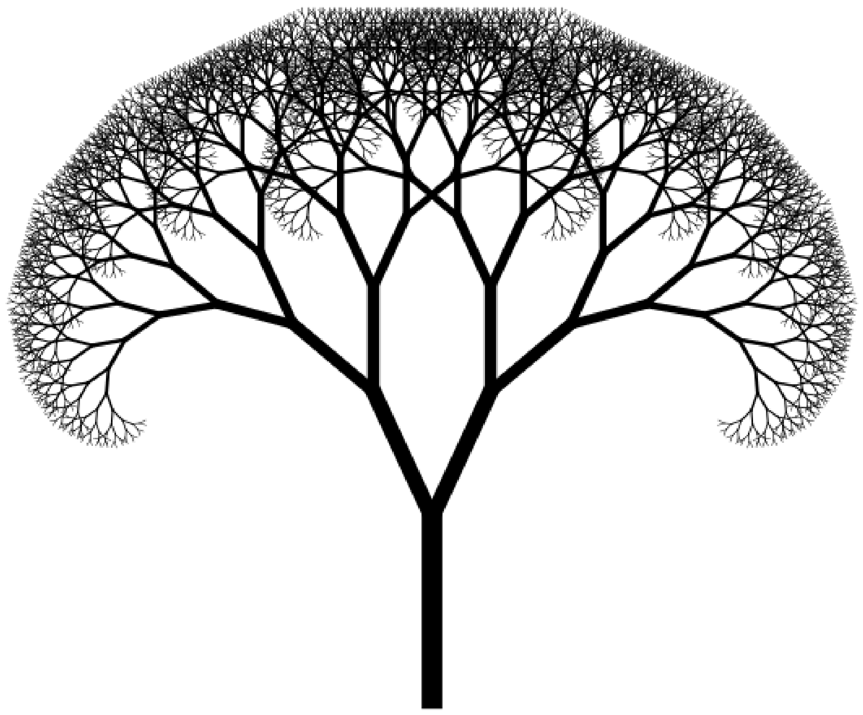
FIT1045: Algorithms and Programming Fundamentals in Python

Lecture 14

Recursion



Goal: Let Python solve the Towers of Hanoi Puzzle

In the great temple at Benares beneath the dome which marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four disks of pure gold, the largest disk resting on the brass plate and the others getting smaller and smaller up to the top one. This is the Tower of Brahma. Day and night unceasingly, the priest transfer the disks from one diamond needle to another according to the fixed and immutable laws of Brahma, which require that the priest on duty must not move more than one disk at a time and that he must place this disk on a needle so that there is no smaller disk below it. When all the sixty-four disks shall have been thus transferred from the needle on which at the creation God placed them to one of the other needles, tower, temple and Brahmins alike will crumble into dust, and with a thunderclap the world will vanish.

- Henri de Parville (La Nature)

Image at: mmvr41.blogspot.com/2005_01_01_archive.html

Objectives

Objectives of this lecture are to:

1. Get familiar with recursive functions
2. Practice to come up with "recurrence relations" that relate problem to simpler version of itself

This covers learning outcomes:

- 1 – Translate between problem descriptions and program designs with appropriate input/output representations
- 2 – Choose and implement appropriate problem solving strategies
- 5 – Decompose problems into simpler problems and reduce unknown to known problems

Overview

1. Recursive functions
2. Revisiting Mergesort
3. The Towers of Hanoi

Rabbit problem

Assume

- In month 1 we have single pair of baby rabbits (one male one female)
- Rabbits take 1 month to mature
- Every pair of adult rabbits gives birth to a male and female child (after 1 month of breeding)
- Rabbits live forever (ignore inbreeding)

How many rabbits do we have in month 4?

- 1st month: 2 (babies)
- 2nd month: 2 (adults)
- 3rd month: 4 (2 adults + 2 babies)
- 4th month: 6 (4 adults + 2 babies)
- …10, 16, 26 …

Quiz time (https://flux.qa)

Clayton:	AXXULH
Malaysia:	LWERDE

Rabbit problem

- Let R_n be the number of rabbits in month n.
- A_n for adults, B_n for babies
- $R_n = A_n + B_n$
- $A_n = A_{n-1} + B_{n-1} = R_{n-1}$
- $B_n = A_{n-1} = R_{n-2}$
- So $R_n = R_{n-1} + R_{n-2}$

This specifies the number of rabbits in each month (once we know how many we start with)

- 2, 2, 4, 6, 10, 16, 26 …
- Breeding pairs: 1, 1, 2, 3, 5, 8, 13 …

Worked example - Fibonacci

Definition

The Fibonacci numbers F_i are the sequence of numbers defined by

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$, for $n > 1$

0, 1, 1, 2, 3, 5, 8, 13, 21, 24 …

Fibonacci recurrence

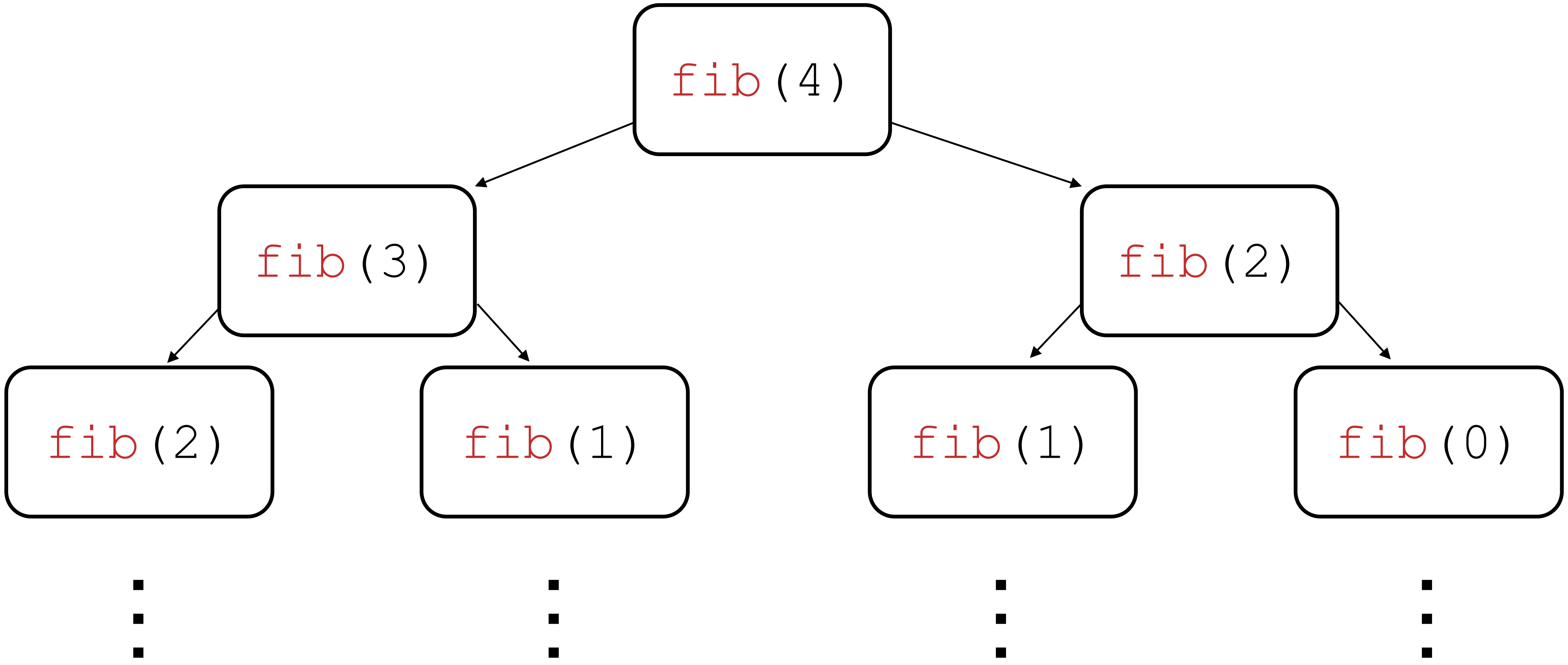
$F_n = F_{n-1} + F_{n-2}$, for $n > 1$

Recurrence relation

- Suppose we had a function `fib(n)` which could compute Fibonacci numbers up to a given input size `k-1`
- How could we use this function to compute `fib(k)`?
- `fib(k)` = `fib(k-1)` + `fib(k-2)`
- Note how this is almost identical to the mathematical definition

Fibonacci calls

```
def fib(n):
    return fib(n-1) + fib(n-2)
```


$$\begin{tikzcd}
 & & \text{fib(4)} \arrow[dll] \arrow[drr] & & \\
 & \text{fib(3)} \arrow[dl] \arrow[d] & & & \text{fib(2)} \arrow[dl] \arrow[d] \\
\text{fib(2)} \arrow[d, phantom, "\vdots"] & \text{fib(1)} \arrow[d, phantom, "\vdots"] & & \text{fib(1)} \arrow[d, phantom, "\vdots"] & \text{fib(0)} \arrow[d, phantom, "\vdots"] \\
 & & & &
\end{tikzcd}$$

Where does it end?

What base cases do we need?

A. F_0

B. F_1

C. F_0 and F_1

D. F_0 or F_1

Quiz time (https://flux.qa)

Clayton:	AXXULH
Malaysia:	LWERDE

Base case

Definition

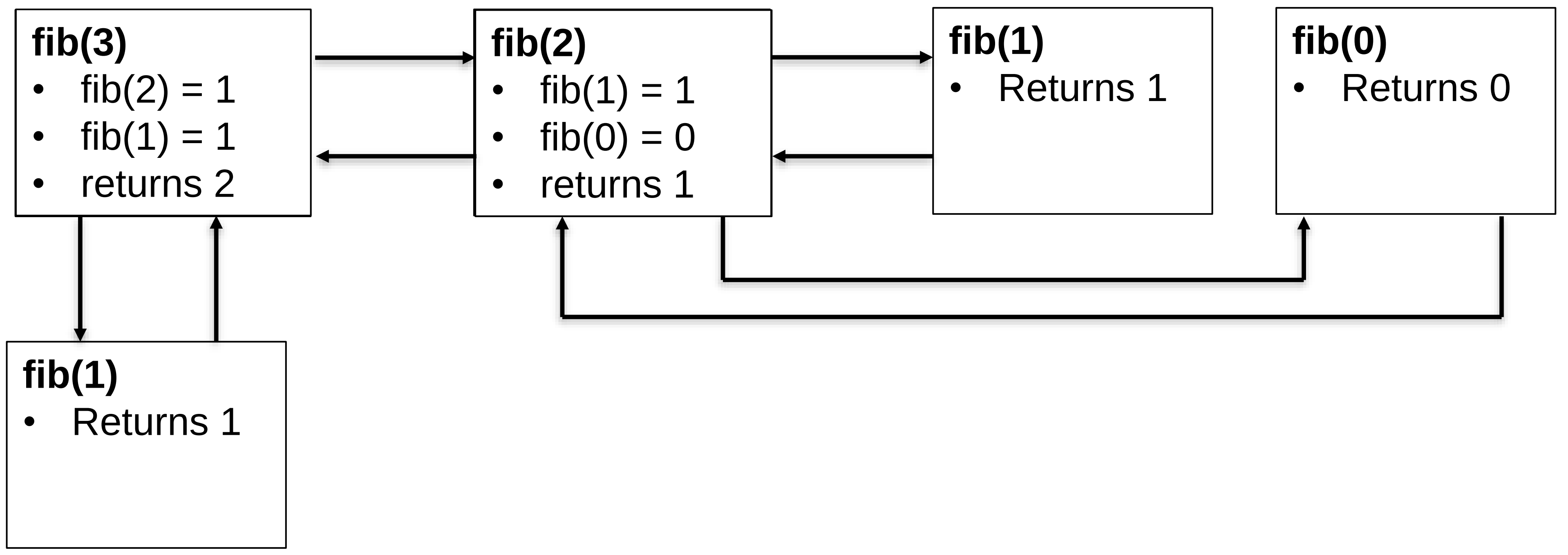
The Fibonacci numbers F_i are a sequence of numbers with

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}, when\ n > 1$

```
def fib(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fib(n-1) + fib(n-2)
```

Fibonacci calls

```
def fib(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fib(n-1) + fib(n-2)
```



What does this do?

```
def something(lst):
    n = len(lst)
    if n == 0:
        return 0
    else:
        return lst[0] + something(lst[1:])
```

What is the value of `something([10, 20, 30])`?

A. 30
B. 10
C. 40
D. 60
E. None of the above

Quiz time (https://flux.qa)

Clayton:	AXXULH
Malaysia:	LWERDE

Recursive sum

```
def sum(lst):
    n = len(lst)
    if n == 0:
        return 0
    else:
        return lst[0] + sum(lst[1:])
```

`sum(lst)=0` if `len(lst)==0`
(base case)

`sum(lst)=lst[0]+lst[1:]`
if `len(lst)>0`
(recurrence relation)

Definition

A recursive function is a function that contains one or more subcalls to itself inside its function body.

For termination need subcalls with *simpler* input until input with no subcalls to self is reached. These inputs are called base case.

Conceptually

Reduce problem to *simpler* version of itself (cf. decomposition)

Recall: decomposition…

…can be thought of from **two perspectives**:

1. Breaking down programs into sub-programs (components)
2. Breaking down problems into sub-problems

…is ***most useful if two views coincide***, i.e., sub-programs correspond to sub-problems

- structures thinking/attention for developing algorithms and *reasoning* about programs
- leads to *re-usable components* (because they solve a well-defined problem)

When breaking down problem into *simpler form of itself*, use recursive function method to break down program

Recursive problem solving

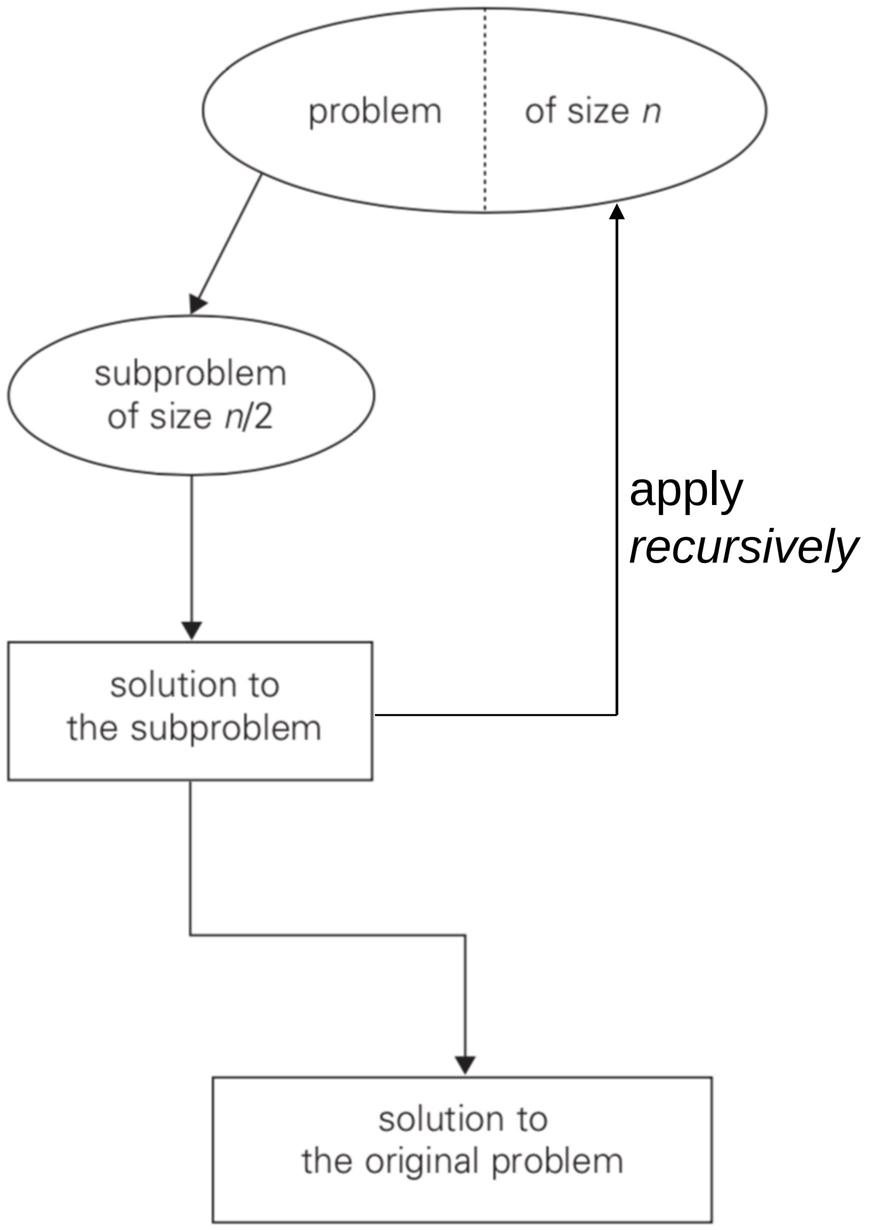


illustration: [Levitin, p. 133]

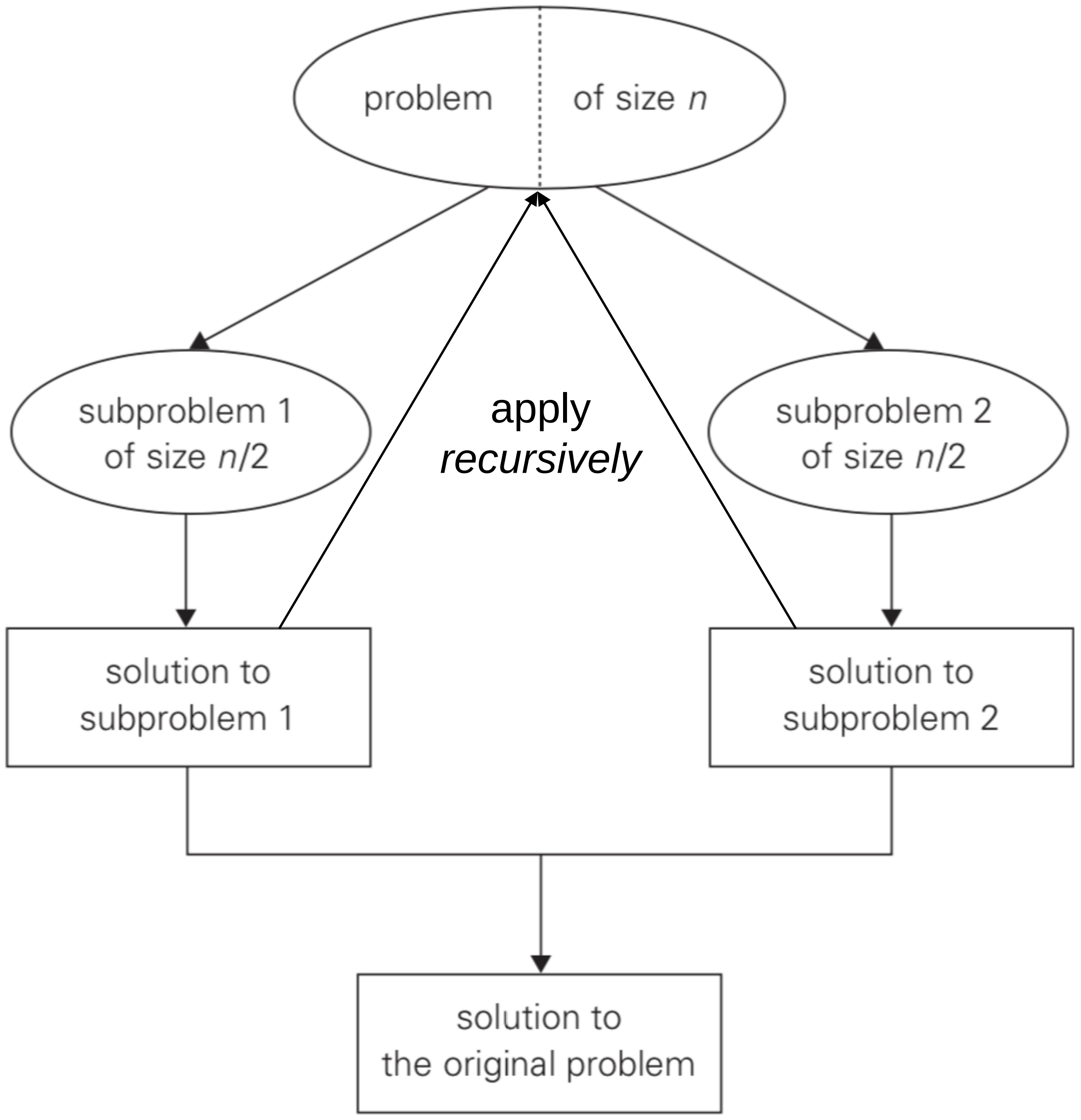


illustration: [Levitin, p. 170]

Overview

1. Recursive functions and problem solving
2. Revisiting Mergesort
3. The Towers of Hanoi

Example: Mergesort

b	f	h	a	d	k	g	j	e	c	i
0	1	2	3	4	5	6	7	8	9	10

consider as two separate
sorting problems

b	f	h	a	d	k
0	1	2	3	4	5

g	j	e	c	i
6	7	8	9	10

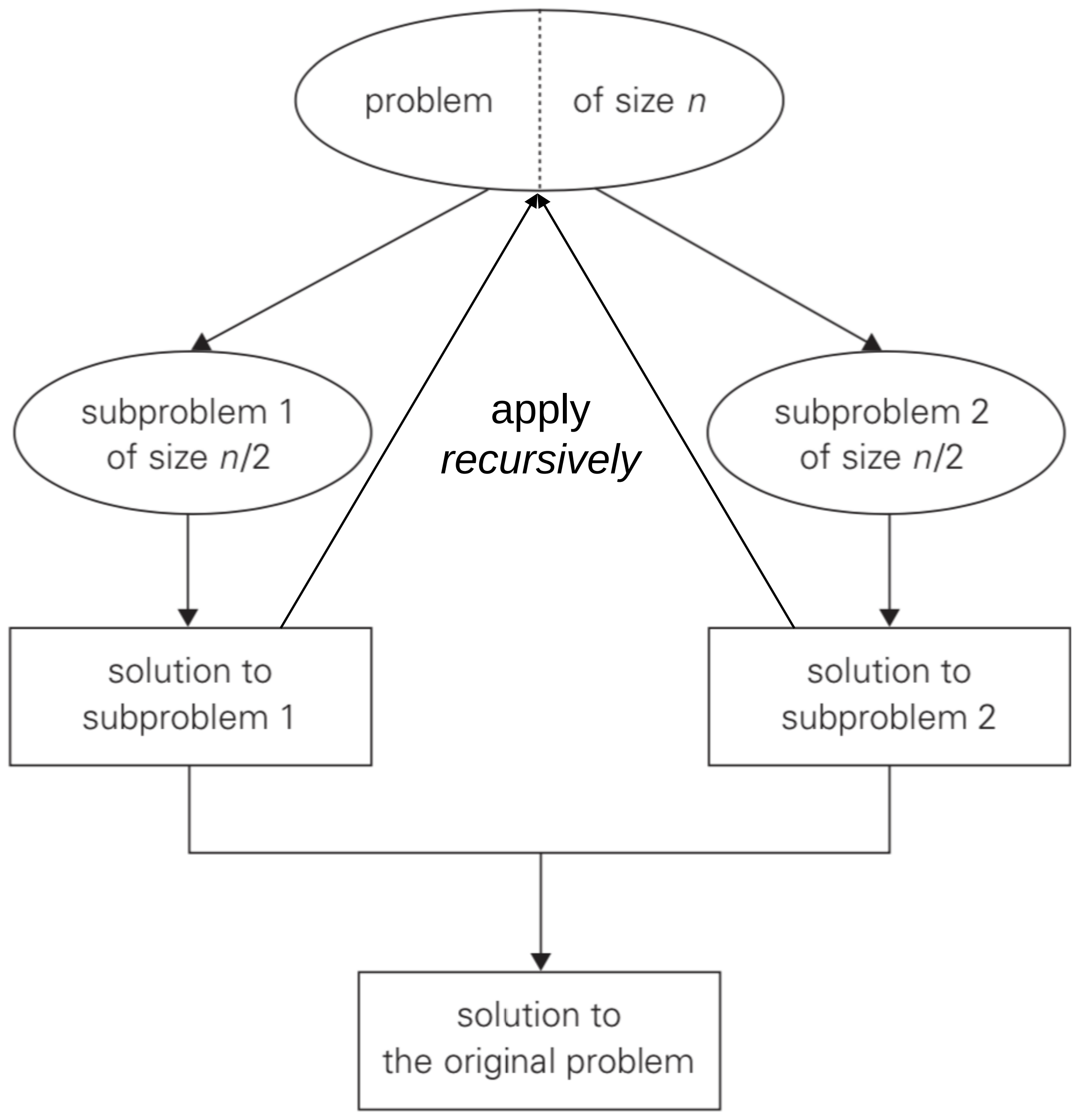
solve independently

a	b	d	f	h	k
0	1	2	3	4	5

c	e	g	j	i
6	7	8	9	10

merge solutions

a	b	c	d	e	f	g	h	i	j	k
0	1	2	3	4	5	6	7	8	9	10



Our implementation of Merge Sort

```python
def mergesort(ls):
    k, n = 1, len(ls)
    while k < n:
        nxt = []
        for a in range(0, n, 2*k):
            b, c = a + k, a + 2*k
            nxt += merge(ls[a:b], ls[b:c])
        ls = nxt
        k = 2 * k
    return ls
```

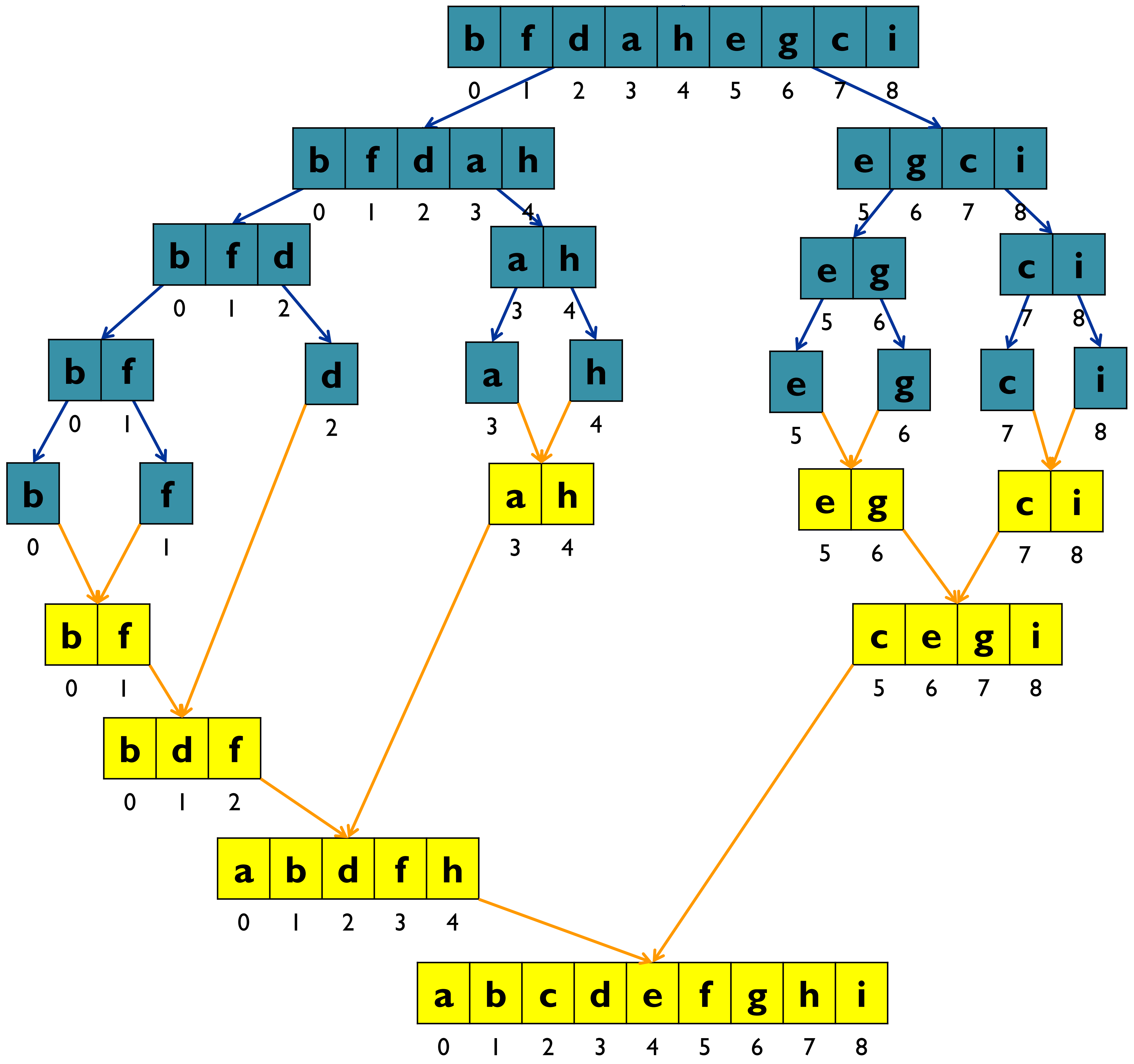
Problems:

- No explicit representation of subproblems
- Lots of variable re-assignment/object mutation
- Hard to see correctness (or that it implements algorithms)

Let’s rewrite as recursive function

```python
def mergesort(ls):
    k, n = 1, len(ls)
    while k < n:
        nxt = []
        for a in range(0, n, 2*k):
            b, c = a + k, a + 2*k
            nxt += merge(ls[a:b], ls[b:c])
        ls = nxt
        k = 2 * k
    return ls
```

```python
def mergesort(ls):
    n = len(ls)
    if n <= 1:
        return ls
    else:
        sub1 = mergesort(ls[:n//2])
        sub2 = mergesort(ls[n//2:])
        return merge(sub1, sub2)
```



b f d a h e g c i
0 1 2 3 4 5 6 7 8
b f d a h
e g c i
b f d
a h
e g
c i
b f
d
a
h
e
g
c
i
b
f
a h
e g
c i
b f
c e g i
b d f
a b d f h
a b c d e f g h i
0 1 2 3 4 5 6 7 8

Properties of recursive implementation

- One-to-one mapping of subproblems to function calls
- No local variables needed to represent subproblem as part of global problem (calls isolate scope of subproblem)
- No re-assignment/mutation
- Easy to see correctness based on *recurrence relation* of problem solutions:

  sorted(lst)=merged(sorted(lst[:n//2]), sorted(lst[n//2:]))

```
def merge_sort(ls):
    n = len(ls)
    if n <= 1:
        return ls
    else:
        sub1 = mergesort(ls[:n//2])
        sub2 = mergesort(ls[n//2:])
        return merge(sub1, sub2)
```

Overview

1. Recursive functions and problem solving
2. Revisiting Mergesort
3. The Towers of Hanoi

Towers of Hanoi

In the great temple at Benares beneath the dome which marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four disks of pure gold, the largest disk resting on the brass plate and the others getting smaller and smaller up to the top one. This is the Tower of Brahma. Day and night unceasingly, the priest transfer the disks from one diamond needle to another according to the fixed and immutable laws of Brahma, which require that the priest on duty must not move more than one disk at a time and that he must place this disk on a needle so that there is no smaller disk below it. When all the sixty-four disks shall have been thus transferred from the needle on which at the creation God placed them to one of the other needles, tower, temple and Brahmins alike will crumble into dust, and with a thunderclap the world will vanish.

- Henri de Parville (La Nature)

Image at: mmvr41.blogspot.com/2005_01_01_archive.html

Let's extract the rules of puzzle

In the great temple at Benares beneath the dome which marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four disks of pure gold, the largest disk resting on the brass plate and the others getting smaller and smaller up to the top one. This is the Tower of Brahma. Day and night unceasingly, the priest transfer the disks from one diamond needle to another according to the fixed and immutable laws of Brahma, which require that the priest on duty must not move more than one disk at a time and that he must place this disk on a needle so that there is no smaller disk below it. When all the sixty-four disks shall have been thus transferred from the needle on which at the creation God placed them to one of the other needles, tower, temple and Brahmins alike will crumble into dust, and with a thunderclap the world will vanish.

Let’s extract the rules of puzzle

In the great temple at Benares beneath the dome which marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four disks of pure gold, the largest disk resting on the brass plate and the others getting smaller and smaller up to the top one. This is the Tower of Brahma. Day and night unceasingly, the priest transfer the disks from one diamond needle to another according to the fixed and immutable laws of Brahma, which require that the priest on duty must not move more than one disk at a time and that he must place this disk on a needle so that there is no smaller disk below it. When all the sixty-four disks shall have been thus transferred from the needle on which at the creation God placed them to one of the other needles, tower, temple and Brahmins alike will crumble into dust, and with a thunderclap the world will vanish.

1. Three needles, one containing *n* disks (smaller towards top)

Let's extract the rules of puzzle

In the great temple at Benares beneath the dome which marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four disks of pure gold, the largest disk resting on the brass plate and the others getting smaller and smaller up to the top one. This is the Tower of Brahma. Day and night unceasingly, the priest transfer the disks from one diamond needle to another according to the fixed and immutable laws of Brahma, which require that the priest on duty must not move more than one disk at a time and that he must place this disk on a needle so that there is no smaller disk below it. When all the sixty-four disks shall have been thus transferred from the needle on which at the creation God placed them to one of the other needles, tower, temple and Brahmins alike will crumble into dust, and with a thunderclap the world will vanish.

1. Three needles, one containing *n* disks (smaller towards top)
2. Must not move more than one disk at a time

Let's extract the rules of puzzle

In the great temple at Benares beneath the dome which marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four disks of pure gold, the largest disk resting on the brass plate and the others getting smaller and smaller up to the top one. This is the Tower of Brahma. Day and night unceasingly, the priest transfer the disks from one diamond needle to another according to the fixed and immutable laws of Brahma, which require that the priest on duty must not move more than one disk at a time and that he must place this disk on a needle so that there is no smaller disk below it. When all the sixty-four disks shall have been thus transferred from the needle on which at the creation God placed them to one of the other needles, tower, temple and Brahmins alike will crumble into dust, and with a thunderclap the world will vanish.

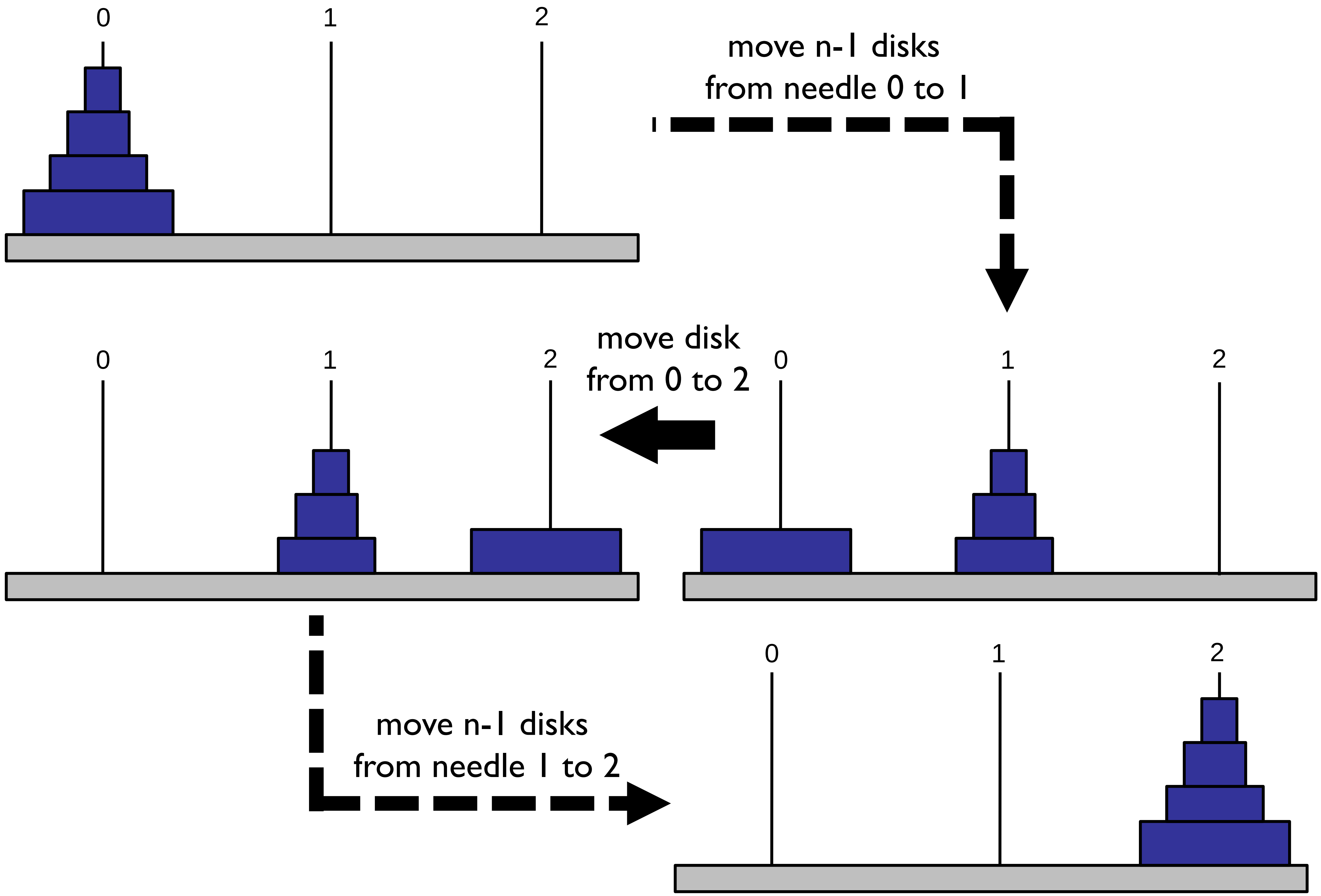
1. Three needles, one containing *n* disks (smaller towards top)
2. Must not move more than one disk at a time
3. Must place disk on needle so that no smaller disk below

Let's extract the rules of puzzle

In the great temple at Benares beneath the dome which marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four disks of pure gold, the largest disk resting on the brass plate and the others getting smaller and smaller up to the top one. This is the Tower of Brahma. Day and night unceasingly, the priest transfer the disks from one diamond needle to another according to the fixed and immutable laws of Brahma, which require that the priest on duty must not move more than one disk at a time and that he must place this disk on a needle so that there is no smaller disk below it. When all the sixty-four disks shall have been thus transferred from the needle on which at the creation God placed them to one of the other needles, tower, temple and Brahmins alike will crumble into dust, and with a thunderclap the world will vanish.

1. Three needles, one containing *n* disks (smaller towards top)
2. Must not move more than one disk at a time
3. Must place disk on needle so that no smaller disk below
4. Must move all disks to another needle (to end game)

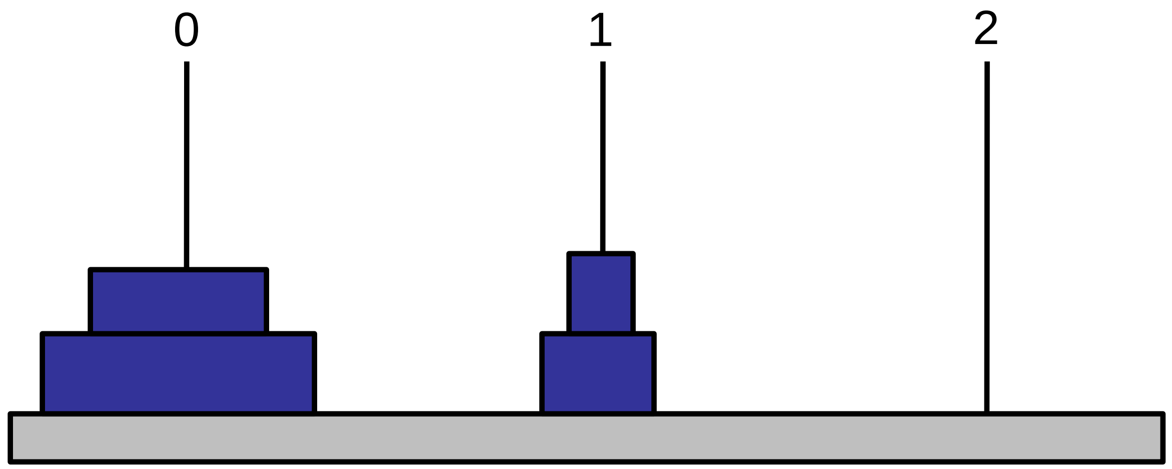
How to move *n* disks from needle *0* to *2*?



Representation in Python

Puzzle components:

disk	positive integer corresponding to size of disk
needle	list of integers, representing disks from bottom to top
state	list of 3 lists, representing needles from left to right



```
[[4,3], [2,1], []]
```

Operations:

```
>>> start_state(4)
[[4, 3, 2, 1], [], []]
>>> next_state(1, 2, [[4,3], [2,1], []])
[[4,3], [2], [1]]
>>> aux(0,2), aux(1,2), aux(0,1)
(1, 0, 2)
```

Python Implementation

```
def hanoi(n, i, j, state):
    """
    I: num n of disks to move from needle i to needle j
    O: intermediate states when carrying out input task
    """
    if n == 1:
        return [(next_state(i, j, state))]
    else:
        sub1 = hanoi(n - 1, i, aux(i, j), state)
        nxt = next_state(i, j, sub1[-1])
        sub2 = hanoi(n - 1, aux(i, j), j, nxt)
    return sub1 + [nxt] + sub2
```

Why correct? Does algorithm obey rules?

```
>>> hanoi(4, 0, 2, start_state(4))
[[[4, 3, 2, 1], [], []], [[4, 3, 2], [1], []]
 [[4, 3], [1], [2]], [[4, 3], [], [2, 1]],
                              …
 [[], [1], [4, 3, 2]], [[], [], [4, 3, 2, 1]]]
```

Python Implementation

```
def hanoi(n, i, j, state):
    """
    I: num n of disks to move from needle i to needle j
       in state such that state[i][-n]<state[j][-1]
       and state[i][-n]<state[aux(i,j)][-1]
    O: intermediate states when carrying out input task
    """
    if n == 1:
        return [(next_state(i, j, state))]
    else:
        sub1 = hanoi(n - 1, i, aux(i, j), state)
        nxt = next_state(i, j, sub1[-1])
        sub2 = hanoi(n - 1, aux(i, j), j, nxt)
    return sub1 + [nxt] + sub2
```

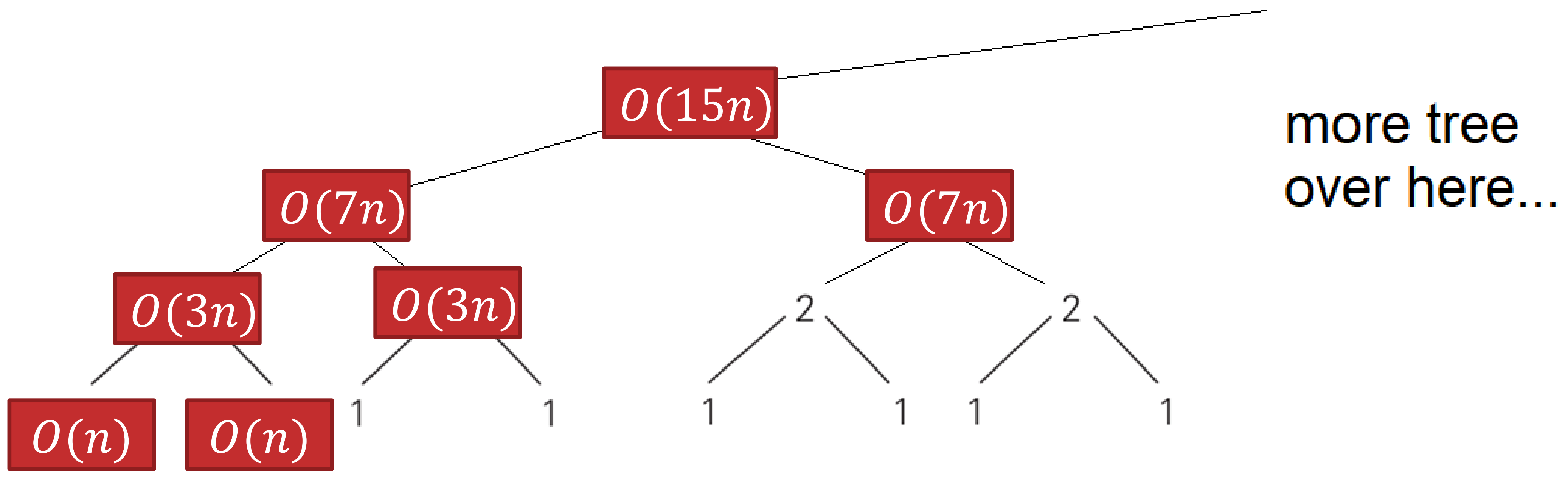
assuming subcalls don't violate rules, this level does not violate rules

Computational complexity of hanoi?

```
def hanoi(n, i, j, state):
    """
    I: num n of disks to move from needle i to needle j
       in state such that state[i][-n]<state[j][-1]
       and state[i][-n]<state[aux(i,j)][-1]
    O: intermediate states when carrying out input task
    """
    if n == 1: ──────────────────────────────────→ O(1)
        return [(next_state(i, j, state))] ──────→ O(n)
    else:
        sub1 = hanoi(n - 1, i, aux(i, j), state) ─→ ?
        nxt = next_state(i, j, sub1[-1]) ────────→ O(n)
        sub2 = hanoi(n - 1, aux(i, j), j, nxt) ──→ ?
    return sub1 + [nxt] + sub2
```

How to account for recursive subcalls?

Time complexity of hanoi



- The calls at the bottom take $O(n)$
- At level i the time taken will be $O(n\ * (2^{i+1} - 1))$
- Which is $O(n * 2^i)$ since constants don't matter
- Top level $i = n \Rightarrow O(n * 2^n)$
- ~600 billion years at 1 disk per second (n=64)

Summary

Recursive functions are a natural way to implement recursive solution strategies (divide-and-conquer, decrease-and-conquer)

Usually easier to reason about correctness

- just need to understand I/O specification and recurrence relation of solutions
- no need for loop invariants; motivation to re-write even iterative ideas recursively

Often harder to see computational complexity; motivation to rewrite iteratively (next week)

Coming Up

- Graph traversals
- Stacks and Queues