns left 30 degrees
t.right(30)            # The turtle turns right 30 degrees
t.forward(20)          # The turtle moves forward 20 pixels, leave a line on the path.
t.backward(30)         # The turtle moves backward 30 pixels, leave a line on the path.

… and more! In this recitation, that's all we need.
```

With the mind set of recursion, let's break down this problem.

1. Move forward
2. Make a turn, aim to the direction for the first branch
3. Recursion for a smaller problem
4. Make a turn, aim to the direction for the second branch
5. Recursion for a smaller problem
6. Make a turn, aim to the direction for coming back
7. Come back

Base case: If the branch is too small, stop.
Otherwise, we should create two new branches (two recursions, two smaller problems)

Problem 7 (Optional): Binary Search

Complete function binary_search( ): this function uses a binary search to determine whether an ordered list contains a specified value.

We will implement two versions of binary search:

1. Recursive
2. Iterative