

# Exercises for lab8

The goal of this lab is to practice writing recursive functions. A recursive function is a function that calls itself with a smaller problem size. A recursive function has a stopping condition, a problem case simple enough that calling the function again is not necessary.

## Question 1: Binary Search

You have seen the binary search in class. However, the algorithm you saw is iterative using high and low indices that move towards each other closing the gap on where the key may exist in the container.

The idea of the binary search can be described as following:

- (1) Compare  $x$  with the middle element.
- (2) If  $x$  matches with middle element, we return the middle index.
- (3) If  $x$  is greater than the middle element, then  $x$  can only lie in the right half subarray, and we do search in that part.
- (4) Similarly, if  $x$  is less than the middle element, we do search in the left subarray.

Whenever we search in the smaller subarray, the same principle applies.

You will need to implement a recursive `binarySearch()` function which takes 4 arguments: *listNumbers*, which is an array assumed to contain elements in ascending order; *low* and *high*, which specify the range we want to search in the array; *key*, which is the element we are searching for.

The function should return the index of the searched element if the element is found, or return -1 if the element is not in the array.

Test your function as follows:

```
# Test array
def main():
    array_for_test = [-8,-2,1,3,5,7,9]
    print(binarySearch(array_for_test,0,len(array_for_test)-1,9))
    print(binarySearch(array_for_test,0,len(array_for_test)-1,-8))
    print(binarySearch(array_for_test,0,len(array_for_test)-1,4))

main()
```

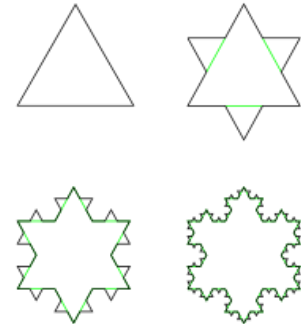
The output for these three cases should be 6, 0, -1. Use the debugger to trace the recursion calls to figure out what is happening.


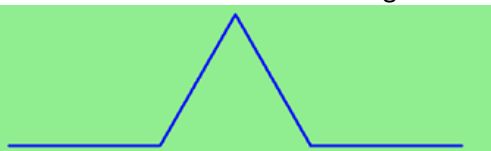
## Question 2: Fractals

Fractals are patterns that are self-similar across different scales. They are created by repeating a simple similar process over and over in a loop. Since the process is repeated often in a smaller and smaller scale, recursion is appropriate to use to create them. You can learn more on fractals on the Wikipedia page: <https://en.wikipedia.org/wiki/Fractal>

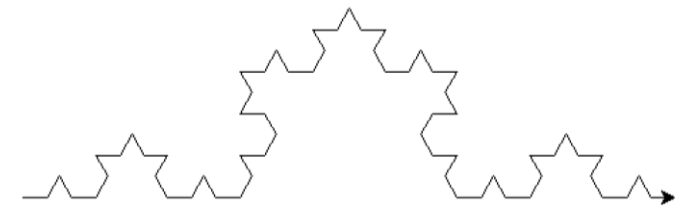
In this lab exercise you will draw part of the Koch Snowflake, also known as Koch fractal. See Wikipedia page [https://en.wikipedia.org/wiki/Koch\\_snowflake](https://en.wikipedia.org/wiki/Koch_snowflake) to know more about it.

The figure on the right illustrates this snowflake. However, we will do just one line of this initial triangle. The Koch function basically transforms a line segment at one level into four line segments at the next level with the four lines forming a peak as follows:



Order 0 Koch fractal: a straight line segment 	Order 1 Koch fractal: Four line segments 
--	--

In general order  $n$  Koch fractal is composed of 4 Order  $(n-1)$  Koch fractals. An order 3 Koch fractal would look like this:



You can see already the recursion. But how will you do this? You can import the Python turtle to do turtle graphics. It is fairly simple. The turtle is this dot on the screen that you can order to move in some directions to trace lines. You can learn more about the turtle here:

<https://docs.python.org/3.3/library/turtle.html>

The following functions would be helpful in this lab exercise: **`turtle.forward(d)`**, **`turtle.right(angle)`**, **`turtle.left(angle)`**.

Write the recursive `Koch()` function which takes 2 arguments, *length* and *order*, and draws the Koch fractal of given order and length.

To draw an Order  $n$  Koch fractal, we can draw the first Order  $(n-1)$  Koch fractal, then turn left by  $60^\circ$  and draw the second piece. After that, make a right turn draw the 3rd piece and finally turn left and draw the last segment.