```
!date
```

```
Mon 13 Feb 2023 03:53:56 AM UTC
```

**Please run the above line to refresh the date before your submission.**

# 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 TA/LA.

Important:

1. Analyzing the output for recursive programs;
2. Determining the big O complexity for recursive programs;
3. Understand “Break large problem into smaller problems + induction”;
4. Understand what type of problem branching can solve.

Name: Margad Gereltbaatar MJ

NetID:mg7502

- For students who have recitation on Wednesday, you should submit your solutions by Friday 11:59pm.
- For students who have recitation on Thursday, you should submit your solutions by Saturday 11:59pm.
- For students who have recitation on Friday, you should submit your solutions by Sunday 11:59pm.

## Problem 1

### Recursion output analysis

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

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

f(4)
```

link text#### Your Answer: 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

## Problem 2

### Determine big-O complexity for the following code snippets:

```
def func1(N):
    if N < 1:
        return
    else:
        for i in range(5*N):
```

```
            print("hi")
        func1(N - 5)
```

#### Your Answer: O(n)

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

link text#### Your Answer: O(N2^N)

## Problem 3

Palindrome (Recursive version)

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

Important:

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

```
def palindrome_recursive(string, index):
    """
    # Complete the palindrome algorithm --- with recursion
    # Think about how to break a large problem into smaller sub problems.
    What is our base case in this problem?

    # Another way to ask: what is our smallest problem?
    How to get to this smallest problem?

    :param string: String -- the string to check whether it is a palindrome
    :param index: Int -- additional parameter for recursion tracking

    :return: True if @string is palindrome, False otherwise
    """
    if index >= len(string) // 2:
        return True
    if string[index] != string[len(string) - 1 - index]:
        return False
    return palindrome_recursive(string, index + 1)
def main():
    s1 = "nodevillivedon"
    s2 = "livenoevil!liveonevil"
    s3 = "beliveivileb"
    r1 = palindrome_recursive(s1, 0)
    r2 = palindrome_recursive(s2, 0)
    r3 = palindrome_recursive(s3, 0)
    print("s1 is", r1)  # Should be True
    print("s2 is", r2)  # Should be True
    print("s3 is", r3)  # Should be False

main()
```

```
s1 is True
s2 is True
s3 is False
```

## Problem 4

Tower of Hanoi

Your task is to code the famous Tower of Hanoi problem. Complete function hanoi( ), so the disks move correctly when you run the program.

Important:

1. We have graphical demo for this question. (Only works in your local IDE)
2. Start, goal, mid parameters represent three poles.
3. To move a disk, call function game.move(num_disks, start, goal)
4. Use recursion to break large problem into smaller problem.

```
from tkinter import Tk, Canvas

def hanoi(num_disks, start, goal, mid):
    """
    Implement the tower of hanoi algorithm.
    If implemented correctly, a window should pop up, and disks will move as desired.
    to move a disk, you can call
    game.move(num_disks, start, goal)

    :param num_disks: number of disks in this function call.
    :param start: Int -- start moving from this pole
    :param goal: Int -- move to this pole
    :param mid: Int -- Intermediate (useless) pole for this move.

    :return: nothing to return
    """
    if num_disks > 0:
        hanoi(num_disks - 1, start, mid, goal)
        game.move(num_disks, start, goal)
        hanoi(num_disks - 1, mid, goal, start)








# The graphical interface
class Tkhanoi:

    # Create our objects
    def __init__(self, n, bitmap = None):
        self.n = n
        self.tk = tk = Tk()
        self.canvas = c = Canvas(tk)
        c.pack()
        width, height = tk.getint(c['width']), tk.getint(c['height'])

        # Add background
        if bitmap:
            self.bitmap = c.create_bitmap(width//2, height//2,
                                          bitmap=bitmap,
                                          foreground='blue')

        # Generate pegs
        pegwidth = 10
        pegheight = height//2
        pegdist = width//3
        x1, y1 = (pegdist-pegwidth)//2, height*1//3
        x2, y2 = x1+pegwidth, y1+pegheight
        self.pegs = []
        p = c.create_rectangle(x1, y1, x2, y2, fill='black')
        self.pegs.append(p)
        x1, x2 = x1+pegdist, x2+pegdist
        p = c.create_rectangle(x1, y1, x2, y2, fill='black')
        self.pegs.append(p)
        x1, x2 = x1+pegdist, x2+pegdist
        p = c.create_rectangle(x1, y1, x2, y2, fill='black')
        self.pegs.append(p)
        self.tk.update()

        # Generate pieces
        pieceheight = pegheight//16
        maxpiecewidth = pegdist*2//3
        minpiecewidth = 2*pegwidth
        self.pegstate = [[], [], []]
        self.pieces = {}
        x1, y1 = (pegdist-maxpiecewidth)//2, y2-pieceheight-2
        x2, y2 = x1+maxpiecewidth, y1+pieceheight
        dx = (maxpiecewidth-minpiecewidth) // (2*max(1, n-1))
        for i in range(n, 0, -1):
            p = c.create_rectangle(x1, y1, x2, y2, fill='red')
            self.pieces[i] = p
            self.pegstate[0].append(i)
            x1, x2 = x1 + dx, x2-dx
            y1, y2 = y1 - pieceheight-2, y2-pieceheight-2
```

```
            self.tk.update()
            self.tk.after(25)

    def run(self):
        hanoi(self.n, 0, 2, 1)

    # Reporting callback for the actual hanoi function
    def move(self, i, a, b):
        if self.pegstate[a][-1] != i: raise RuntimeError # Assertion
        del self.pegstate[a][-1]
        p = self.pieces[i]
        c = self.canvas

        # Lift the piece above peg a
        ax1, ay1, ax2, ay2 = c.bbox(self.pegs[a])
        while 1:
            x1, y1, x2, y2 = c.bbox(p)
            if y2 < ay1: break
            c.move(p, 0, -1)
            self.tk.update()

        # Move it towards peg b
        bx1, by1, bx2, by2 = c.bbox(self.pegs[b])
        newcenter = (bx1+bx2)//2
        while 1:
            x1, y1, x2, y2 = c.bbox(p)
            center = (x1+x2)//2
            if center == newcenter: break
            if center > newcenter: c.move(p, -1, 0)
            else: c.move(p, 1, 0)
            self.tk.update()

        # Move it down on top of the previous piece
        pieceheight = y2-y1
        newbottom = by2 - pieceheight*len(self.pegstate[b]) - 2
        while 1:
            x1, y1, x2, y2 = c.bbox(p)
            if y2 >= newbottom: break
            c.move(p, 0, 1)
            self.tk.update()

        # Update peg state
        self.pegstate[b].append(i)


bitmap = None
game = Tkhanoi(6, bitmap)
game.run()
```

```
---------------------------------------------------------------------------
TclError                                  Traceback (most recent call last)
<ipython-input-1-5de23b4635c9> in <cell line: 124>()
    122
    123 bitmap = None
--> 124 game = Tkhanoi(6, bitmap)
    125 game.run()

                              1 frames
/usr/lib/python3.10/tkinter/__init__.py in __init__(self, screenName, baseName,
className, useTk, sync, use)
   2297                 baseName = baseName + ext
   2298         interactive = False
-> 2299         self.tk = _tkinter.create(screenName, baseName, className, interactive,
wantobjects, useTk, sync, use)
   2300         if useTk:
   2301             self._loadtk()

TclError: no display name and no $DISPLAY environment variable
```

## 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:

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

```python
import copy

def show_team(names, team_size):
    help_show_team(names, team_size, [], 0)

def help_show_team(names, team_size, result_list, position):
    """
    Recursively prints all combinations of players for a team of a given size.
    """
    # Base case 1: Enough players in the result_list.
    if len(result_list) == team_size:
        print(result_list)
        return

    # Base case 2: All players have been checked.
    if position >= len(names):
        return

    # Create two branches:

    # Branch 1: Add current person to the result_list and recurse.
    result_list.append(names[position])
    help_show_team(names, team_size, result_list, position + 1)

    # Branch 2: Do not add current person to the result_list (backtrack) and recurse.
    # Since we've modified result_list in place, we need to remove the last element before the next step.
    result_list.pop()
    help_show_team(names, team_size, result_list, position + 1)

# Example usage:
players = ["Dey", "Ruowen", "Josh", "Kinder", "Mario", "Rock", "LOL"]
show_team(players, 2)
```

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

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

## Problem 6

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

```
import random
def binary_search_rec(x, sorted_list):
    def helper(low, high):
        if low > high:
            return False
        mid = (low + high) // 2
        if sorted_list[mid] == x:
            return True
        elif sorted_list[mid] > x:
            return helper(low, mid - 1)
        else:
            return helper(mid + 1, high)
        return helper(0, len(sorted_list) - 1)

def binary_search_iter(x, sorted_list):
    for i in sorted_list:
        if i == x:
            return True
        if i > x:
            return False


def main():
    sorted_list = []
    for i in range(100):
        sorted_list.append(random.randint(0, 100))
    sorted_list.sort()

    print("Testing recursive binary search ...")
    for i in range(5):
        value = random.randint(0, 100)
        answer = binary_search_rec(value, sorted_list)
        if (answer == True):
            print("List contains value", value)
        else:
            print("List does not contain value", value)

    print("Testing iterative binary search ...")
    for i in range(5):
        value = random.randint(0, 100)
        answer = binary_search_iter(value, sorted_list)
        if (answer == True):
            print("List contains value", value)
        else:
            print("List does not contain value", value)

main()
```

```
Testing recursive binary search ...
List does not contain value 39
List does not contain value 92
List does not contain value 92
List does not contain value 35
List does not contain value 77
```

```
Testing iterative binary search ...
List does not contain value 34
List contains value 23
List contains value 19
List contains value 95
List does not contain value 44
```

## Problem 7 (Optional)

Use Turtle module draw a Tree (**Use recursion**)

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

**For example (the following works only in your IDE)**

```
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)

```
!pip3 install ColabTurtle
```

```
Collecting ColabTurtle
  Downloading ColabTurtle-2.1.0.tar.gz (6.8 kB)
Building wheels for collected packages: ColabTurtle
  Building wheel for ColabTurtle (setup.py) ... done
  Created wheel for ColabTurtle: filename=ColabTurtle-2.1.0-py3-none-any.whl size=7657 sha256=6488f29ea007925f533f1fdf0491ab5ba67b67b378
  Stored in directory: /root/.cache/pip/wheels/0d/ab/65/cc4478508751448dfb4ecb20a6533082855c227dfce8c13902
Successfully built ColabTurtle
Installing collected packages: ColabTurtle
Successfully installed ColabTurtle-2.1.0
```

```
from ColabTurtle import Turtle

def draw_tree(branchLen,t):
    """
    Figure out the tree pattern, then display the recursion tree.
```