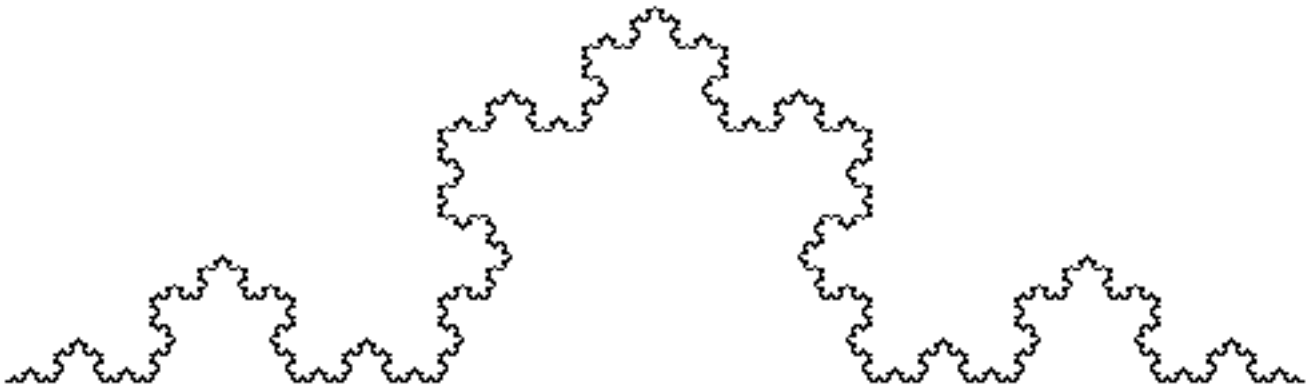
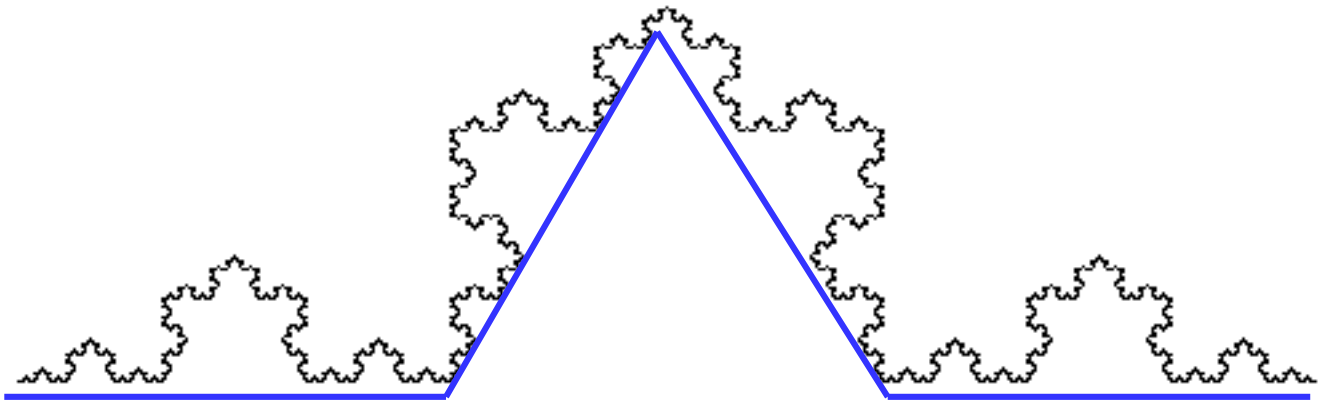
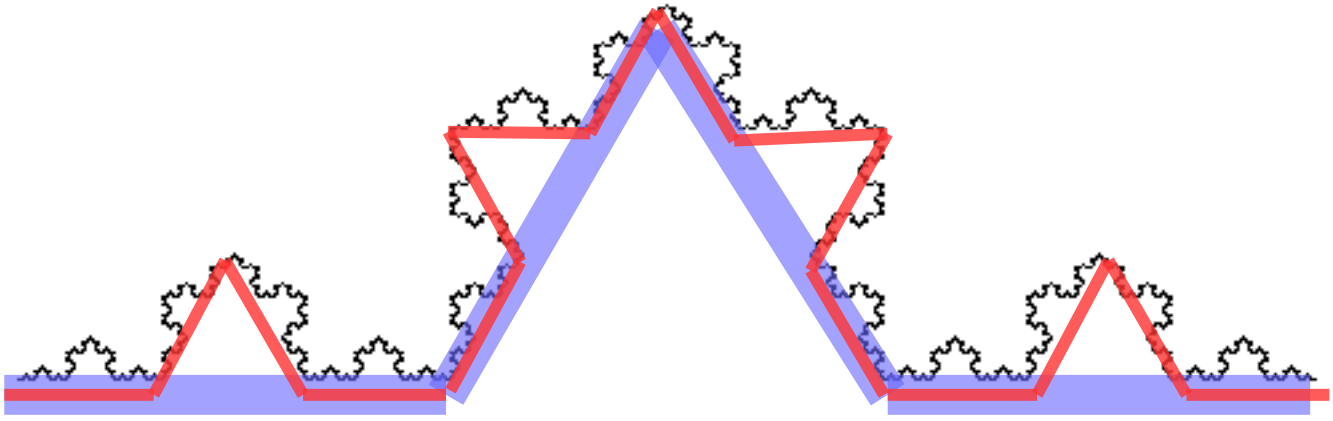
Recursion



Let's take a closer look



Let's take a closer look



This curve is called a **Koch curve.**

Let's see the code

```
def edge(level, d):
    if level > 0:
        edge(level-1, d/2)
        lt(60)
        edge(level-1, d/2)
        rt(120)
        edge(level-1, d/2)
        lt(60)
        edge(level-1, d/2)
    else:
        fd(d)
```

```
def main():
    edge(5,64)
```

How to draw it?

- Follow the following rewrite system.

Alphabet: F
Constants: +, −
Axiom: F
Production rules:
F → F+F--F+F

F → forward
- → turn right 60 degree
+ → turn left 60 degree

```
def edge(level, d):
    if level > 0:
        edge(level-1, d/2)
        lt(60)
        edge(level-1, d/2)
        rt(120)
        edge(level-1, d/2)
        lt(60)
        edge(level-1, d/2)
    else:
        fd(d)
```

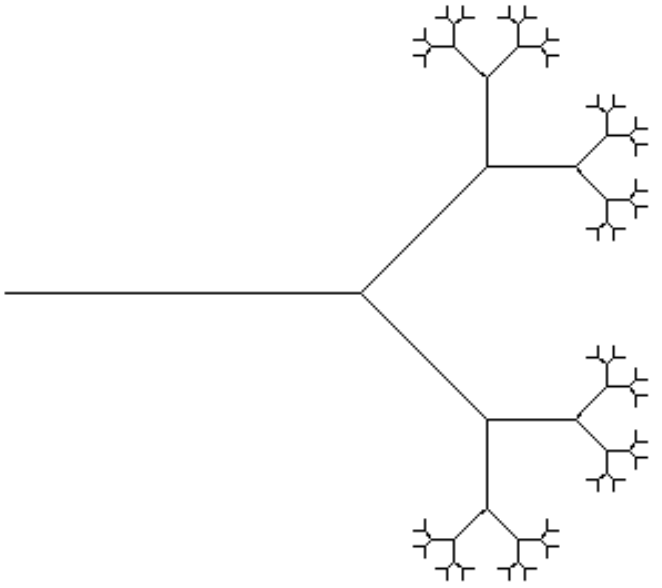
Turtle graphics

- Movement
  - `forward (fd)`
  - `backward (bk,back)`
  - `right (rt)`
  - `left (lt)`
  - `home`
- Pen
  - `pendown (pd)`
  - `penup (pu)`

More complex one

Alphabet: 0, 1
Constants: [,]
Axiom: 0
Production rules:
    1 → 11
    0 → 1[0]0

0,1 → forward
[→ push, left 45
] → pop, right 45



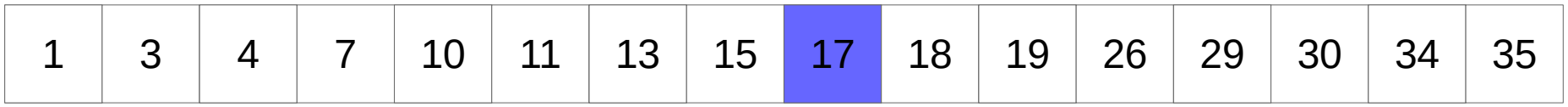
Other curves / fractals

- See https://en.wikipedia.org/wiki/L-system

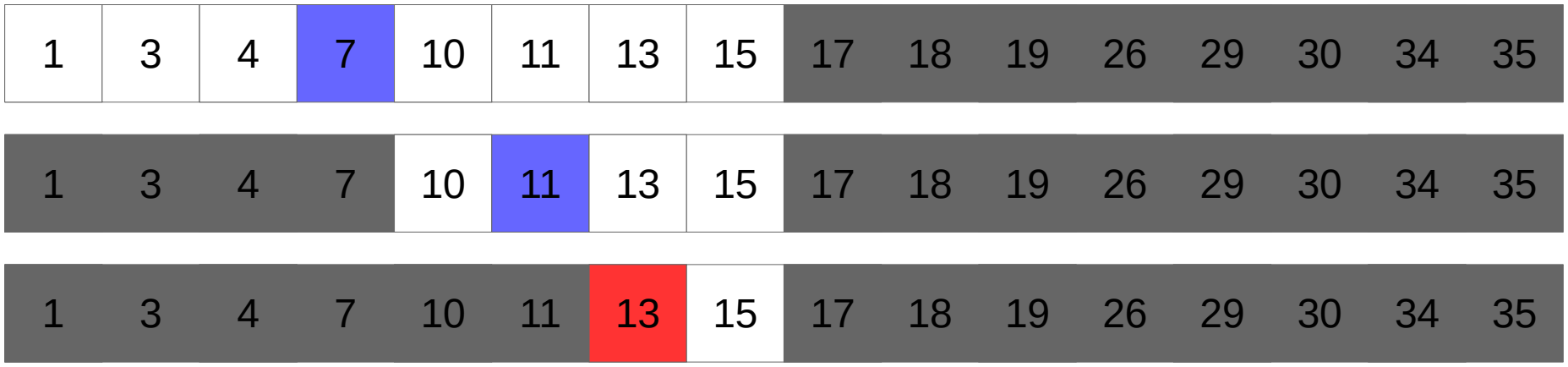
Other applications of recursions

- Binary search
- Sorting algorithms
  - Merge sort
  - Quick sort

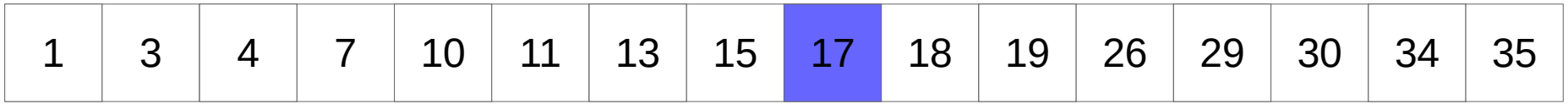
Binary search



find 13



Binary search



find 27

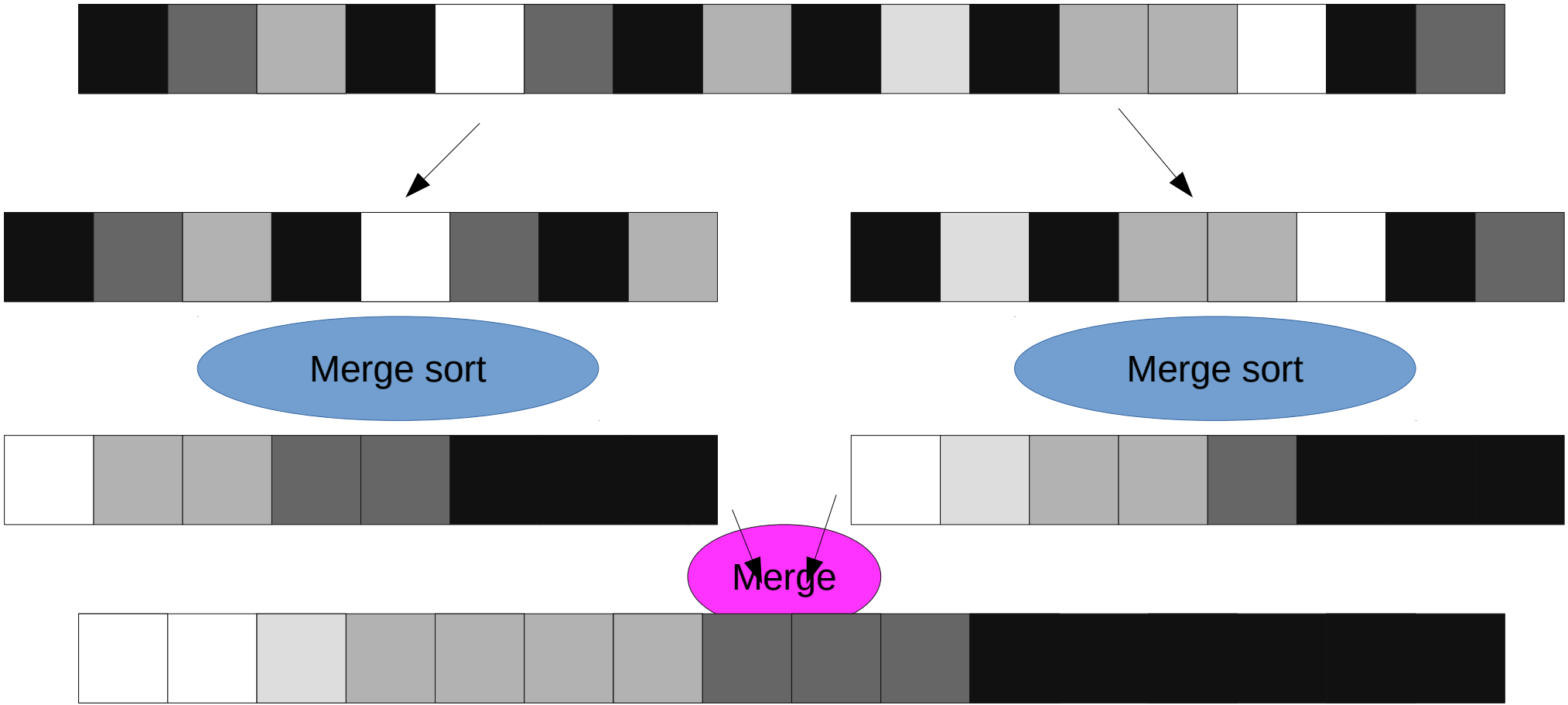
Binary search

```
def binary_search(arr, left, right, x):
    if left > right:
        return -1

    mid = (left + right) // 2
    if arr[mid] == x:
        return mid
    elif arr[mid] < x:
        return binary_search(arr, mid + 1, right, x)
    else:
        return binary_search(arr, left, mid - 1, x)


def search(arr, x):
    return binary_search(arr, 0, len(arr)-1, x)
```

Merge sort



Correctness

- You need mathematical tool called mathematical induction.
- We will discuss about this later on.

Program with no loops

```
def mylen(lst):
    if lst == []:
        return 0
    else:
        return 1 + mylen(lst[1:])
```

mylen([1,3,2,4,5])

- mylen([1,3,2,4,5])
  - 1 + mylen([3,2,4,5])
  - 1 + (1 + mylen([2,4,5]))
  - 1 + (1 + (1 + mylen([4,5])))
  - 1 + (1 + (1 + (1 + mylen([5]))))
  - 1 + (1 + (1 + (1 + (1 + mylen([])))))
  - 1 + (1 + (1 + (1 + (1 + 0))))

Exercise: mysum

```
def mysum(lst):
    """
    >>> mysum([])
    0
    >>> mysum([10])
    10
    >>> mysum([10, 20, 30])
    60
    >>> mysum([1,0,-10,5,9])
    5
    """

    return 0
```

Exercise: mymax

```
def mymax(lst):
    """
    >>> mymax([1])
    1
    >>> mymax([-10, -30, -50])
    -10
    >>> mymax([1, 2, 3, -10])
    3
    >>> mymax([5000,6,7,100,2])
    5000
    >>> mymax([5,6,7,100,2])
    100
    >>> mymax([5,6,7,100,200])
    200
    """
    return 0
```

Exercise: mymerge

```
def mymerge(lst1, lst2):
    """
    >>> mymerge([1,2,3],[])
    [1, 2, 3]
    >>> mymerge([],[4,5,6])
    [4, 5, 6]
    >>> mymerge([1,2,5,6,10],[2,3,4,5,8,9,12])
    [1, 2, 2, 3, 4, 5, 5, 6, 8, 9, 10, 12]
    >>> mymerge([6],[2,3,4,5,8,9,12])
    [2, 3, 4, 5, 6, 8, 9, 12]
    >>> mymerge([2,3,4,5,8,9,12],[7])
    [2, 3, 4, 5, 7, 8, 9, 12]
    """
    return []
```