Tutorial 12 (week of November 11th)

- Covered recursion this week week. Plus API calling (though this may be kept an extra topic.)
- Clarify doubts about them. In particular: show some recursive programs on python tutor.
- Do some problems in the class for writing recursive functions. For this, make sure before coding you discuss (i) base case, (ii) the recursive case, and (iii) some argument that recursive case will eventually hit the base case.

1. Do the pascal's triangle problem given in the slides (which I did not cover in the class - left it as self reading).
2. Recursive program for prime number checking: https://www.geeksforgeeks.org/recursive-program-prime-number/
3. Recursive implementation of the binary search algorithm
   Explanation:
   Assume that you have a sorted list of numbers. Use the recursive binary search approach to check if a number is present in the list or not. If it is present, return its index.
   Approach:
   1. **Base case:** If low > high, the target is not in the list, so return -1.
   2. **Find the Midpoint:** Calculate the middle index.
   3. **Compare Target with Midpoint:**
      a. If lst[mid] is equal to the target, return mid (target found).
      b. If lst[mid] is greater than the target, recursively search the left half.
      c. If lst[mid] is less than the target, recursively search the right half.

Code:

```
    def binary_search_recursive(lst, target, low, high):
        # Base case: if the interval is invalid
        if low > high:
            return -1  # Target not found

        # Find the middle index
        mid = (low + high) // 2

        # Check if the middle element is the target
        if lst[mid] == target:
            return mid
        # If target is smaller, search the left half
        elif lst[mid] > target:
            return binary_search_recursive(lst, target, low,
    mid - 1)
        # If target is larger, search the right half
        else:
            return binary_search_recursive(lst, target, mid +
    1, high)

# Example usage
lst = [-8, 3, 5, 7, 9, 11]
target = 7
result = binary_search_recursive(lst, target, 0, len(lst) - 1)

if result != -1:
    print(f"Element found at index {result}")
else:
    print("Element not found")
```

If time permits, do the Tower of Hanoi Problem also.

4. **Tower of Hanoi:** Write a recursive function to solve the Tower of Hanoi problem.

   Given 3 rods (A, B, C) and n disks.

   Problem: Move the entire stack of disks from rod A to C

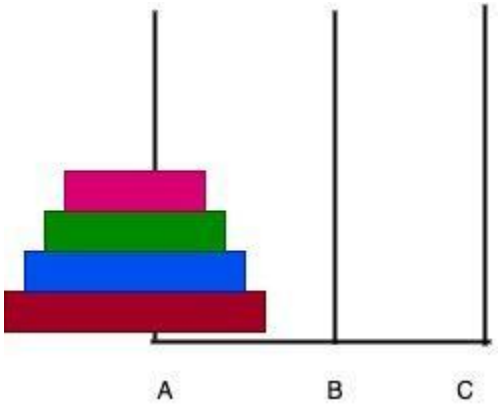
A : Source

B : Auxiliary

C : Destination

Rules:

1. Only one disk can be moved at a time.
2. A disk can only be moved if it is the uppermost disk in the stack.
3. Larger disk cannot be placed on top of a smaller disk



Recursive Approach:

The idea is to recursively break down the problem:

1. **Move n−1 disks** from the source rod (A) to the auxiliary rod (B).
2. Move the **largest disk** (disk n) from the source rod (A) to the target rod (C).
3. **Move n−1 disks** from the auxiliary rod (B) to the target rod (C).

Code:

```
def tower_of_hanoi(n, source, target, auxiliary):
    if n == 1:
        print(f"Move disk 1 from {source} to {target}")
        return
    tower_of_hanoi(n - 1, source, auxiliary, target)
    print(f"Move disk {n} from {source} to {target}")
    tower_of_hanoi(n - 1, auxiliary, target, source)
```

Explanation:

If n=1, we can directly move the disk from A to C.

```
tower_of_hanoi(1, 'A', 'C', 'B')
```

Output:

Move disk 1 from A to C

If n=2, we will have to move the topmost disk (disk 1) to auxiliary (B) and then move the 2nd disk to the target (C). Finally, we can move the disk 1 from Auxiliary to the target.

```
tower_of_hanoi(2, 'A', 'C', 'B')
```

Output:

Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C

Similarly, if n=3

```
tower_of_hanoi(3, 'A', 'C', 'B')
```

Output:

Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C

So, if you have n disks, the Tower of Hanoi problem can be solved in 2^n-1 moves