COMP108
Algorithmic Foundations

Divide and Conquer

Prudence Wong

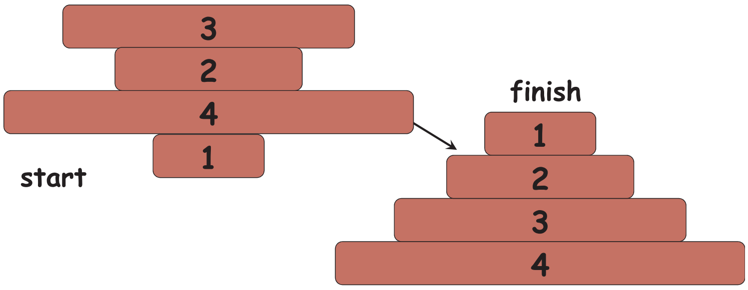
http://www.csc.liv.ac.uk/~pwong/teaching/comp108/201617

Pancake Sorting

Input: Stack of pancakes, each of **different** sizes

Output: Arrange in **order of size** (smallest on top)

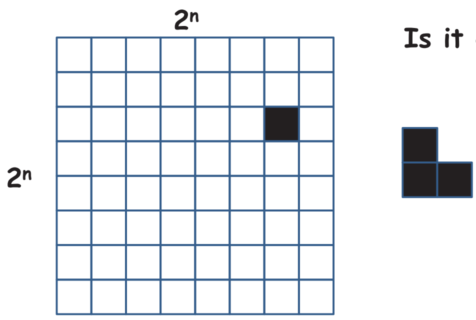
Action: Slip a **flipper** under one of the pancakes and **flip** over the **whole stack** above the flipper



Triomino Puzzle

Input: 2^n-by-2^n chessboard with **one** missing square & many **L-shaped** tiles of **3** adjacent squares

Question: **Cover** the chessboard with L-shaped tiles without overlapping



Is it do-able?

Divide and Conquer ...

Learning outcomes

- Understand how divide and conquer works and able to analyse complexity of divide and conquer methods by solving recurrence
- See examples of divide and conquer methods

Divide and Conquer

One of the **best-known** algorithm design techniques

Idea:

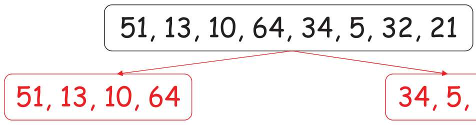
- A problem instance is ***divided*** into several **smaller** instances of the same problem, ideally of about same size
- The smaller instances are **solved**, typically **recursively**
- The solutions for the smaller instances are ***combined*** to get a solution to the large instance

Merge Sort ...

Merge sort

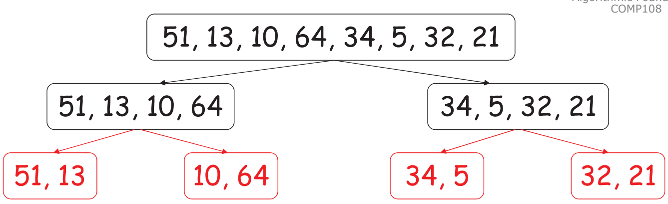
- using divide and conquer technique
- divide the sequence of n numbers into two halves
- **recursively** sort the two halves
- **merge** the two sorted halves into a single sorted sequence

we want to sort these 8 numbers,
divide them into two halves

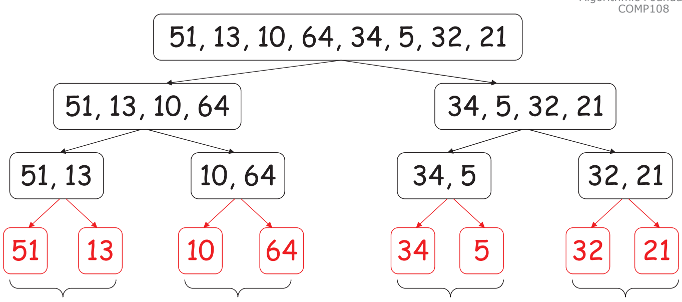


divide these 4 numbers into halves

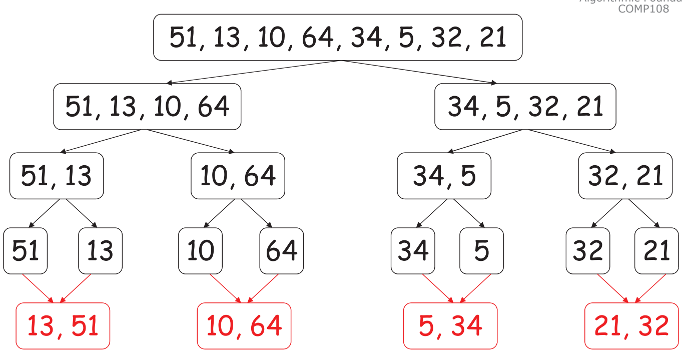
similarly for these 4



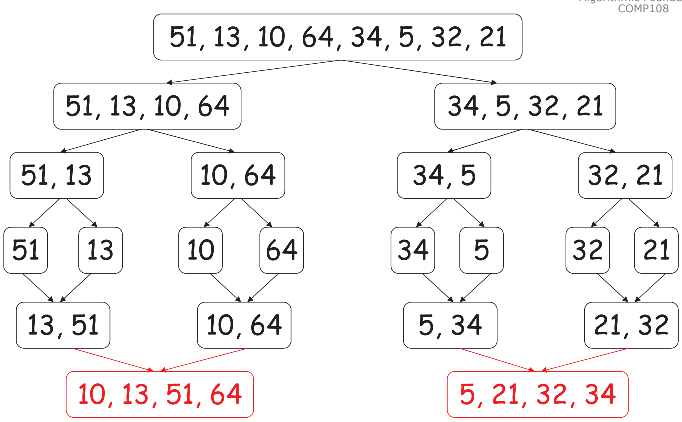
further divide each shorter sequence …
until we get sequence with only **1** number



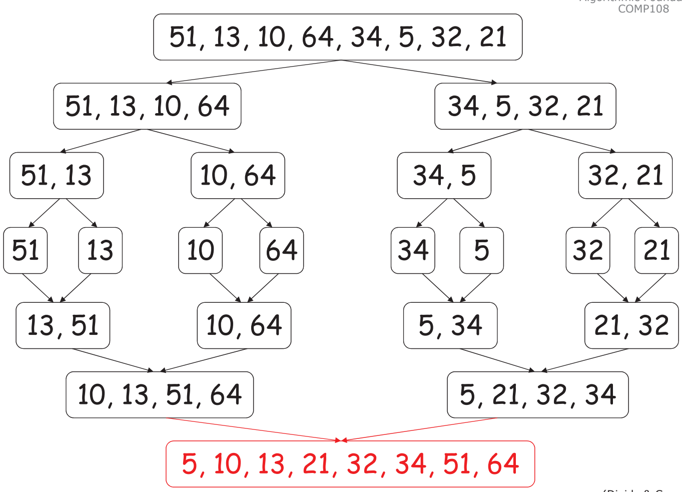
merge pairs of single number into a sequence of 2 sorted numbers



then **merge** again into sequences of
4 sorted numbers



one more merge give the **final** sorted sequence



Summary

Divide

- dividing a sequence of n numbers into **two** smaller sequences is straightforward

Conquer

- merging two sorted sequences of **total length n** can also be done easily, at most **n-1** comparisons

10, 13, 51, 64

5, 21, 32, 34

Result:

To merge two sorted sequences,
we keep two **pointers**, one to each sequence

Compare the two numbers pointed,
copy the **smaller** one to the result
and **advance** the corresponding pointer

10, 13, 51, 64

5, 21, 32, 34

Result: **5**,

Then compare again the two numbers
pointed to by the pointer;
copy the smaller one to the result
and advance that pointer

10, 13, 51, 64

5, 21, 32, 34

Result: 5, **10**,

Repeat the same process …

10, 13, 51, 64

5, 21, 32, 34

Result: 5, 10, **13**

Again …

10, 13, 51, 64

5, 21, 32, 34

Result: 5, 10, 13, **21**

and again …

10, 13, 51, 64

5, 21, 32, 34

Result: 5, 10, 13, 21, **32**

…

10, 13, 51, 64

5, 21, 32, 34

Result: 5, 10, 13, 21, 32, **34**

When we reach the **end** of one sequence,
simply copy the **remaining** numbers in the other
sequence to the result

10, 13, 51, 64

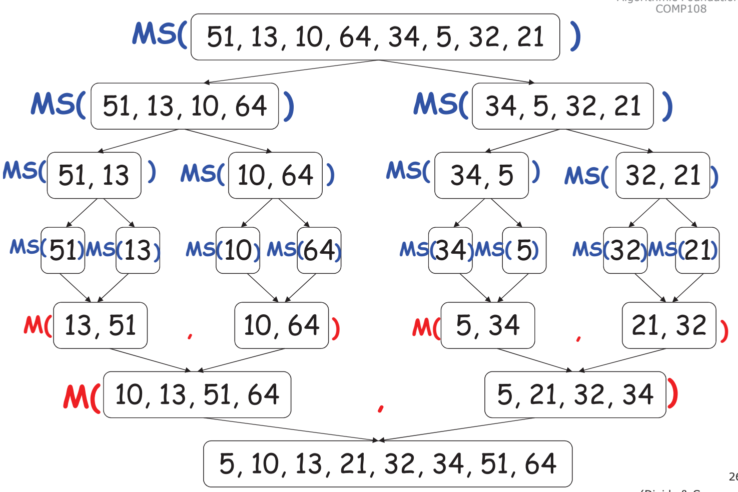
5, 21, 32, 34

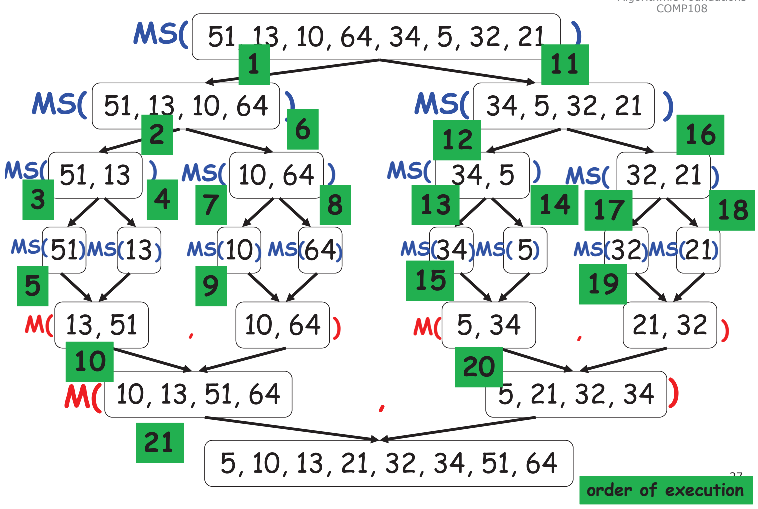
Result: 5, 10, 13, 21, 32, 34, **51, 64**

Then we obtain the final sorted sequence

Pseudo code

```
Algorithm Mergesort(A[1..n])
    if n > 1 then begin
        copy A[1..⌊n/2⌋] to B[1..⌊n/2⌋]
        copy A[⌊n/2⌋+1..n] to C[1..⌈n/2⌉]
        Mergesort(B[1..⌊n/2⌋])
        Mergesort(C[1..⌈n/2⌉])
        Merge(B, C, A)
    end
```



Pseudo code

```
Algorithm Merge(B[1..p], C[1..q], A[1..p+q])
  set i=1, j=1, k=1
  while i<=p and j<=q do
  begin
    if B[i]≤C[j] then
      set A[k] = B[i] and i = i+1
    else set A[k] = C[j] and j = j+1
    k = k+1
  end
  if i==p+1 then copy C[j..q] to A[k..(p+q)]
  else copy B[i..p] to A[k..(p+q)]
```


p=4 q=4

B: 10, 13, 51, 64 C: 5, 21, 32, 34

	i	j	k	A[ ]
Before loop	1	1	1	empty
End of 1st iteration	1	2	2	5
End of 2nd iteration	2	2	3	5, 10
End of 3rd	3	2	4	5, 10, 13
End of 4th	3	3	5	5, 10, 13, 21
End of 5th	3	4	6	5, 10, 13, 21, 32
End of 6th	3	5	7	5, 10, 13, 21, 32, 34
				5, 10, 13, 21, 32, 34, 51, 64

Time complexity

Let T(n) denote the time complexity of running merge sort on n numbers.

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2\times T(\frac{n}{2}) + n & \text{otherwise} \end{cases}$$

We call this formula a *recurrence*.

A recurrence is an equation or inequality that describes a function in terms of ***its value on smaller inputs***.

To **solve** a recurrence is to derive ***asymptotic bounds*** on the solution

Time complexity

Prove that $T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2\times T(\frac{n}{2}) + n & \text{otherwise} \end{cases}$ is O(n log n)

Make a guess: T(n) ≤ 2 n log n (We prove by MI)

For the base case when n=2,
L.H.S = T(2) = 2×T(1) + 2 = 4,
R.H.S = 2 × 2 log 2 = 4
L.H.S ≤ R.H.S

Substitution method

Time complexity

Prove that $T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2\times T(\frac{n}{2}) + n & \text{otherwise} \end{cases}$ is O(n log n)

Make a guess: T(n) ≤ 2 n log n (We prove by MI)

Assume true for all n'<n [assume $T(\frac{n}{2}) \le 2 \times (\frac{n}{2}) \times \log(\frac{n}{2})$]

$T(n) = \underbrace{2\times T(\frac{n}{2})}_{}+n$ **by hypothesis**

$\le 2 \times (2\times(\frac{n}{2})\times\log(\frac{n}{2})) + n$

$= 2\,n\,(\log n - 1) + n$

$= 2\,n \log n - 2n + n$ $\boxed{\text{i.e., } T(n) \le 2\,n \log n}$

$\le 2\,n \log n$

Example

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(\frac{n}{2}) + 1 & \text{otherwise} \end{cases}$$

Guess: $T(n) \le 2 \log n$

For the base case when n=2,
L.H.S = T(2) = T(1) + 1 = 2
R.H.S = 2 log 2 = 2
L.H.S ≤ R.H.S

Example

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(\frac{n}{2}) + 1 & \text{otherwise} \end{cases}$$

Guess: $T(n) \le 2 \log n$

Assume true for all n' < n [assume $T(\frac{n}{2}) \le 2 \times \log(\frac{n}{2})$]

$T(n) = T(\frac{n}{2}) + 1$

$\le 2 \times \log(\frac{n}{2}) + 1$ ← *by hypothesis*

$= 2x(\log n - 1) + 1$ ← $\log(\frac{n}{2}) = \log n - \log 2$

$< 2\log n$

$\boxed{\text{i.e., } T(n) \le 2 \log n}$

More example

Prove that $T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2 \times T(\frac{n}{2}) + 1 & \text{otherwise} \end{cases}$ is O(n)

Guess: $T(n) \le 2n - 1$

For the base case when n=1,
L.H.S = T(1) = 1
R.H.S = 2×1 - 1 = 1
L.H.S ≤ R.H.S

More example

Prove that $T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2 \times T(\frac{n}{2}) + 1 & \text{otherwise} \end{cases}$ is O(n)

Guess: $T(n) \le 2n - 1$

Assume true for all n' < n [assume $T(\frac{n}{2}) \le 2\times(\frac{n}{2})-1$]

$T(n) = 2\times T(\frac{n}{2})+1$

$\le 2 \times (2\times(\frac{n}{2})-1) + 1$ ← *by hypothesis*

$= 2n - 2 + 1$

$= 2n - 1$

$\boxed{\text{i.e., } T(n) \le 2n-1}$

Summary

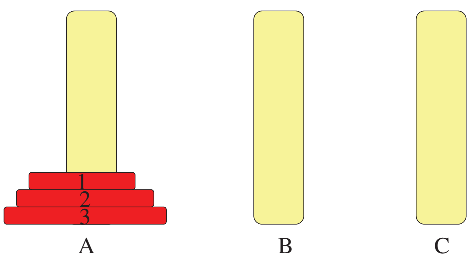
Depending on the recurrence, we can guess the order of growth

$T(n) = T(\frac{n}{2})+1$	T(n) is O(log n)
$T(n) = 2\times T(\frac{n}{2})+1$	T(n) is O(n)
$T(n) = 2\times T(\frac{n}{2})+n$	T(n) is O(n log n)

Tower of Hanoi ...

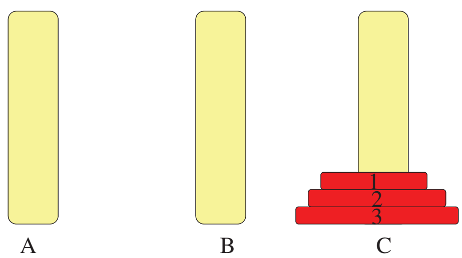
Tower of Hanoi - Initial config

There are three pegs and some discs of different sizes are on Peg A



Tower of Hanoi - Final config

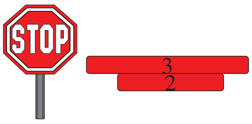
Want to move the discs to Peg C



Tower of Hanoi - Rules

Only 1 disk can be moved at a time

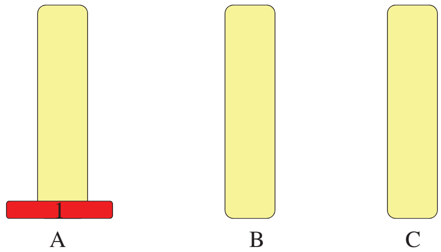
A disc cannot be placed on top of other discs that are smaller than it



Target: Use the smallest number of moves

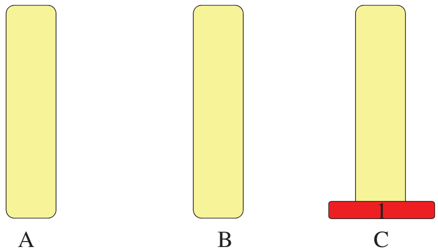
Tower of Hanoi - One disc only

Easy!



Tower of Hanoi - One disc only

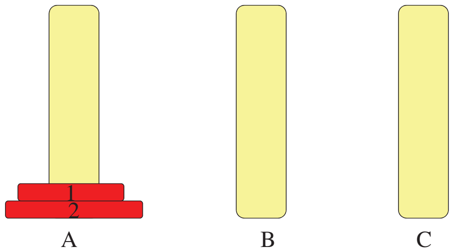
Easy! Need one move only.



Tower of Hanoi - Two discs

We first need to move Disc-2 to C, How?

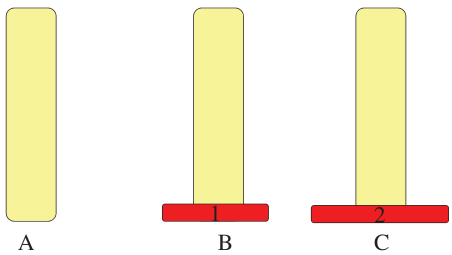
by moving Disc-1 to B first, then Disc-2 to C



Tower of Hanoi - Two discs

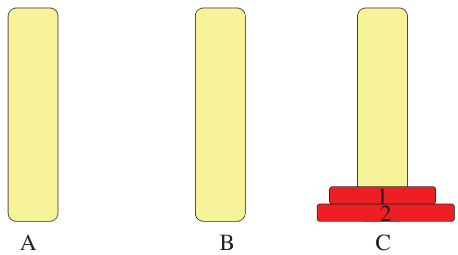
Next?

Move Disc-1 to C



Tower of Hanoi - Two discs

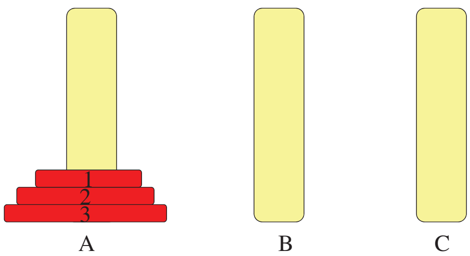
Done!



Tower of Hanoi - Three discs

We first need to move Disc-3 to C, How?

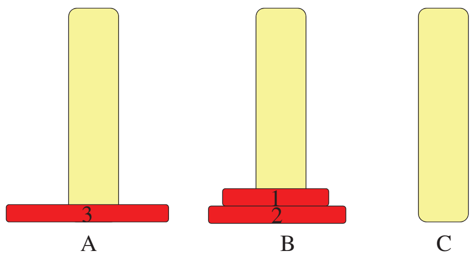
- Move Disc-1&2 to B (recursively)



Tower of Hanoi - Three discs

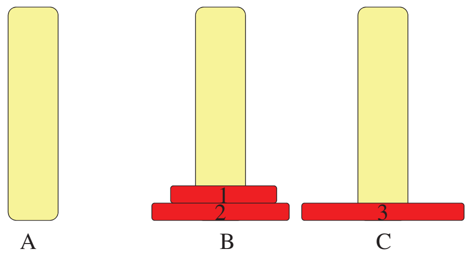
We first need to move Disc-3 to C, How?

- Move Disc-1&2 to B (recursively)
- Then move Disc-3 to C



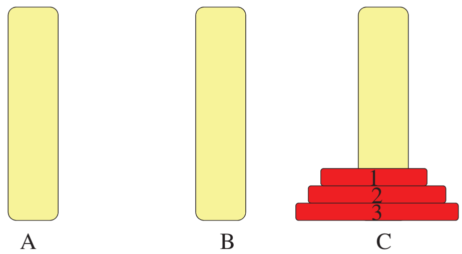
Tower of Hanoi - Three discs

Only task left: move Disc-1&2 to C (similarly as before)



Tower of Hanoi - Three discs

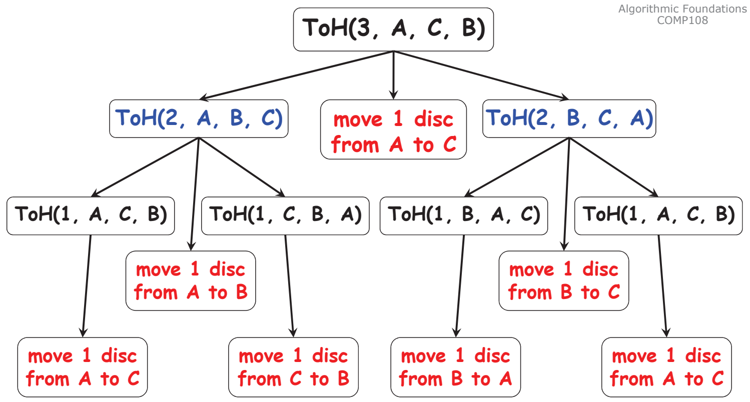
Done!

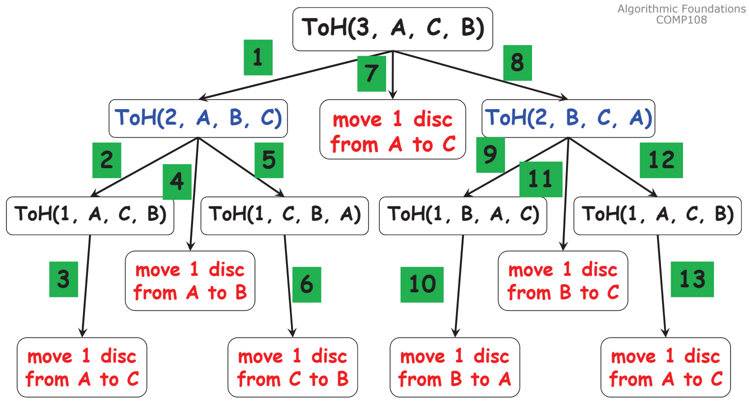


Tower of Hanoi

invoke by calling ToH(3, A, C, B)

```
ToH(num_disc, source, dest, spare)
begin
   if (num_disc > 1) then
       ToH(num_disc-1, source, spare, dest)
   Move the disc from source to dest
   if (num_disc > 1) then
       ToH(num_disc-1, spare, dest, source)
end
```



from A to C; from A to B; from C to B;
from A to C;
from B to A; from B to C; from A to C;

Time complexity

Let T(n) denote the time complexity of running the Tower of Hanoi algorithm on n discs.

$$T(n) = \underbrace{T(n-1)}_{\text{move n-1 discs from A to B}} + \underbrace{1}_{\text{move Disc-n from A to C}} + \underbrace{T(n-1)}_{\text{move n-1 discs from B to C}}$$

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2\times T(n-1) + 1 & \text{otherwise} \end{cases}$$

Time complexity (2)

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2\times T(n-1) + 1 & \text{otherwise} \end{cases}$$

$$\begin{aligned} T(n) &= 2\times T(n-1) + 1 \\ &= 2[2\times T(n-2) + 1] + 1 \\ &= 2^2\, T(n-2) + 2 + 1 \\ &= 2^2\, [2\times T(n-3) + 1] + 2^1 + 2^0 \\ &= 2^3\, T(n-3) + 2^2 + 2^1 + 2^0 \\ &\dots \\ &= 2^k\, T(n-k) + 2^{k-1} + 2^{k-2} + \dots + 2^2 + 2^1 + 2^0 \\ &\dots \\ &= 2^{n-1}\, T(1) + 2^{n-2} + 2^{n-3} + \dots + 2^2 + 2^1 + 2^0 \\ &= 2^{n-1} + 2^{n-2} + 2^{n-3} + \dots + 2^2 + 2^1 + 2^0 \\ &= 2^n - 1 \end{aligned}$$

iterative method

i.e., T(n) is $O(2^n)$

In Tutorial 2, we prove by MI that $2^0 + 2^1 + \dots + 2^{n-1} = 2^n - 1$

Summary - continued

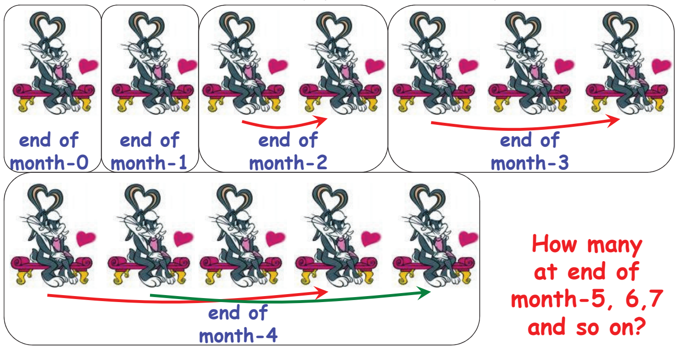
Depending on the recurrence, we can guess the order of growth

$T(n) = T(\frac{n}{2})+1$	T(n) is O(log n)
$T(n) = 2\times T(\frac{n}{2})+1$	T(n) is O(n)
$T(n) = 2\times T(\frac{n}{2})+n$	T(n) is O(n log n)
$T(n) = 2\times T(n-1)+1$	**T(n) is $O(2^n)$**

Fibonacci number ...

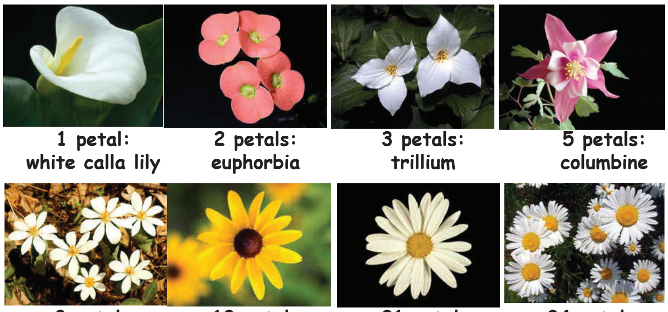
Fibonacci's Rabbits

A pair of rabbits, one month old, is too young to reproduce. Suppose that in their second month, and every month thereafter, they produce a new pair.



How many at end of month-5, 6,7 and so on?

Petals on flowers



1 petal: white calla lily
2 petals: euphorbia
3 petals: trillium
5 petals: columbine
8 petals: bloodroot
13 petals: black-eyed susan
21 petals: shasta daisy
34 petals: field daisy

Search: Fibonacci Numbers in Nature

Fibonacci number

Fibonacci number F(n)

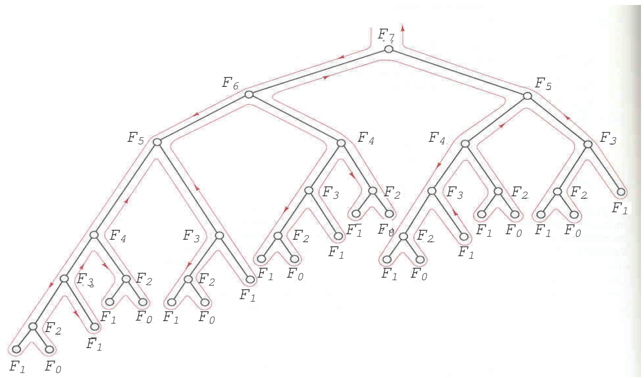
$$F(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
F(n)	1	1	2	3	5	8	13	21	34	55	89

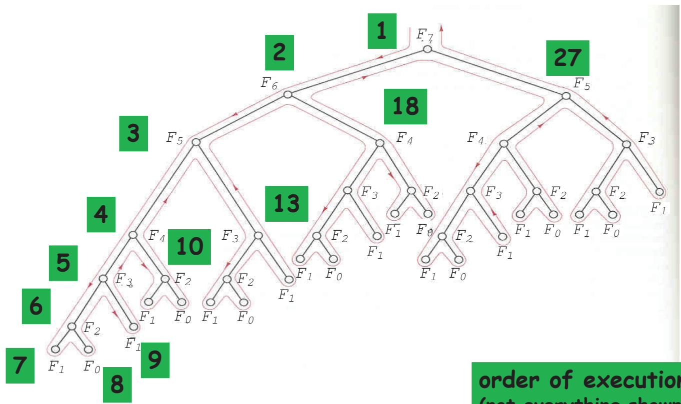
Pseudo code for the recursive algorithm:

```
Algorithm F(n)
   if n==0 or n==1 then
      return 1
   else
      return F(n-1) + F(n-2)
```


The execution of F(7)

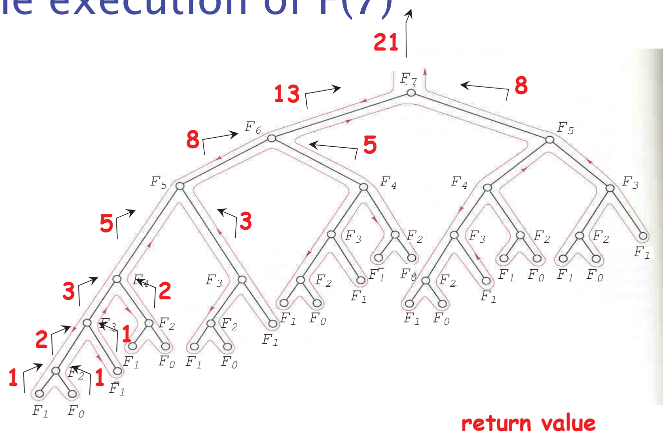


The execution of F(7)



order of execution
(not everything shown)

The execution of F(7)



return value
(not everything shown)

Time complexity - exponential

$$\begin{aligned} f(n) &= f(n-1) + f(n-2) + 1 \\ &= [f(n-2)+f(n-3)+1] + f(n-2) + 1 \\ &> 2\ f(n-2) \\ &> 2\ [2\times f(n-2-2)] = 2^2\ f(n-4) \\ &> 2^2\ [2\times f(n-4-2)] = 2^3\ f(n-6) \\ &> 2^3\ [2\times f(n-6-2)] = 2^4\ f(n-8) \\ &\ldots \\ &> 2^k\ f(n-2k) \end{aligned}$$

Suppose f(n) denote the time complexity to compute F(n)

exponential in n

If n is even, $f(n) > 2^{n/2}\ f(0) = 2^{n/2}$

If n is odd, $f(n) > f(n-1) > 2^{(n-1)/2}$

Robozzle - Recursion

Task: to program a robot to pick up all stars in a certain area

Command: Go straight, Turn Left, Turn Right