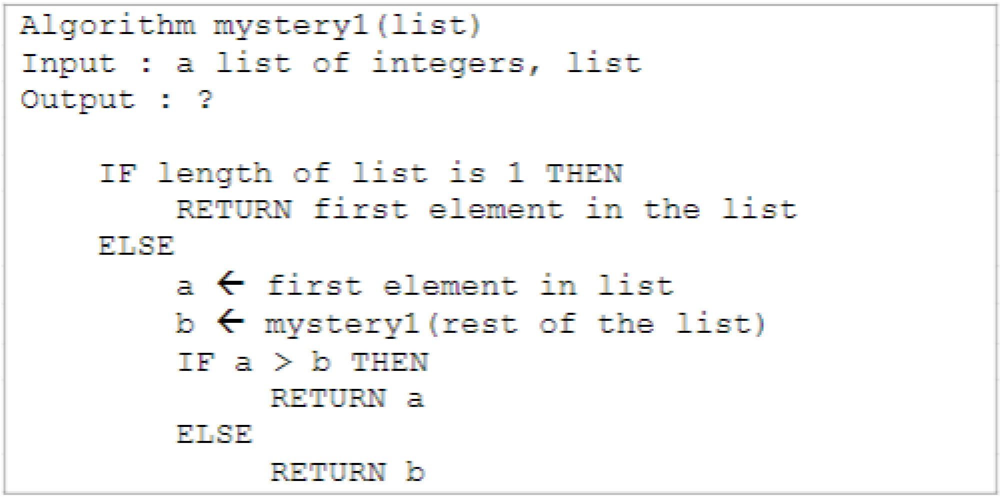
# Lab 5 Recursion

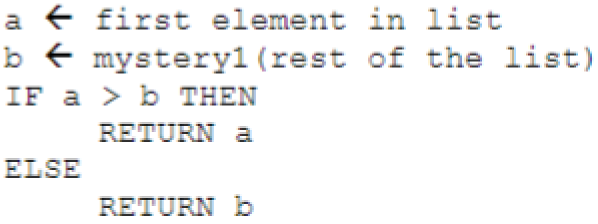
## Pre-Lab



```
Algorithm mystery1(list)
Input : a list of integers, list
Output : ?

    IF length of list is 1 THEN
        RETURN first element in the list
    ELSE
        a ← first element in list
        b ← mystery1(rest of the list)
        IF a > b THEN
             RETURN a
        ELSE
             RETURN b
```

1. What are the base case and recursive case of `mystery1`?
   The base case is `RETURN first element in the list`
   The recursive case is



```
a ← first element in list
b ← mystery1(rest of the list)
IF a > b THEN
     RETURN a
ELSE
     RETURN b
```

2. Trace function `mystery1` for a list of 5 integers. Show the call stack.

| |
|---|
| |
| |
| |
| |
| Mystery1 (1, 2, 3, 4 ,5) list = [1, 2, 3, 4, 5] |

Call Stack

| |
|---|
| |
| |
| |
| Mystery1 ( 2, 3, 4 ,5) list = [2, 3, 4, 5] |
| Mystery1 (1, 2, 3, 4 ,5) list = [1, 2, 3, 4, 5] |

Call Stack

| |
|---|
| |
| |
| Mystery1 (3, 4 ,5) list = [2, 3, 4, 5] |
| Mystery1 (2, 3, 4 ,5) list = [2, 3, 4, 5] |
| Mystery1 (1, 2, 3, 4 ,5) list = [1, 2, 3, 4, 5] |

Call Stack

| |
|---|
| |
| Mystery1 (4 ,5) list = [4, 5] |
| Mystery1 (3, 4 ,5) list = [3, 4, 5] |
| Mystery1 (2, 3, 4 ,5) list = [2, 3, 4, 5] |
| Mystery1 (1, 2, 3, 4 ,5) list = [1, 2, 3, 4, 5] |

Call Stack

| |
|---|
| Mystery1 (5), list = [5] |
| Mystery1 (4 ,5), list = [4, 5] |
| Mystery1 (3, 4 ,5), list = [3, 4, 5] |
| Mystery1 (2, 3, 4 ,5) list = [2, 3, 4, 5] |
| Mystery1 (1, 2, 3, 4 ,5) list = [1, 2, 3, 4, 5] |

Call Stack

| | |
|---|---|
| ~~Mystery1 (5), list = [5]~~ | Return 5 |
| Mystery1 (4 ,5), list = [4, 5] | |
| Mystery1 (3, 4 ,5), list = [3, 4, 5] | |
| Mystery1 (2, 3, 4 ,5) list = [2, 3, 4, 5] | |
| Mystery1 (1, 2, 3, 4 ,5) list = [1, 2, 3, 4, 5] | |

Call Stack

| | |
|---|---|
| ~~Mystery1 (5), list = [5]~~ | Return 5 |
| ~~Mystery1 (4 ,5), list = [4, 5]~~ | Return 5 |
| Mystery1 (3, 4 ,5), list = [3, 4, 5] | |
| Mystery1 (2, 3, 4 ,5) list = [2, 3, 4, 5] | |
| Mystery1 (1, 2, 3, 4 ,5) list = [1, 2, 3, 4, 5] | |

Call Stack

| | |
|---|---|
| ~~Mystery1 (5), list = [5]~~ | Return 5 |
| ~~Mystery1 (4 ,5), list = [4, 5]~~ | Return 5 |
| ~~Mystery1 (3, 4 ,5), list = [3, 4, 5]~~ | Return 5 |
| Mystery1 (2, 3, 4 ,5) list = [2, 3, 4, 5] | |
| Mystery1 (1, 2, 3, 4 ,5) list = [1, 2, 3, 4, 5] | |

Call Stack

| | |
|---|---|
| ~~Mystery1 (5), list = [5]~~ | Return 5 |
| ~~Mystery1 (4 ,5), list = [4, 5]~~ | Return 5 |
| ~~Mystery1 (3, 4 ,5), list = [3, 4, 5]~~ | Return 5 |
| ~~Mystery1 (2, 3, 4 ,5) list = [2, 3, 4, 5]~~ | Return 5 |
| Mystery1 (1, 2, 3, 4 ,5) list = [1, 2, 3, 4, 5] | |

Call Stack

| | |
|---|---|
| ~~Mystery1 (5), list = [5]~~ | Return 5 |
| ~~Mystery1 (4 ,5), list = [4, 5]~~ | Return 5 |
| ~~Mystery1 (3, 4 ,5), list = [3, 4, 5]~~ | Return 5 |
| ~~Mystery1 (2, 3, 4 ,5) list = [2, 3, 4, 5]~~ | Return 5 |
| ~~Mystery1 (1, 2, 3, 4 ,5) list = [1, 2, 3, 4, 5]~~ | Return 5 |

Call Stack

3. What does `mystery1` do?
   Mystery 1 will return the largest number in a given list
4. Write an iterative version of `mystery1` (using pseudocode)

```
a ← 0
list ← input List
FOR i IN list:
    IF i > a:
        a = i
    ELSE:
        CONTINUE
    END IF
END FOR
RETURN a
```

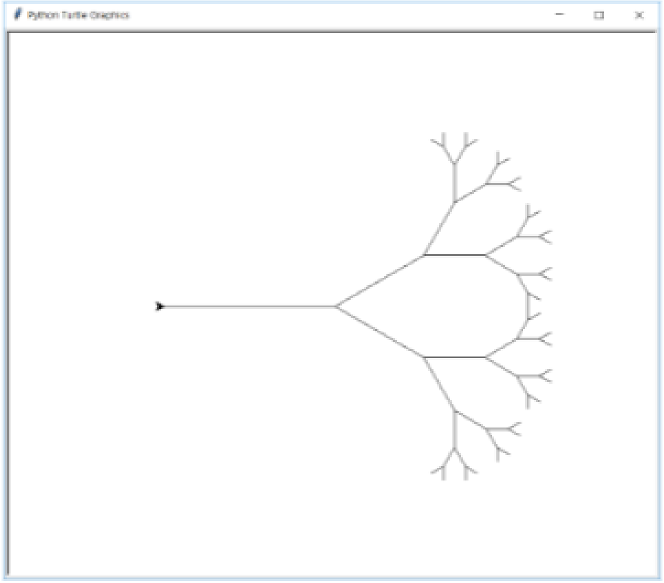
5. Write recursive algorithm using pseudocode to find out
   a. Whether a given string is a Palindrome

```
DEF palindome(str, i):
    IF i > (len(str) / 2):
        RETURN True
    ans ← True
    END IF
    IF ((str[ i ] is str[ len( string ) - i - 1]) and palindome(str, i + 1):
        ans ← True
    RETURN ans
    END IF
```

b. Calculate a * b where a and b are positive integers. Note that you are not allowed to use the “ * “ operator

```
DEF multiply(a, b)
    IF b > 0:
        RETURN (a + multiply(a, b - 1))
    ELSE:
        RETURN 0
    END IF
```

# In-Lab

1. 

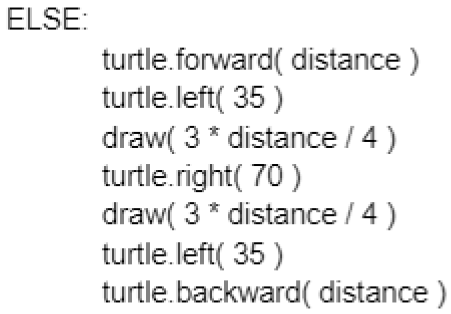
a. Write a recursive algorithm (using pseudocode) to create this tree.

```
DEF draw(distance):
    IF distance < 20:
        hello ← 0
    ELSE:
        turtle.forward( distance )
        turtle.left( 35 )
        draw( 3 * distance / 4 )
        turtle.right( 70 )
        draw( 3 * distance / 4 )
        turtle.left( 35 )
        turtle.backward( distance )
    END IF
```

```
WHILE TRUE:
        draw(85)
END WHILE
```

i. What is the base case?
The base case is if distance < 20, hello ← 0

ii. What is a recursive case?
The recursive case is



```
ELSE:
        turtle.forward( distance )
        turtle.left( 35 )
        draw( 3 * distance / 4 )
        turtle.right( 70 )
        draw( 3 * distance / 4 )
        turtle.left( 35 )
        turtle.backward( distance )
```

b. (optional) Write a program to draw this tree using `turtle`.

# Post-Lab

1. Write an iterative algorithm (using pseudocode) of fib(n)

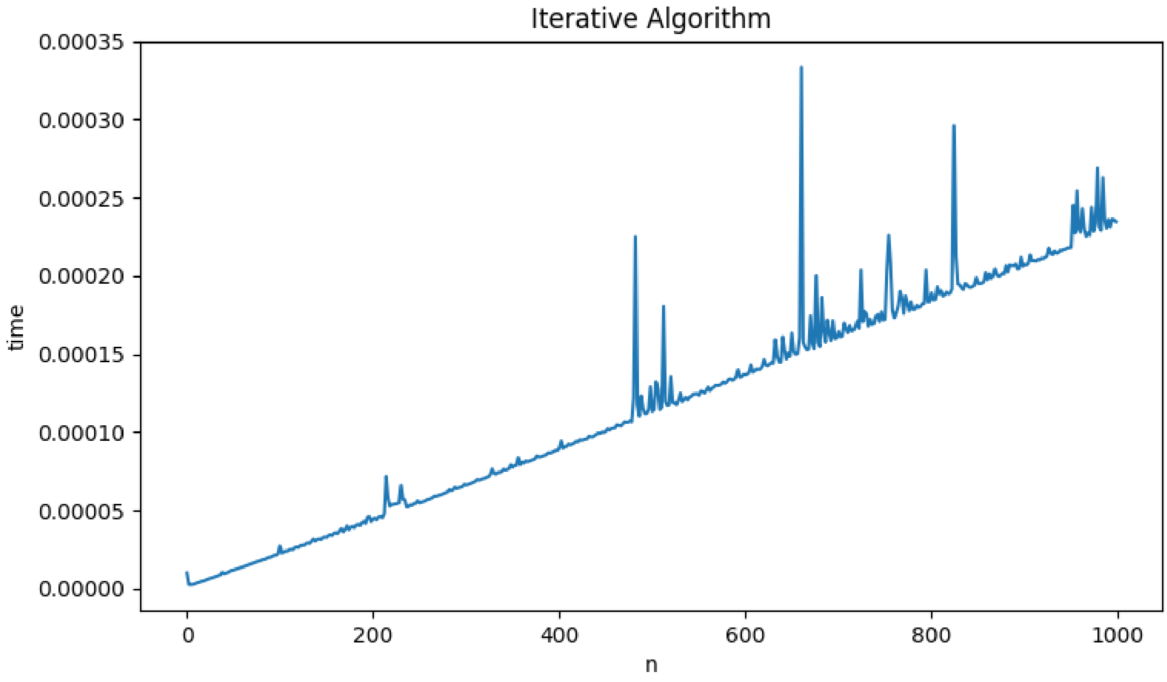
```
a ← 0
b ← 0
ans ← 0
FOR i IN RANGE (n):
        IF i % 2 == 0:
                b ← b + a
                ans ← b
        ELSE:
                a ← a + b
                ans ← a
        END IF
END FOR
RETURN ans
```

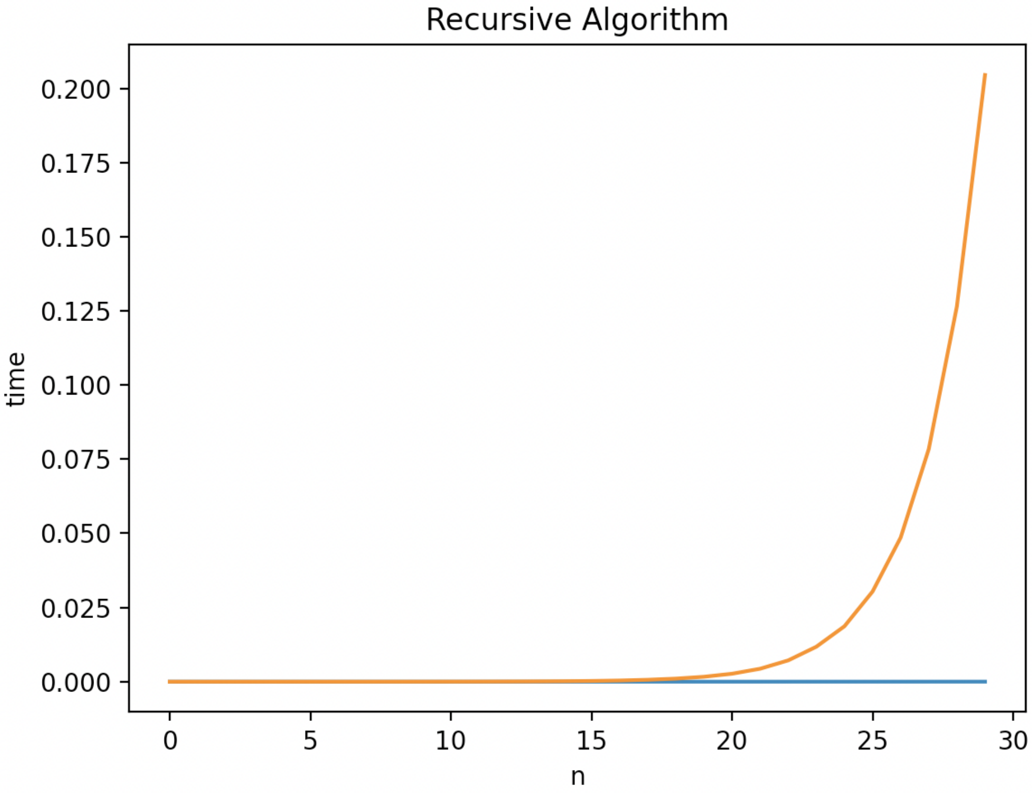
2. Write recursive definition of fib(n)

```
DEF fibo_rec(n):
        IF n <= 1:
```

```
        RETURN n
    END IF
    RETURN fibo_rec(n-1) + fibo_rec(n-2)
```

3. Write a recursive algorithms in 1 and 3 in Python
4. Implement algorithms in 1 and 3 in Python
5. Run your programs against a number of n’s. Record the run time and draw graph (plot the runtime of both versions of Fibonacci on the graph)





6. Explain what you found

I found that using recursive algorithm is slower than iterative algorithm since iterative has Big O of O(n) and recursive has Big O of (n**4)