COMPSCI 105 S1 2017
Principles of Computer Science

22-Recursion(3)

Agenda & Readings

- Agenda
  - Radix Conversion
  - The Fibonacci Sequence
  - The Towers of Hanoi
  - Binary Search
- Reference:
  - Textbook:
    - Problem Solving with Algorithms and Data Structures
      - Chapter 4 – Recursion

Radix Conversion

- Radix is the base of number representation
  - Examples:
    - Decimal, 10
    - Binary, 2
    - Octal, 8
    - Hexadecimal, 16

Decimal	Binary	Octal	Hexadecimal
20	10100_2	24_8	14_{16}
7	111_2	7_8	7_{16}
32	100000_2	40_8	20_{16}

Radix Conversion

- Conversion by division from larger base to a smaller base
  - Examples: Decimal to Octal
    - 735 / 8 = 91 … 7
    - 91 / 8 = 11 … 3
    - 11 / 8 = 1 … 3
    - 735 = 1337_8

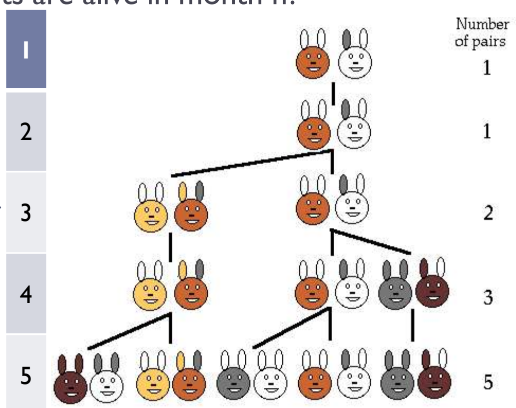
```
def Dec_to_Oct(n):
    a = n // 8
    b = n % 8
    if (a > 0):
        result = b + 10 * Oct_to_Dec(a)
    else:
        result = b
    return result
```

The Fibonacci Sequence

- Describes the growth of an idealized (biologically unrealistic) rabbit population, assuming that:
  - Rabbits never die
  - A rabbit reaches sexual maturity exactly two months after birth, that is, at the beginning of its third month of life
  - Rabbits are always born in male-female pairs
  - At the **beginning** of every month, each sexually mature male-female pair gives **birth** to exactly one male-female pair

The Fibonacci Sequence

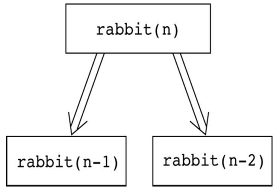
- Problem:
  - How many pairs of rabbits are alive in month n?
- Example:
  - rabbit(5) = 5



- Recurrence relation
  - rabbit(n) = rabbit(n-1) + rabbit(n-2)

Recursive Definition

- Base cases
  - rabbit(2), rabbit(1)
- Recursive case
  - rabbit(n) = 1 if n is 1 or 2; rabbit(n-1) + rabbit(n-2) if n > 2



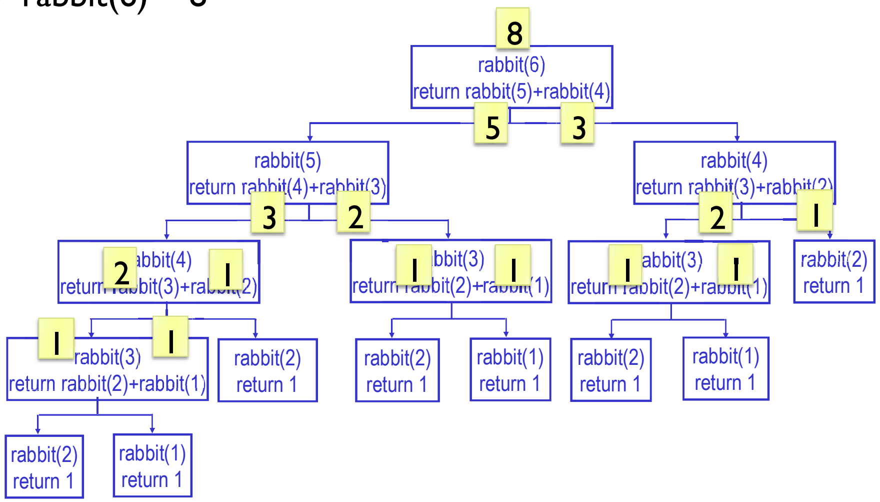
- Fibonacci sequence
  - The series of numbers rabbit(1), rabbit(2), rabbit(3), and so on

The sequence of numbers rabbit(n) for all n is called Fibonacci Sequence or Fibonacci numbers

```
def rabbit(n):
  if n <=2:
    return 1
  return rabbit(n-1) + rabbit(n-2)
```

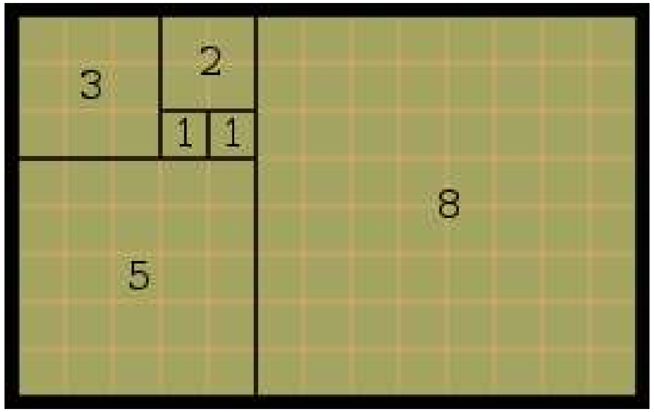

Examples

- rabbit(6) = 8



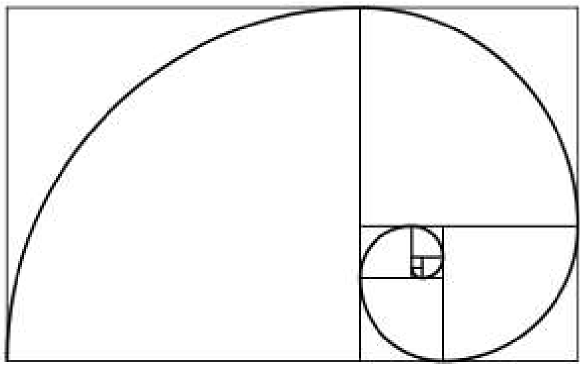
Examples

- Fibonacci Tiling



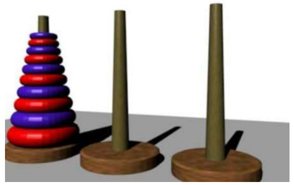
Examples

- Fibonacci Spiral



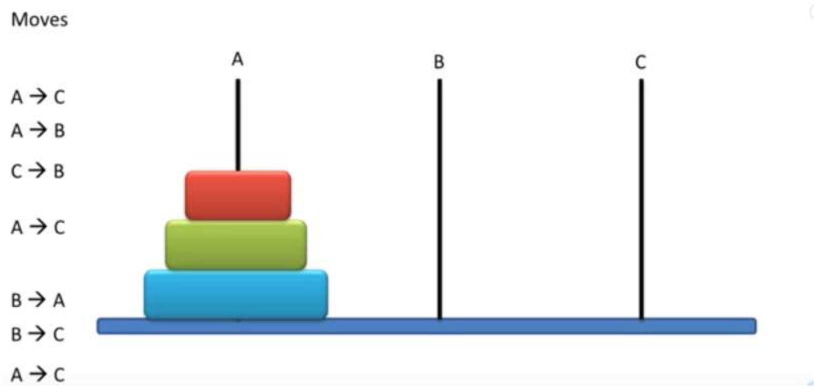
The Towers of Hanoi

- Puzzle consists of n disks and three poles
  - The disks are of different size and have holes to fit themselves on the poles
  - Initially all the disks were on one pole, e.g., pole A
  - The task was to move the disks, one by one, from pole A to another pole B, with the help of a spare pole C
  - Due to its weight, a disks could be placed only on top of another disk larger than itself



The Towers of Hanoi

- Example:
  - https://www.youtube.com/watch?v=5QuiCcZKyYU



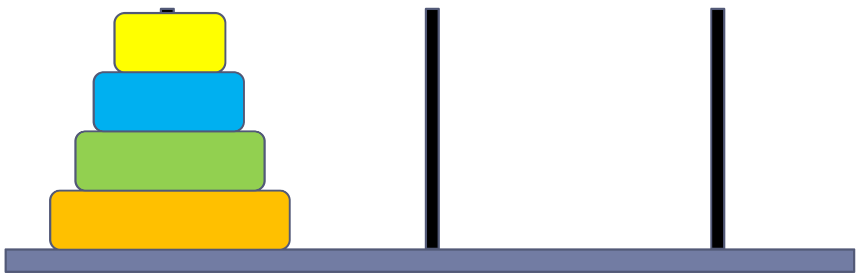
The Towers of Hanoi

- Solution for moving n disks from A to B
  - If you have only one disk (i.e., n=1)
    - Move it from pole A to pole B
  - If you have more than one disk,
    - Simply ignore the bottom disk and solve the problem for n-1 disk, with pole C is the destination and pole B is the spare
    - Then move the largest disk from pole A to B; then move the n-1 disks from the pole C back to pole B
  - We can use a recursion with the arguments:
    - Number of disks, source pole, destination pole, spare pole

The Towers of Hanoi

- Examples:

```
def hanoi(count,source,destination,spare):
    if count is 1:
        Move a disk directly from source to destination
    Move count-1 disks from source to spare
    Move 1 disk from source to destination
    Move count-1 disk from spare to destination
```



The Towers of Hanoi

- Satisfies the four criteria of a recursive solution
  - Recursive method calls itself
  - Each recursive call solves an identical, but smaller problem
  - Stops at base case
  - Base case is reached in finite time

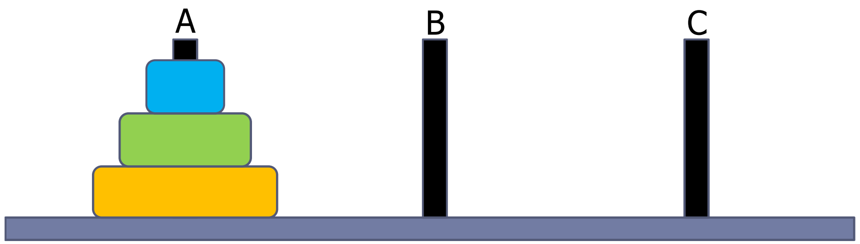
```
def hanoi(count, source, destination, spare):
    if count <= 1:
        print ("base case: move disk from", source, "to", destination)
    else:
        hanoi(count - 1, source, spare, destination)
        print ("step2: move disk from", source, "to", destination)
        hanoi(count - 1, spare, destination, source)
```


The Towers of Hanoi

- Examples:

```
Case 3
hanoi(3, A, B, C)
Count: 3
Source: A
Spare: B
Dest: C
```

```
def hanoi(count, source, destination, spare):
    if count <= 1:
        print ("base case: move disk from", source, "to", destination)
    else:
        hanoi(count - 1, source, spare, destination)
        print ("step2: move disk from", source, "to", destination)
        hanoi(count - 1, spare, destination, source)
```

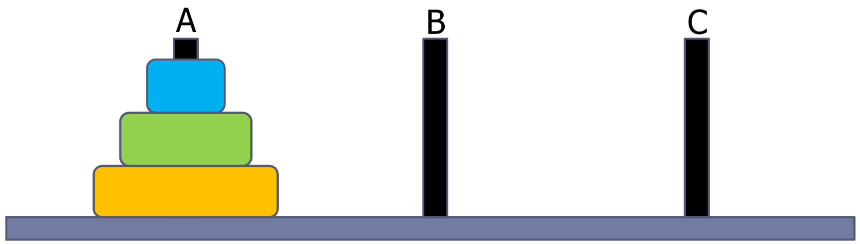


The Towers of Hanoi

- Examples:

Case 3_1.2
hanoi(2, A, C, B)
Count: 2
Source: A
Spare: C
Dest: B

```
def hanoi(count, source, destination, spare):
    if count <= 1:
        print ("base case: move disk from", source, "to", destination)
    else:
        hanoi(count - 1, source, spare, destination)
        print ("step2: move disk from", source, "to", destination)
        hanoi(count - 1, spare, destination, source)
```



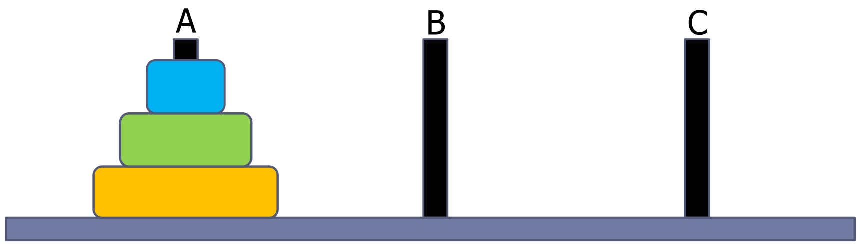
The Towers of Hanoi

- Examples:

Case $3_1.2_1.1$
hanoi(1, A, B, C)
Count: 1
Source: A
Spare: B
Dest: C

```
def hanoi(count, source, dest
    if count <= 1:
        print ("base case: move disk from", source, "to", destination)
    else:
        hanoi(count - 1, source, spare, destination)
        print ("step2: move disk from", source, "to", destination)
        hanoi(count - 1, spare, destination, source)
```

base case: move disk from A to B



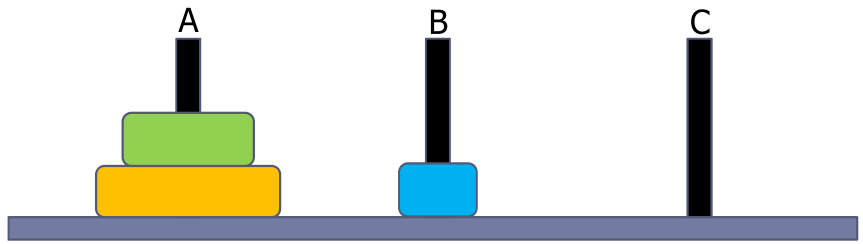
The Towers of Hanoi

- Examples:

Case 3_1.2
hanoi(2, A, C, B)
Count: 2
Source: A
Spare: C
Dest: B

```
def hanoi(count, source, destination, spare):
    if count <= 1:
        print ("base case: move dis                destination)
    else:
        hanoi(count - 1, source
        print ("step2: move disk from", source, "to", destination)
        hanoi(count - 1, spare, destination, source)
```

base case: move disk from A to C



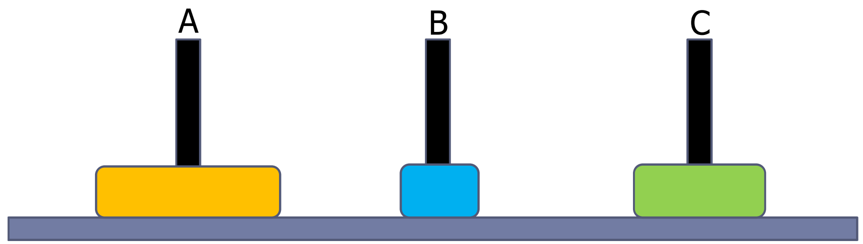
The Towers of Hanoi

- Examples:

Case $3_1.2_2.1$
hanoi(1, B, C, A)
Count: 1
Source: B
Spare: C
Dest: A

```
def hanoi(count, source, dest
    if count <= 1:
        print ("base case: move disk from", source, "to", destination)
    else:
        hanoi(count - 1, source, spare, destination)
        print ("step2: move disk from", source, "to", destination)
        hanoi(count - 1, spare, destination, source)
```

base case: move disk from B to C



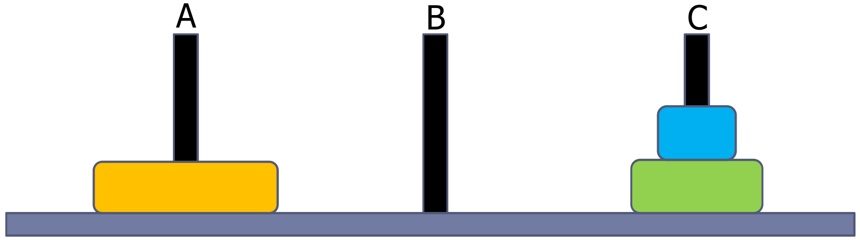
The Towers of Hanoi

- Examples:

Case 3
hanoi(3, A, B, C)
Count: 3
Source: A
Spare: B
Dest: C

```
def hanoi(count, source, destination, spare):
    if count <= 1:
        print ("base case: move dis                destination)
    else:
        hanoi(count - 1, source
        print ("step2: move disk from", source, "to", destination)
        hanoi(count - 1, spare, destination, source)
```

step2: : move disk from A to B



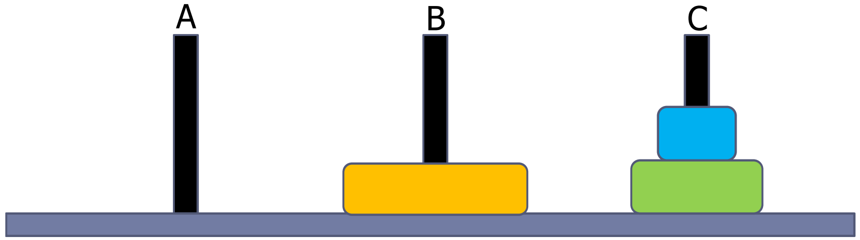
22.3 The Towers of Hanoi

The Towers of Hanoi

- Examples:

Case $3_2.2$
hanoi(2, C, B, A)
Count: 2
Source: C
Spare: B
Dest: A

```
def hanoi(count, source, destination, spare):
    if count <= 1:
        print ("base case: move disk from", source, "to", destination)
    else:
        hanoi(count - 1, source, spare, destination)
        print ("step2: move disk from", source, "to", destination)
        hanoi(count - 1, spare, destination, source)
```



22.3 The Towers of Hanoi

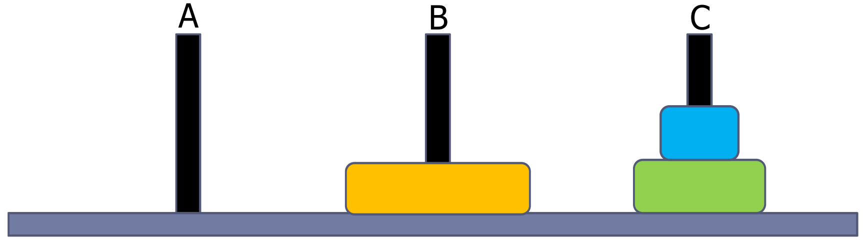
The Towers of Hanoi

- Examples:

Case $3_2.2_1.1$
hanoi(1, C, A, B)
Count: 2
Source: C
Spare: A
Dest: B

```
def hanoi(count, source, dest
    if count <= 1:
        print ("base case: move disk from", source, "to", destination)
    else:
        hanoi(count - 1, source, spare, destination)
        print ("step2: move disk from", source, "to", destination)
        hanoi(count - 1, spare, destination, source)
```

base case: move disk from C to A



22.3 The Towers of Hanoi

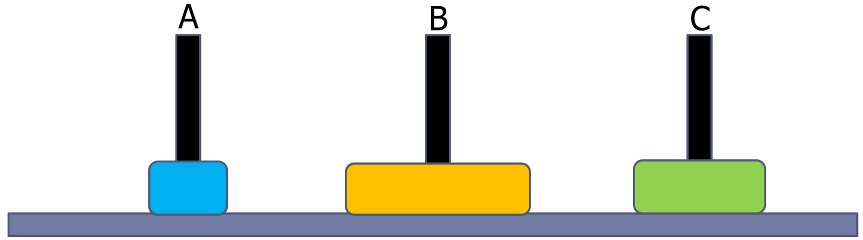
The Towers of Hanoi

- Examples:

Case $3_2.2$
hanoi(2, C, B, A)
Count: 2
Source: C
Spare: B
Dest: A

```
def hanoi(count, source, destination, spare):
    if count <= 1:
        print ("base case: move dis                destination)
    else:
        hanoi(count - 1, source
        print ("step2: move disk from", source, "to", destination)
        hanoi(count - 1, spare, destination, source)
```

step2: move disk from C to B



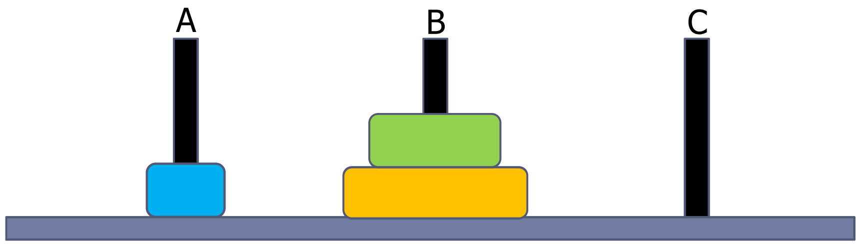
The Towers of Hanoi

- Examples:

Case $3_2.2_2.1$
hanoi(1, A, B, C)
Count: 1
Source: A
Spare: B
Dest: C

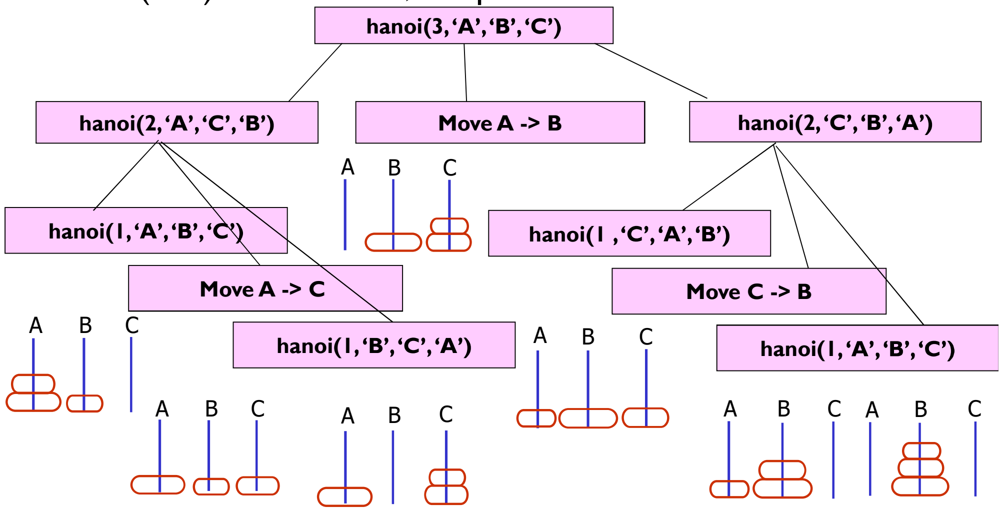
```
def hanoi(count, source, destination, spare):
    if count <= 1:
        print ("base case: move disk from", source, "to", destination)
    else:
        hanoi(count - 1, source, spare, destination)
        print ("step2: move disk from", source, "to", destination)
        hanoi(count - 1, spare, destination, source)
```

base case: move disk from A to B



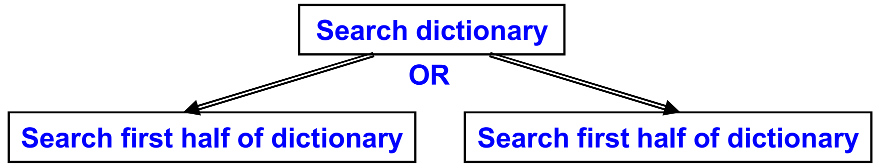
Call Tree

- hanoi(3…) uses 10 calls, a top-level one and 9 recursive calls



Binary Search

- Problem: look for an element (key) in an ordered collection (e.g. find a word in a dictionary)
- Sequential search
  - Starts at the beginning of the collection Looks at every item in the collection in order until the item being searched for is found
- Binary search
  - Cost?
  - Repeatedly halves the collection and determines which half could contain the item Uses a divide and conquer strategy



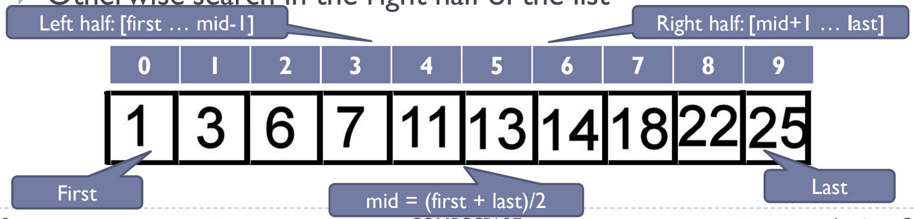
Binary Search

- Implementation issues:
  - How will you pass "half of list" to the recursive calls to binary_search?
  - How do you determine which half of the list contains value?
  - What should the base case(s) be?
  - How will binary_search indicate the result of the search?
- Example: a sorted list

1	3	6	7	11	13	14	18	22	25

Binary Search

- Base case:
  - If array is empty number is not in the list, or
  - If element is the one we look for return it
- Recursive call
  - Determine element in the middle
  - If the one we look for is smaller than element in the middle then search in the left half
  - Otherwise search in the right half of the list



Binary Search

- Code

```
def binary_search(num_list, first, last, value):
    index = 0
    if first > last:
        index = -1
    else:
        mid = (first + last) // 2
        if value == num_list[mid]:
            index = mid
        elif value < num_list[mid]:
            index = binary_search(num_list, first, mid-1, value)
        else:
            index = binary_search(num_list, mid+1, last, value)
    return index
```


Summary

- Understand and learn how to implement the recursive functions for different applications