Recursion

Plan

Recursion is one of the most important techniques in computer science.

The main idea is to capture the invariants over repeated actions.

- Setting up recurrences
  - Fibonacci recurrence
  - Problem solving recurrences
  - Catalan recurrences
- Solving recurrences

Recursively Defined Sequences

We can define a sequence by specifying relation between the current term and the previous terms.

- Arithmetic sequence: $(a, a+d, a+2d, a+3d, \ldots,)$

  recursive definition: $a_0=a$, $a_{i+1}=a_i+d$

- Geometric sequence: $(a, ra, r^2a, r^3a, \ldots,)$

  recursive definition: $a_0=a$, $a_{i+1}=ra_i$

- Harmonic sequence: $(1, 1/2, 1/3, 1/4, \ldots,)$

  recursive definition: $a_0=1$, $a_{i+1}=ia_i/(i+1)$

Rabbit Populations

The Rabbit Population

- A mature boy/girl rabbit pair reproduces every month.
- Rabbits mature after one month.

  w_n::= # newborn pairs in the n-th month

  r_n::= # pairs in the n-th month

- Start with a newborn pair: $w_0 = 1$, $r_0 = 0$

How many rabbits after n months?

Rabbit Populations

w_n::= # newborn pairs in the n-th month

r_n::= # pairs in the n-th month

$$r_1 = 1$$

$$r_n = r_{n-1} + w_{n-1}$$

$$w_n = r_{n-1} \quad \text{so}$$

$$r_n = r_{n-1} + r_{n-2}$$

Rabbits could overpopulate easily, see Rabbits in Australia for example.

It was Fibonacci who studied rabbit population growth.

We will compute the closed form for r_n soon.

A formula so that # steps for calculating $r_n \leq$ constant

Warm Up

We will solve counting problems by setting up recurrence relations.

First we use recursion to count something we already know,
to get familiar with this approach.

Let us count the number of elements in pow(S_n)
where $S_n = \{a_1, a_2, \ldots, a_n\}$ is an n-element set.

Let r_n be the size of pow(S_n).

Then $r_1 = 2$, where pow(S_1) $= \{\Phi, \{a_1\}\}$

$r_2 = 4$, where pow(S_2) $= \{\Phi, \{a_1\}, \{a_2\}, \{a_1,a_2\}\}$

Warm Up

Let r_n be the size of pow(S_n) where $S_n = \{a_1, a_2, \ldots, a_n\}$ is an n-element set.

Then $r_1 = 2$, where pow(S_1) = $\{\Phi, \{a_1\}\}$

$r_2 = 4$, where pow(S_2) = $\{\Phi, \{a_1\}, \{a_2\}, \{a_1,a_2\}\}$

The main idea of recursion is to define r_n in terms of the previous r_i.

How to define r_3 in terms of r_1 and r_2?

pow(S_3) = the union of $\{\Phi, \{a_1\}, \{a_2\}, \{a_1,a_2\}\}$

and $\{a_3, \{a_1,a_3\}, \{a_2,a_3\}, \{a_1,a_2,a_3\}\}$

So $r_3 = 2r_2$.

while the lower sets are obtained by adding a_3 to the upper sets.

Warm Up

Let r_n be the size of pow(S_n) where $S_n = \{a_1, a_2, \ldots, a_n\}$ is an n-element set.

$\boxed{\text{The main idea of recursion is to define } r_n \text{ in terms of the previous } r_i.}$

pow(S_n) = the union of $S_{n-1} = \{\Phi,\ \{a_1\},\ \{a_2\},\ \ldots,\ \{a_1,a_2,\ldots,a_{n-1}\}\}$

$\updownarrow \quad \updownarrow \quad \updownarrow \quad \updownarrow$

and $\{a_n,\ \{a_1,a_n\},\ \{a_2,a_n\},\ \ldots,\ \{a_1,a_2,\ldots,a_{n-1},a_n\}\}$

while the lower sets are obtained by adding a_n to the upper sets.

Every subset is counted exactly once. $\boxed{\text{So } r_n = 2r_{n-1}.}$

Solving this recurrence relation will show that $r_n = 2^n$ (geometric sequence).

Number of Bit Strings **without** a Specific Pattern

How many n-bit strings without the bit pattern 11?

Let r_n be the number of such strings.

e.g. $r_1 = |\{0, 1\}| = 2$,

$r_2 = |\{00, 01, 10\}| = 3$

$r_3 = |\{000, 001, 010, 100, 101\}| = 5$

$r_4 = |\{0000, 0001, 0010, 0100, 0101, 1000, 1001, 1010\}| = 8$

Can you see the pattern?

$r_4 = |\{0000, 0001, 0010, 0100, 0101\}$ union

$\{1000, 1001, 1010\}|$ $= 5 + 3 = 8$

Number of Bit Strings without a Specific Pattern

How many n-bit strings without the bit pattern 11?

Let r_n be the number of such strings.

How do we compute it using $r_1, r_2, \ldots, r_{n-1}$?

Case 1: The first bit is 0.

Then any (n-1)-bit string without the bit pattern 11

can be appended to the end to form an n-bit string without 11.

So in this case there are exactly r_{n-1} such n-bit strings.

0 +

00000000000000000
00000000000000001
…
10101010101010101

The set of all (n-1)-bit strings without 11.

Totally r_{n-1} of them.

Number of Bit Strings without a Specific Pattern

How many n-bit strings without the bit pattern 11?

Let r_n be the number of such strings.

How do we compute it using $r_1, r_2, \ldots, r_{n-1}$?

Case 2: The first bit is 1.

Then the second bit must be 0, because we can't have 11.

Then any (n-2)-bit string without the bit pattern 11

can be appended to the end to form an n-bit string without 11.

So in this case there are exactly r_{n-2} such n-bit strings.

10 +

0000000000000000
0000000000000001
…
1010101010101010

The set of all (n-2)-bit strings without 11.

Totally r_{n-2} of them.

Number of Bit Strings without a Specific Pattern

How many n-bit string without the bit pattern 11?

0 +

00000000000000000
00000000000000001
…
10101010101010101

The set of all (n-1)-bit strings without 11. Totally r_{n-1} of them.

10 +

0000000000000000
0000000000000001
…
1010101010101010

The set of all (n-2)-bit strings without 11. Totally r_{n-2} of them.

Every string without the bit pattern 11 is counted exactly once.

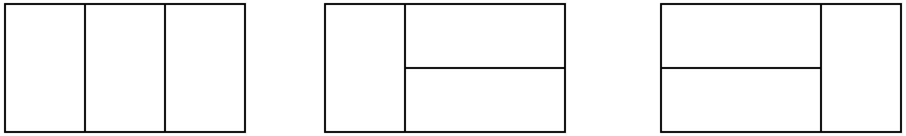
Therefore, $r_n = r_{n-1} + r_{n-2}$

Exercise

How many n-bit strings without the bit pattern 111?

Domino

Given a 2xn puzzle, how many ways to fill it with dominos (2x1 tiles)?



E.g. There are 3 ways to fill a 2x3 puzzle with dominos.

Let r_n be the number of ways to fill a 2xn puzzle with dominos.

How do we compute it using $r_1, r_2, \ldots, r_{n-1}$?

Domino

Given a 2xn puzzle, how many ways to fill it with dominos (2x1 tiles)?

Let r_n be the number of ways to fill a 2xn puzzle with dominos.

Case 1: put the domino vertically

r_{n-1} to fill the remaining 2x(n-1) puzzle

Case 2: put the domino horizontally

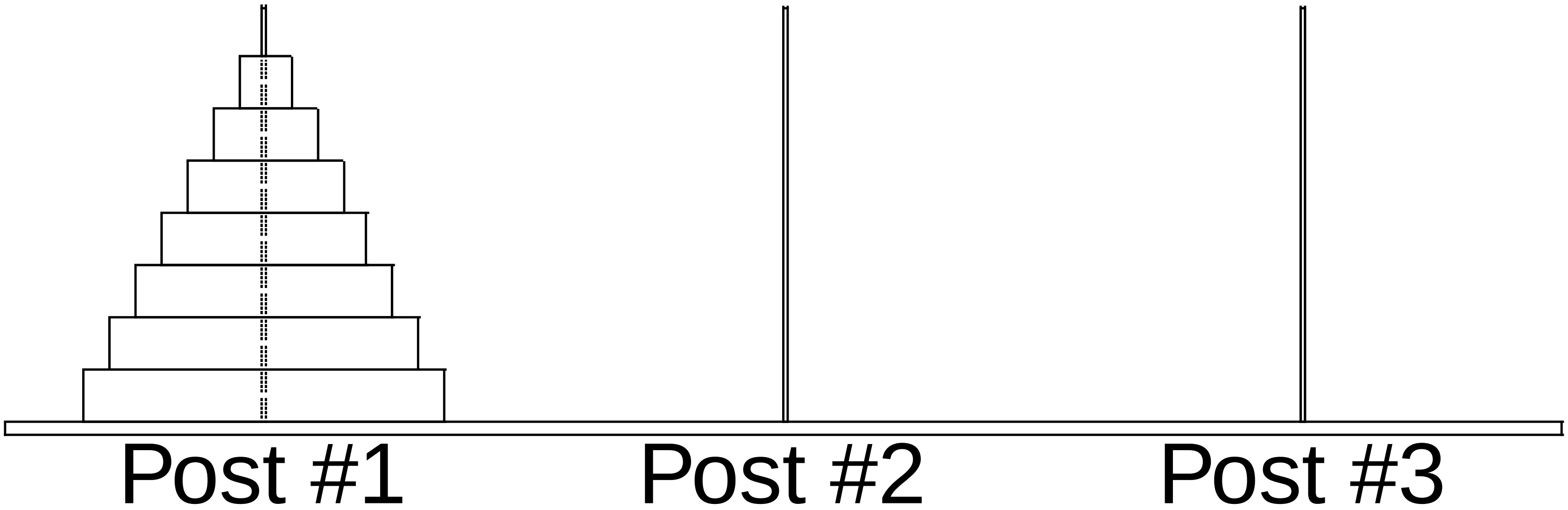
r_{n-2} to fill the remaining 2x(n-2) puzzle

Therefore, $r_n = r_{n-1} + r_{n-2}$

Plan

- Setting up recurrences
  - Fibonacci recurrence
  - Problem solving recurrences
  - Catalan recurrences
- Solving recurrences

Tower of Hanoi

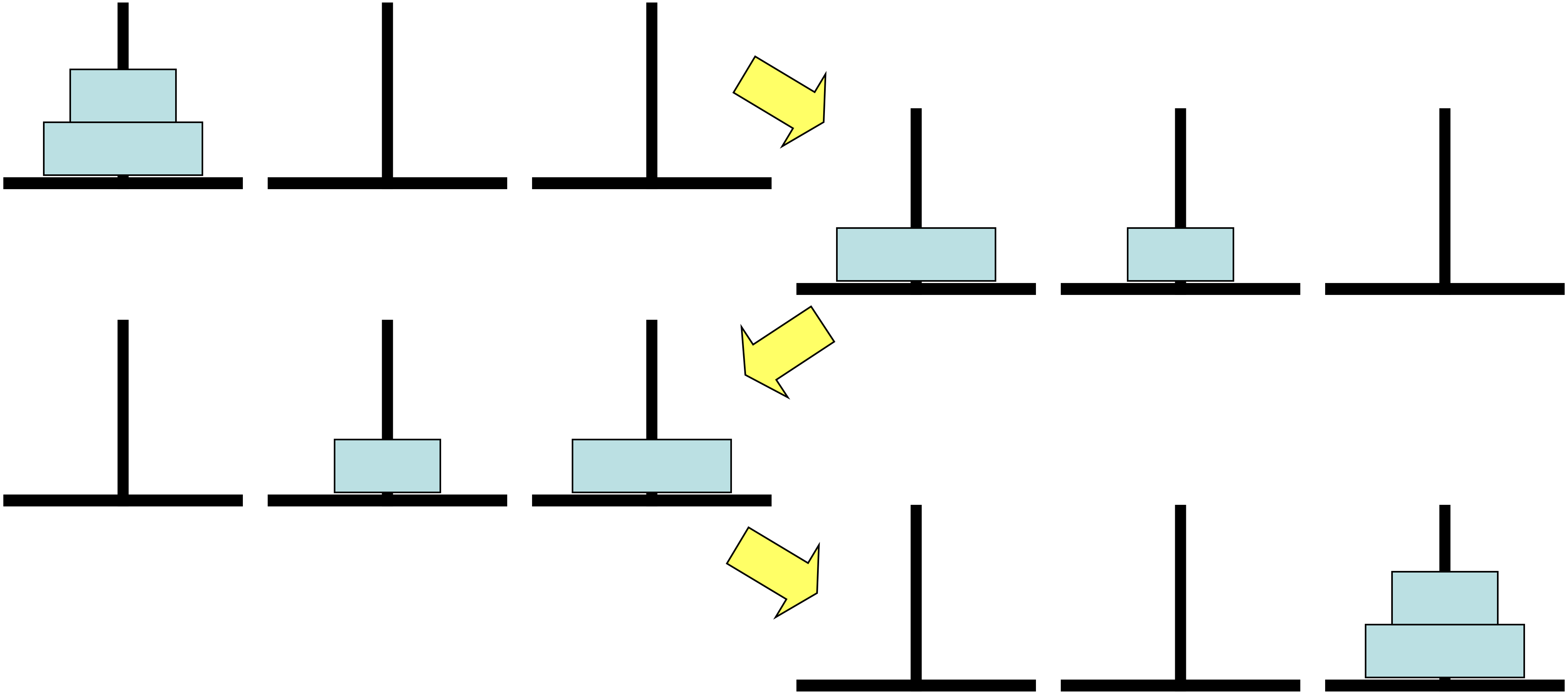


The goal is to move all the disks to post 3.

The rule is that a bigger disk cannot be placed on top of a smaller disk.

Tower of Hanoi

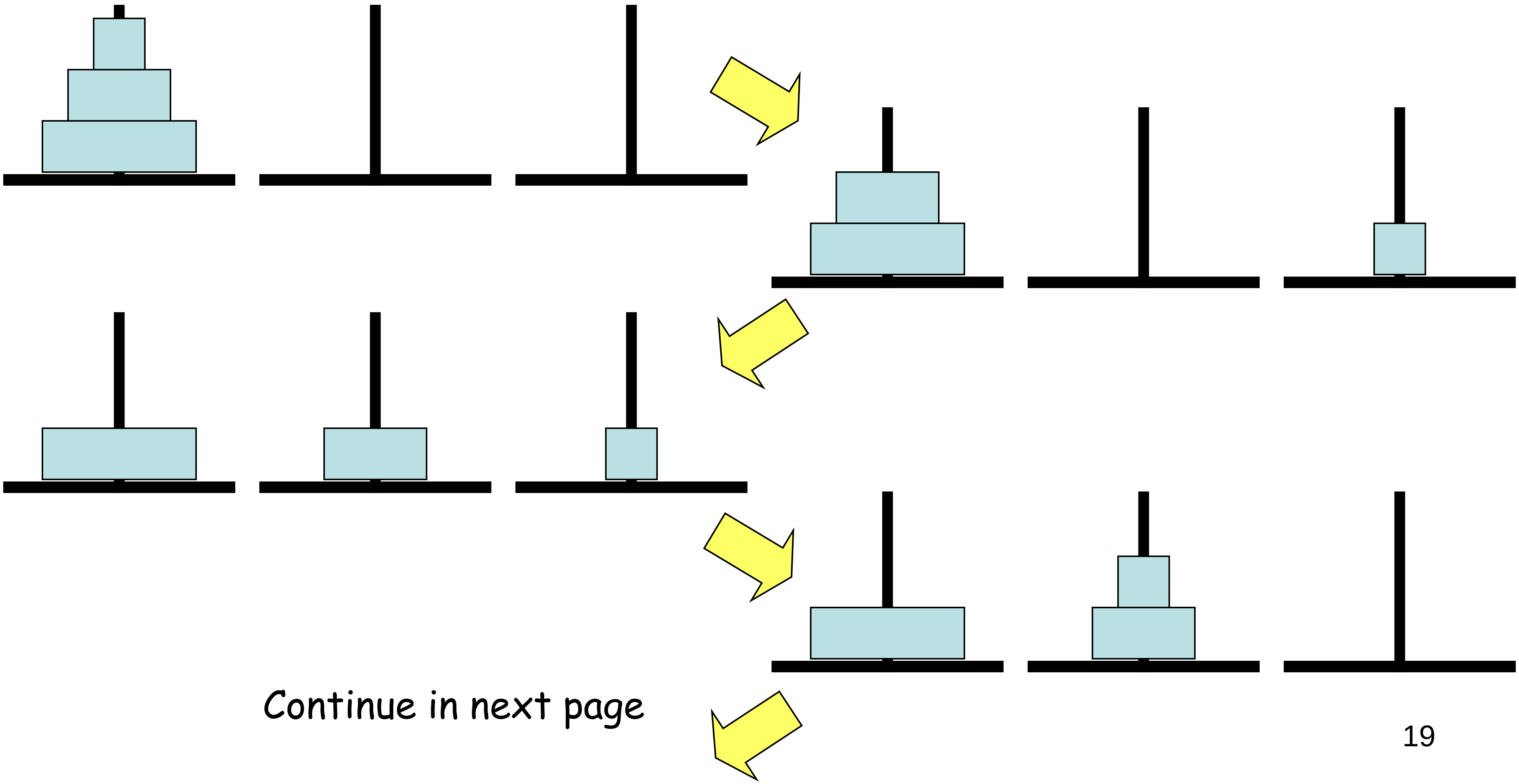
It is easy if we only have 2 disks.



A total of 3 steps.

Tower of Hanoi

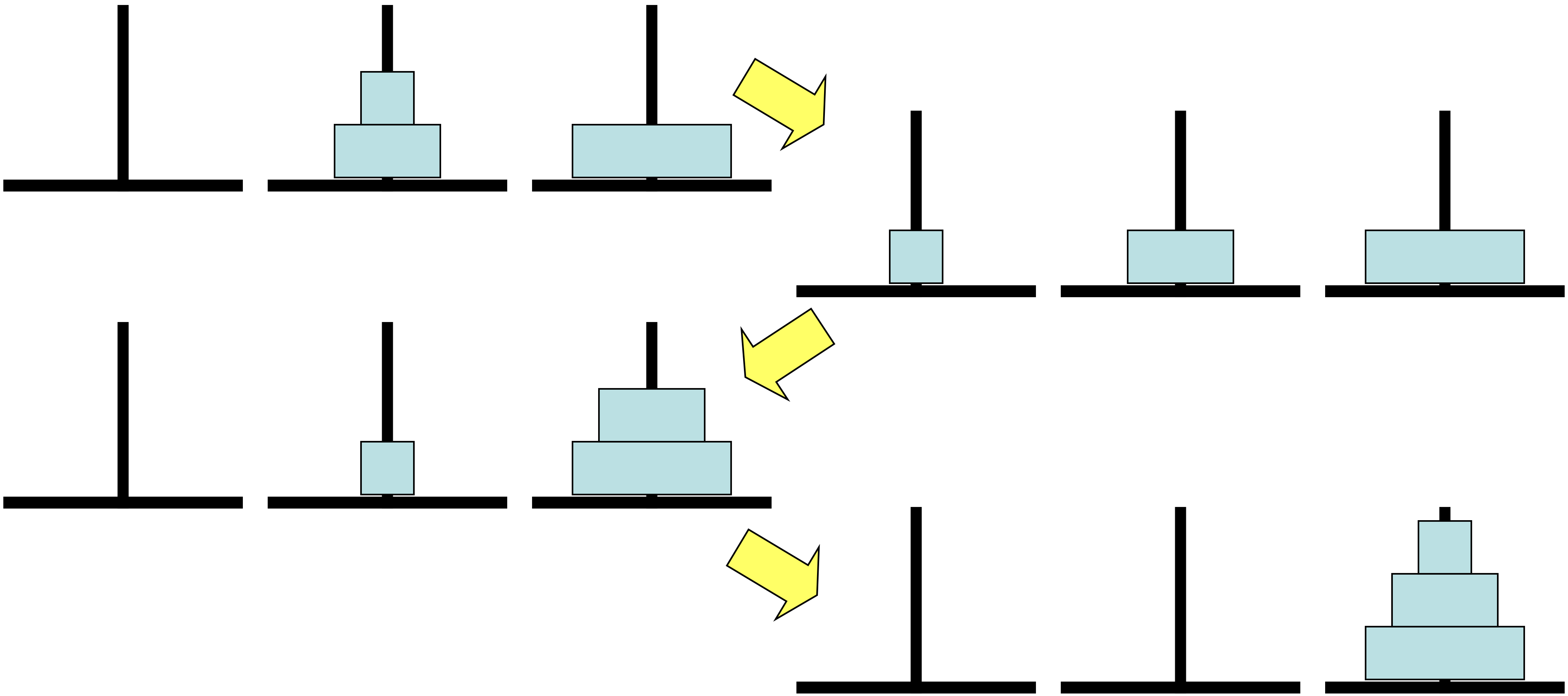
It is not difficult if we only have 3 disks.



Continue in next page

Tower of Hanoi

It is not difficult if we only have 3 disks.



A total of seven steps.

Tower of Hanoi

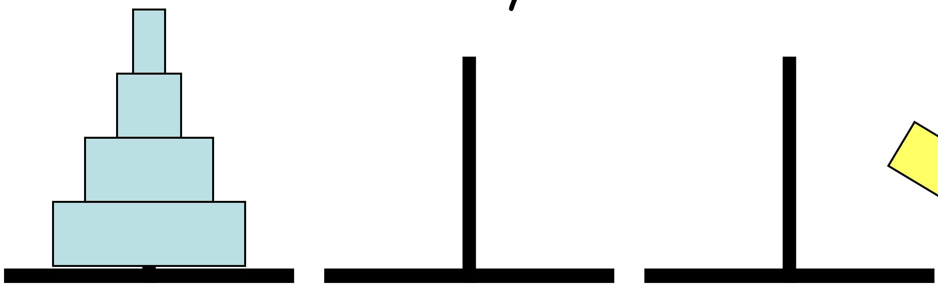
Can you write a program to solve this problem?

Think recursively!

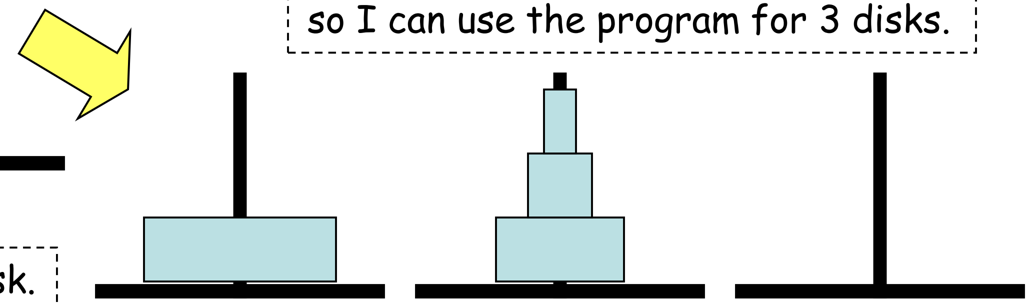
Tower of Hanoi

Suppose you already have a program for 3 disks.

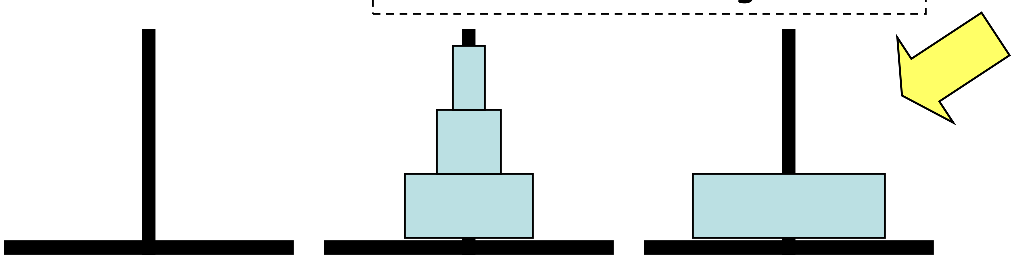
Can you use it to solve 4 disks?



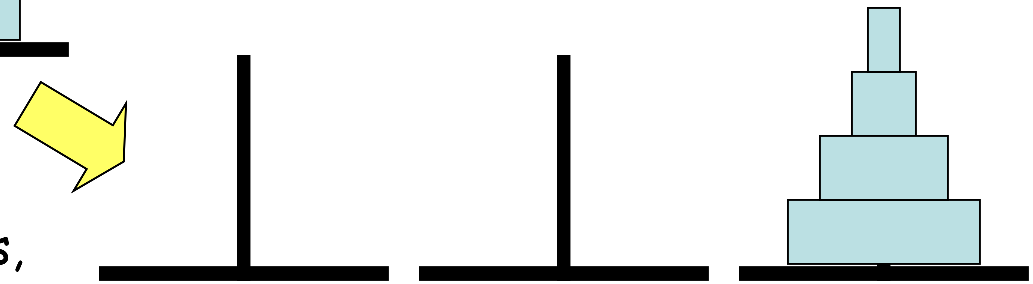
In order to move the largest disk,
I have to move the top 3 disks first,
so I can use the program for 3 disks.



Then I move the largest disk.



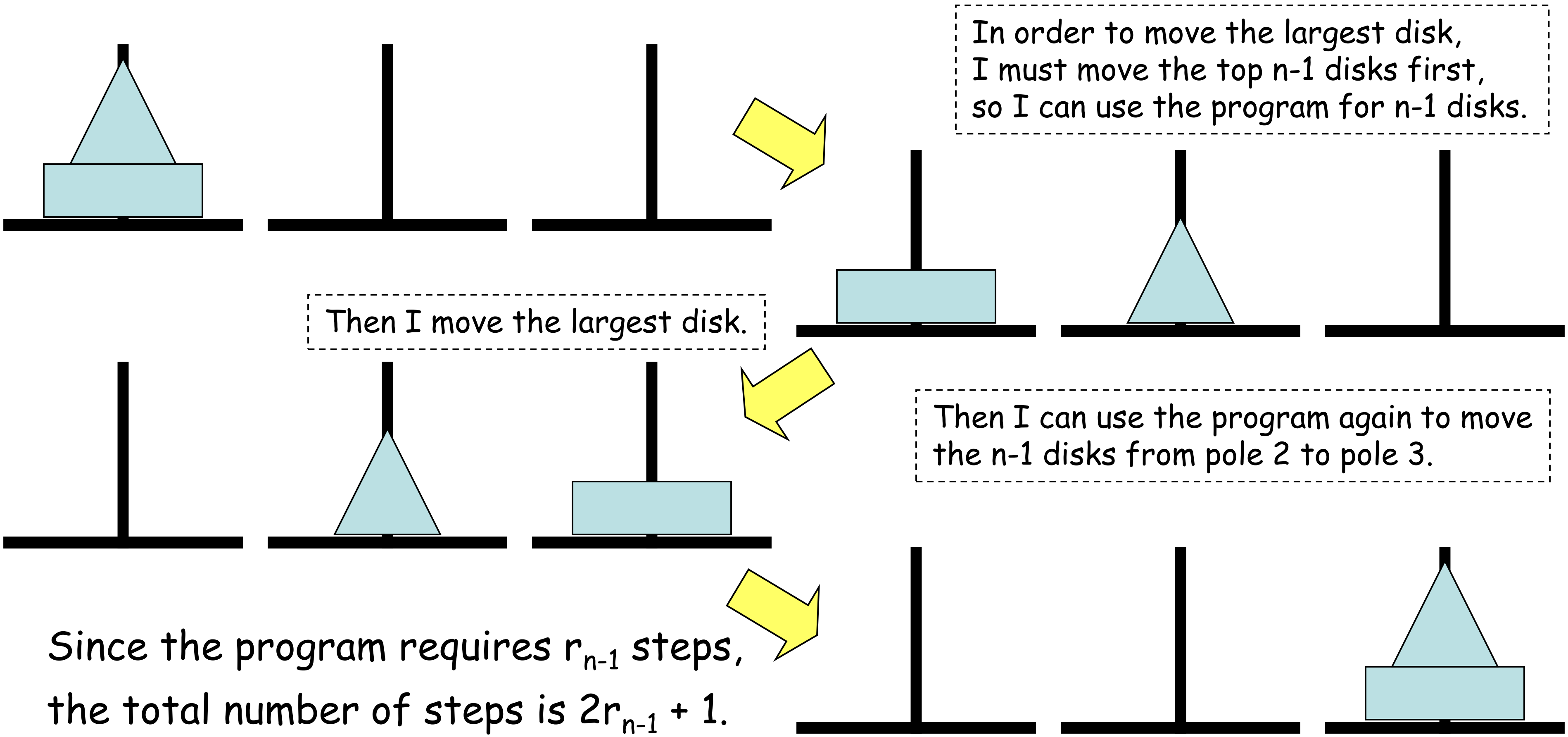
Then I can use the program to move
the 3 disks from pole 2 to pole 3.



Since the program requires 7 steps,
the total number of steps is 15.

Tower of Hanoi

This recursion is true for any n.



Since the program requires r_{n-1} steps,

the total number of steps is $2r_{n-1} + 1$.

$$\boxed{r_n = 2r_{n-1} + 1}$$

Optimality

What will be the minimum number of steps needed if we have 4 poles?

Frame–Stewart algorithm is optimal. (2016)

What will be the minimum number of steps needed if we have 5 poles?

What if we have even more poles?

Open problem…

This leads to the "Frame-Stewart conjecture"

Note: Another generalization of the original puzzle is to move a given configuration to another. This leads to general shortest path problem, which is hard to compute in general.

Merge Sort

Given a sequence of n numbers,
how many steps are required to sort them into non-decreasing order?

One way to sort the numbers is called the “bubble sort”, in which every step we compare two adjacent numbers, and make a swap if they are in the wrong order:

- In the first pass, we start by comparing the last two numbers and exchange them if necessary, and we continue up the list until we compare the second and the first item
- In the second pass, we work from the nth number to the second number and so on.
- Assuming the maximum number of passes is (n-1), this algorithm could take up to roughly $n^2/2$ steps.

For example, if we are given the reversed sequence n,n-1,n-2,…,1.

Every time it will search to the end to find the smallest number,
so the algorithm takes roughly $(n-1)+(n-2)+\ldots+1 = n(n-1)/2$ steps.

Can we design a faster algorithm?

Think recursively!

Merge Sort

Suppose we have a program to sort n/2 numbers.

We can use it to sort n numbers in the following way.

3 8 4 7 2 1 6 5

Divide the sequence into two halves.

Use the program to sort the two halves independently.

3 4 7 8	1 2 5 6

With these two sorted sequences of n/2 numbers,
we can merge them into a sorted sequence of n numbers easily!

Merge Sort

Claim. Suppose we have two sorted sequences of k numbers. We can merge them into a sorted sequence of 2k numbers in 2k steps.

Proof by example:

3 5 7 8 9 10

1 2 4 6 11 12

↑ ↑

To decide the smallest number in the two sequences, we just need to look at the "heads" of the two sequences.

So, for each step or comparison, we can extend the sorted sequence by one number. So the total number of comparisons for 2k numbers is 2k.

Merge Sort

3 5 7 8 9 10	1 2 4 6 11 12	1
3 5 7 8 9 10	1 2 4 6 11 12	1 2
3 5 7 8 9 10	1 2 4 6 11 12	1 2 3
3 5 7 8 9 10	1 2 4 6 11 12	1 2 3 4
3 5 7 8 9 10	1 2 4 6 11 12	1 2 3 4 5
3 5 7 8 9 10	1 2 4 6 11 12	1 2 3 4 5 6
3 5 7 8 9 10	1 2 4 6 11 12	1 2 3 4 5 6 7
3 5 7 8 9 10	1 2 4 6 11 12	1 2 3 4 5 6 7 8
3 5 7 8 9 10	1 2 4 6 11 12	1 2 3 4 5 6 7 8 9
3 5 7 8 9 10	1 2 4 6 11 12	1 2 3 4 5 6 7 8 9 10
3 5 7 8 9 10	1 2 4 6 11 12	1 2 3 4 5 6 7 8 9 10 11
3 5 7 8 9 10	1 2 4 6 11 12	1 2 3 4 5 6 7 8 9 10 11 12

Merge Sort

Claim. Suppose we have two sorted sequences of k numbers.
We can merge them into a sorted sequence
of $2k$ numbers in $2k$ steps.

Suppose we can sort k numbers in T_k steps.

Then we can sort 2k numbers in $2T_k + 2k$ steps.

Therefore, $T_{2k} = 2T_k + 2k$. (What if n is odd?)

If we solve this recurrence (which we will do later),

then we will see that $T_{2n} \le n \log_2 n$.

This is significantly faster than bubble sort!

Remark

This is an example of the “divide and conquer” algorithm.

This idea is very powerful.

It can be used to design faster algorithms for some basic problems such as integer multiplications, matrix multiplications, etc.

Plan

- Setting up recurrences
  - Fibonacci recurrence
  - Problem solving recurrences
  - Catalan recurrences
- Solving recurrences

Parenthesis

How many valid ways to add n pairs of parentheses?

E.g. There are 5 valid ways to add 3 pairs of parentheses.

((()))   (()())   (())()   ()(())   ()()()

Let r_n be the number of ways to add n pairs of parentheses.

How do we compute it using $r_1, r_2, \ldots, r_{n-1}$?

Parenthesis

How many valid ways to add n pairs of parentheses?

Let r_n be the number of ways to add n pairs of parentheses.

Case 1: ()--------------------

r_{n-1} ways to add the remaining n-1 pairs.

So there are r_{n-1} in this case.

Case 2: (--)-----------------

1 way to add 1 pair

r_{n-2} ways to add the remaining n-2 pairs.

So there are r_{n-2} in this case.

Case 3: (----)----------------

2 ways to add 2 pairs

r_{n-3} ways to add the remaining n-3 pairs.

So there are $2r_{n-3}$ in this case.

Parenthesis

How many valid ways to add n pairs of parentheses?

Let r_n be the number of ways to add n pairs of parenthese.

Case k: (----------)----------

r_{k-1} ways to add k-1 pairs r_{n-k} ways to add the remaining n-k pairs.

By the product rule, there are $r_{k-1}r_{n-k}$ ways in case k.

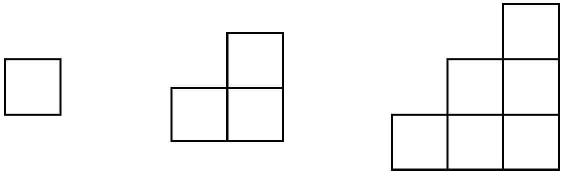
The cases are sorted by the position of the close parenthesis of the first open parenthesis, and so these cases are disjoint.

Therefore, by the sum rule, $r_n = \sum_{k=1}^{n} r_{k-1} r_{n-k}$

Stairs

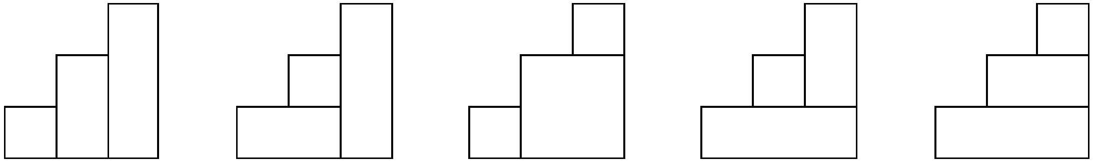
An **n-stair** is the collection of unit squares bounded by x-axis, y=x and x=n+1.

For example 1-stair, 2-stair, and 3-stair are like this:



How many ways to fill up the n-stair by exactly n rectangles??

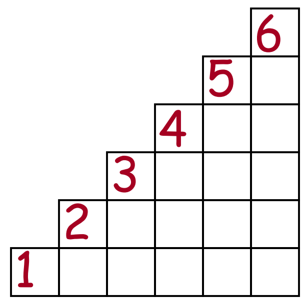
For example, there are 5 ways to fill up the 3-stair by 3 rectangles.



Stairs

Let r_n be the number of ways to fill the n-stair by n rectangles.

How do we compute it using $r_1, r_2, \ldots, r_{n-1}$?



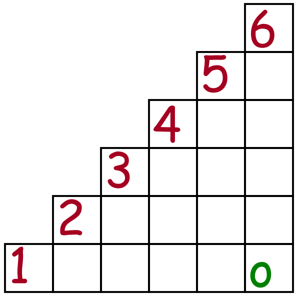
Given the n-stair, the first observation is that the positions on the diagonal (red numbers) must be covered by different rectangles.

Since there are n positions in the diagonal and we can only use n rectangles, each rectangle must cover exactly one red number.

Stairs

Let r_n be the number of ways to fill the n-stair by n rectangles.

How do we compute it using $r_1, r_2, \ldots, r_{n-1}$?



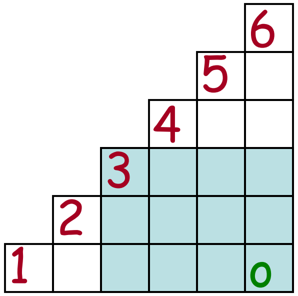
Consider the rectangle R that covers the bottom right corner (marked with o).

We consider different cases depending on which red number is covered by R.

Stairs

Let r_n be the number of ways to fill the n-stair by n rectangles.

How do we compute it using $r_1, r_2, \ldots, r_{n-1}$?



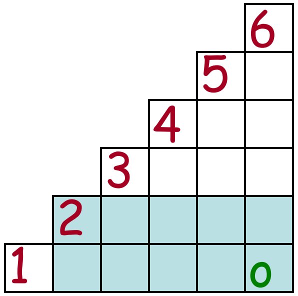
Suppose R covers 3. Then the 6-stair is divided into 3 parts.
One part is the rectangle R. The other two parts are a 2-stair and a 3-stair.

Therefore, in this case, the number of ways to fill in the remaining parts is equal to $r_2 r_3$

Stairs

Let r_n be the number of ways to fill the n-stair by n rectangles.

How do we compute it using $r_1, r_2, \ldots, r_{n-1}$?



Similarly suppose R covers 2.

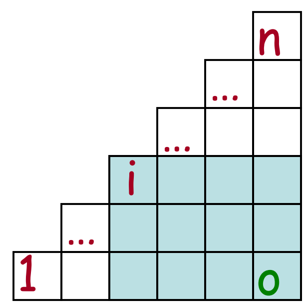
Then the rectangle "breaks" the stair into a 1-stair and a 4-stair.

Therefore, in this case, the number of ways to fill in the remaining parts is equal to $r_1 r_4$

Stairs

Let r_n be the number of ways to fill the n-stair by n rectangles.

How do we compute it using $r_1, r_2, \ldots, r_{n-1}$?



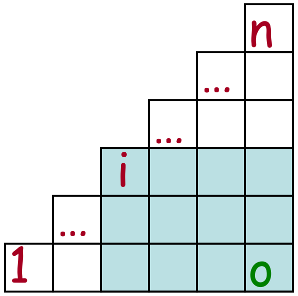
In general suppose the rectangle covers i

Then the rectangle "breaks" the stair into an (i-1)-stair and an (n-i)-stair.

Therefore, in this case, the number of ways to fill in the remaining parts is equal to $r_{i-1}r_{n-i}$ (we define $r_0=1$)

Stairs

The number of ways to fill in the remaining parts is equal to $r_{i-1}r_{n-i}$



Rectangle R covering different i will form different configurations, and each configuration must correspond to one of these cases.

Therefore the total number of ways is equal to

$$r_n = \sum_{i=1}^{n} r_{i-1}r_{n-i}$$

Catalan Number

How many valid ways to add n pairs of parentheses?

$$r_n = \sum_{k=1}^{n} r_{k-1} r_{n-k}$$

So the recursion for the stair problem is the same as the recursion for the parentheses problem. It can be shown that

$$r_n = \frac{1}{n+1}\binom{2n}{n}$$

This is well known as the n-th Catalan number.

Plan

- Setting up recurrences
  - Fibonacci recurrence
  - Problem solving recurrences
  - Catalan recurrences
- Solving recurrences

Warm Up

$a_0=1, \qquad a_k = a_{k-1} + 2$

$a_1 = a_0 + 2$

$a_2 = a_1 + 2 = (a_0 + 2) + 2 = a_0 + 4$

$a_3 = a_2 + 2 = (a_0 + 4) + 2 = a_0 + 6$

$a_4 = a_3 + 2 = (a_0 + 6) + 2 = a_0 + 8$

See the pattern is $a_k = a_0 + 2k = 2k+1$

You can verify by induction.

Solving Hanoi Sequence

$a_1=1, \qquad a_k = 2a_{k-1} + 1$

$a_2 = 2a_1 + 1 = 3$

$a_3 = 2a_2 + 1 = 2(2a_1 + 1) + 1 = 4a_1 + 3 = 7$

$a_4 = 2a_3 + 1 = 2(4a_1 + 3) + 1 = 8a_1 + 7 = 15$

$a_5 = 2a_4 + 1 = 2(8a_1 + 7) + 1 = 16a_1 + 15 = 31$

$a_6 = 2a_5 + 1 = 2(16a_1 + 15) + 1 = 32a_1 + 31 = 63$

Guess the pattern is $a_k = 2^k-1$

You can verify by induction.

Solving Merge Sort Recurrence

$$\boxed{T_{2k} = 2T_k + 2k}$$

If we could guess that $T_k = k \log_2 k$,

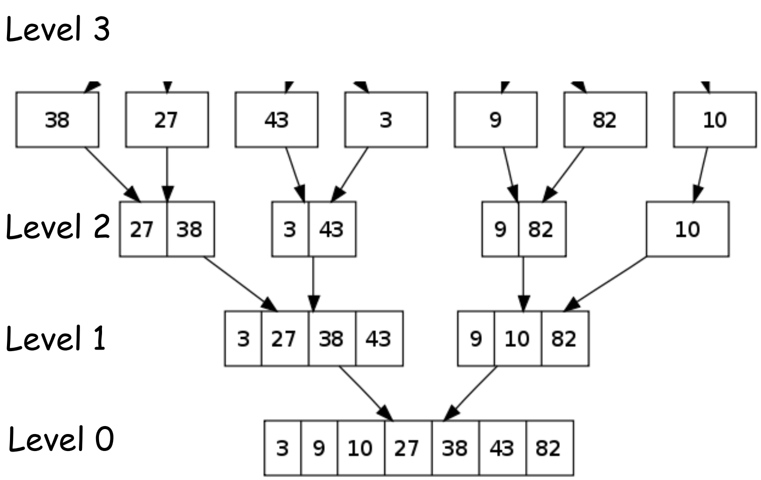
then we can prove that $T_{2k} = 2k \log_2(2k)$.

This is because
$$
\begin{aligned}
T_{2k} &= 2T_k + 2k \\
&= 2k\log_2 k + 2k \\
&= 2k(\log_2 k + 1) \\
&= 2k(\log_2 k + \log_2 2) \\
&= 2k\log_2 2k
\end{aligned}
$$

Solving Merge Sort Recurrence

$$T_{2k} = 2T_k + 2k$$

How could we guess $T_k = k \log_2 k$?



If there are k numbers, there are $h=\log_2 k$ levels (since we keep splitting the numbers until it reaches a single element, i.e. $k/2^h=1 \Rightarrow h=\log_2 k$)

In each level i we need to solve 2^{i-1} merge problems, since we keep splitting into two each time from level 0

Each merge problem in level i has two subsequences, each having $n/2^i$ numbers, and so can be merged in $2x(k/2^i) = k/2^{i-1}$ steps (from earlier results)

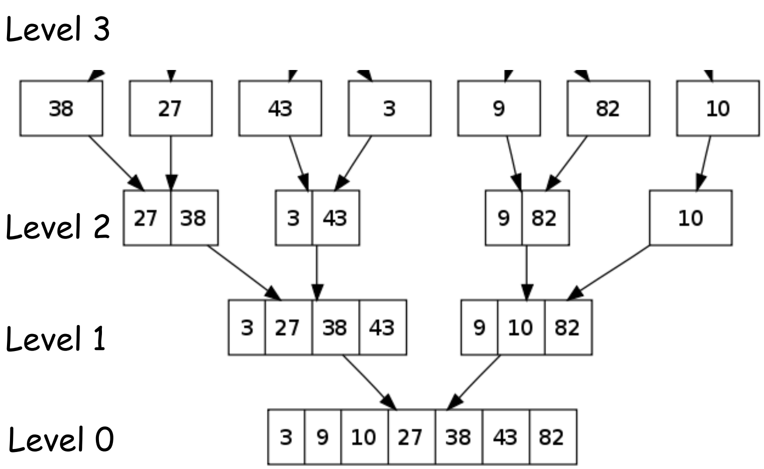
$\Rightarrow$ each level requires a total of $(2^{i-1})(k/2^{i-1})=k$ steps.

Since there are $\log_2 n$ levels, the total number of steps is at most $n\log_2 n$.

Solving Merge Sort Recurrence

$$T_{2k} = 2T_k + 2k$$

How could we guess $T_k = k \log_2 k$?



$$T_k \leq \log_2 k \cdot (2^{i-1}) \cdot (k/2^{i-1}) = k\log_2 k$$

#levels

#merge problems in level i

#numbers in each merge problem in level i
= #steps needed to sort numbers in each merge problem

Equality is achieved when k is a power of 2.

Solving Fibonacci Sequence

$a_0=0, \quad a_1=1, \quad a_n = a_{n-1} + a_{n-2}$

$a_2 = a_1 + a_0 = 1$

$a_3 = a_2 + a_1 = 2$

$a_4 = a_3 + a_2 = 3$

$a_5 = a_4 + a_3 = 5$

$a_6 = a_5 + a_4 = 8$

$a_7 = a_6 + a_5 = 13$

…

How do we find a formula for a_n ?

Generating Functions

$\langle 0,0,0,0,\ldots\rangle \leftrightarrow 0+0x+0x^2+0x^3+\ldots=0$

$\langle 1,1,1,1,\ldots\rangle \leftrightarrow 1+x+x^2+x^3+\ldots \quad =1/(1-x)$

$\langle 1,-1,1,-1,\ldots\rangle \leftrightarrow 1-x+x^2-x^3+\ldots \quad =1/(1+x)$

$$\boxed{\langle a_0,a_1,a_2,a_3,\ldots\rangle \leftrightarrow F(x)}$$

This is called the ordinary generating function for $\{a_n\}$:

$$\boxed{F(x)=a_0+a_1x+a_2x^2+a_3x^3+\ldots}$$

sequence $\leftrightarrow$ generating function

Generating Functions

Scaling:

$$\langle f_0, f_1, f_2, \ldots \rangle \longleftrightarrow F(x) \quad \Longrightarrow \quad \langle cf_0,\ cf_1,\ cf_2,\ \ldots \rangle \longleftrightarrow c \cdot F(x)$$

Addition:

$$\begin{array}{rl} & \langle f_0, f_1, f_2, \ldots \rangle \longleftrightarrow F(x) \\ + & \langle g_0, g_1, g_2, \ldots \rangle \longleftrightarrow G(x) \\ \hline & \langle f_0 + g_0,\ f_1 + g_1,\ f_2 + g_2,\ \ldots \rangle \longleftrightarrow F(x) + G(x) \end{array}$$

Right shifting:

$$\langle \underbrace{0, 0, \ldots, 0}_{k \text{ zeroes}}, f_0, f_1, f_2, \ldots \rangle \longleftrightarrow x^k \cdot F(x)$$

Differentiation:

$$\langle f_0, f_1, f_2, f_3, \ldots \rangle \longleftrightarrow F(x) \quad \Longrightarrow \quad \langle f_1, 2f_2, 3f_3, \ldots \rangle \longleftrightarrow F'(x)$$

Generating Functions

How do we find the generating function for <0,1,4,9,...> ?

$$\langle 1,1,1,1,\ldots\rangle \longleftrightarrow \frac{1}{1-x}$$

$$\langle 1,2,3,4,\ldots\rangle \longleftrightarrow \frac{d}{dx}\frac{1}{1-x} = \frac{1}{(1-x)^2}$$

$$\langle 0,1,2,3,\ldots\rangle \longleftrightarrow x\cdot\frac{1}{(1-x)^2} = \frac{x}{(1-x)^2}$$

$$\langle 1,4,9,16,\ldots\rangle \longleftrightarrow \frac{d}{dx}\frac{x}{(1-x)^2} = \frac{1+x}{(1-x)^3}$$

$$\langle 0,1,4,9,\ldots\rangle \longleftrightarrow x\cdot\frac{1+x}{(1-x)^3} = \frac{x(1+x)}{(1-x)^3}$$

So the generating function is

$$\boxed{\frac{x(1+x)}{(1-x)^3}}$$

Solving Fibonacci Sequence

How does generating function help solve Fibonacci sequence ?

Fibonacci sequence: $f_0=0, \quad f_1=1, \quad f_n = f_{n-1} + f_{n-2}$

So the generating function for $\{f_n\}$ is

$$F(x)=f_0+f_1x+f_2x^2+f_3x^3+\ldots=\boxed{0+x+(f_1+f_0)x^2+(f_2+f_1)x^3+\ldots}$$

We can find the generating function for $\langle 0,1,f_1+f_0,f_2+f_1,\ldots\rangle$!

$$\begin{array}{rccccccclcl}
 & \langle & 0, & 1, & 0, & 0, & 0, & \ldots & \rangle & \longleftrightarrow & x \\
 & \langle & 0, & f_0, & f_1, & f_2, & f_3, & \ldots & \rangle & \longleftrightarrow & xF(x) \\
+ & \langle & 0, & 0, & f_0, & f_1, & f_2, & \ldots & \rangle & \longleftrightarrow & x^2F(x) \\
\hline
 & \langle & 0, & 1+f_0, & f_1+f_0, & f_2+f_1, & f_3+f_2, & \ldots & \rangle & \longleftrightarrow & x+xF(x)+x^2F(x)
\end{array}$$

$$\boxed{F(x)=x+xF(x)+x^2F(x)}$$

Solving Fibonacci Sequence

Resolving F(x) we get:

$$\boxed{F(x) = x/(1-x-x^2)}$$

Partial fractioning gives:

$$F(x) = \frac{1}{\sqrt{5}}\left(\frac{1}{1-\alpha_1 x} - \frac{1}{1-\alpha_2 x}\right)$$

where $\alpha_1 = \frac{1}{2}(1+\sqrt{5})$ and $\alpha_2 = \frac{1}{2}(1-\sqrt{5})$.

Expand F(x) using Taylor series:

$$F(x) = \frac{1}{\sqrt{5}}\left((1+\alpha_1 x + \alpha_1^2 x^2 + \cdots) - (1 + \alpha_2 x + \alpha_2^2 x^2 + \cdots)\right)$$

$$\Rightarrow \boxed{f_n = \frac{\alpha_1^n - \alpha_2^n}{\sqrt{5}} = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)}$$

Second Order Recurrence Relation

It seems that we do something more general like:

$$\boxed{a_k = Aa_{k-1} + Ba_{k-2}}$$

A and B are real numbers and $B \neq 0$

This is called "second-order linear homogeneous recurrence relation with constant coefficients".

For example, Fibonacci sequence is when A=B=1.

Can we give a general answer to this problem ?

Distinct-Roots Theorem

Suppose a sequence $(a_0,a_1,a_2,a_3,\ldots)$ satisfies a recurrence relation

$$a_k = Aa_{k-1} + Ba_{k-2}$$

If $t^2 - At - B = 0$ has two *distinct* roots r and s,

then $a_n = Cr^n + Ds^n$ for some C and D.

The theorem says that any solution of the recurrence relation is a linear combination of the two series $(1,r,r^2,r^3,r^4,\ldots,r^n,\ldots)$ and $(1,s,s^2,s^3,s^4,\ldots,s^n,\ldots)$ defined by the distinct roots of $t^2 - At - B = 0$.

So, if a_0 and a_1 are given, then C and D are uniquely determined.

Example

$$\boxed{a_n = a_{n-1} + 2a_{n-2}}$$

Need to find solutions of the form $(1, t, t^2, t^3, t^4, \ldots, t^n, \ldots)$

where t is a root of the quadratic equation $t^2 - t - 2 = 0$.

This implies that $r=2$ or $s=-1$.

If we did not know a_0, a_1, they are many solutions for $\{a_n\}$:

(i) (1,2,4,8,16,32,64,…) $\boxed{a_n=r^n}$

(ii) (1,-1,1,-1,1,-1,…) $\boxed{a_n=s^n}$

(iii) (2,1,5,7,17,31,65,…) $\boxed{a_n=r^n+s^n}$

(iv) … …

Revisiting Fibonacci Sequence

If a_0, a_1 are given…

$a_0=0$, $a_1=1$, $a_n = a_{n-1} + a_{n-2}$

First solve the quadratic equation $t^2 - t - 1 = 0$.

$$t = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

So the distinct roots are:

$$r = \frac{1+\sqrt{5}}{2} \qquad s = \frac{1-\sqrt{5}}{2}$$

Revisiting Fibonacci Sequence

$a_0=0, \quad a_1=1, \quad a_n = a_{n-1} + a_{n-2}$

By the distinct-roots theorem, the solutions satisfy the formula:

$$a_n = C(\frac{1+\sqrt{5}}{2})^n + D(\frac{1-\sqrt{5}}{2})^n$$

To figure out C and D, we substitute the value of a_0 and a_1:

$$0 = C + D$$

$$1 = C(\frac{1+\sqrt{5}}{2}) + D(\frac{1-\sqrt{5}}{2})$$

Revisiting Fibonacci Sequence

Solving these two equations, we get:

$$C = \frac{1}{\sqrt{5}}, D = -\frac{1}{\sqrt{5}}$$

Therefore:

$$a_n = C(\frac{1+\sqrt{5}}{2})^n + D(\frac{1-\sqrt{5}}{2})^n$$

$$= \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^n$$

General Homogenous Linear Difference Equations (Optional)

The constant coefficient linear difference equation of order r is

$$\boxed{a_0 u_{n+r} + a_1 u_{n+r-1} + \ldots + a_r u_n = \varphi(n)}$$

where the coefficients $a_0, \ldots, a_r$ are constants and the function $\varphi(n)$ is given; the equation is in homogeneous form if $\varphi(n) \equiv 0$, and we shall concentrate on the homogeneous case.

By trial, it is evident that $u_n = m^n$ is a solution for those values of m which satisfy the equation

$$m^n [a_0 m^r + a_1 m^{r-1} + \ldots + a_r] = 0$$

The solution m=0 is of no interest, and the above expression, disregarding m^n, is called the indicial equation. This r^{th} degree polynomial equation has r roots, and the form of the solution depends on whether the roots are distinct or repeated.

Distinct Roots (Optional)

Supposing the indicial equation has r distinct roots $m_1, m_2, \ldots, m_r$, then the general solution of the homogeneous equation is

$$\boxed{u_n = A_1 m_1^n + A_2 m_2^n + \ldots + A_r m_r^n}$$

where $A_1, \ldots, A_r$ are constants which may be chosen to satisfy any applicable boundary conditions.

Example Solve $u_{n+2} - 7u_{n+1} + 12u_n = 0$
subject to the boundary conditions $u_0 = 2$, $u_1 = 7$.

The indicial equation is

$$m^2 - 7m + 12 = 0$$

giving roots $m = 3, 4$, and so the general solution is

$$u_n = A_1 3^n + A_2 4^n$$

Applying the boundary conditions, we have $A_1 + A_2 = 2$ and $3A_1 + 4A_2 = 7$, which gives $A_1 = A_2 = 1$, and the required solution is

$$u_n = 3^n + 4^n$$

Multiple Roots (Optional)

Suppose that m_1 is a multiple root of multiplicity k, then the solution is

$$\boxed{u_n = (A_1 + A_2 n + \ldots + A_k n^{k-1}) m_1^n + A_{k+1} m_{k+1}^n + \ldots + A_r m_r^n}$$

Example Solve $u_{n+3} - 3u_{n+1} - 2u_n = 0$

The indicial equation is

$$m^3 - 3m - 2 = 0$$

$$\Rightarrow (m+1)^2(m-2) = 0$$

giving roots of -1 (twice), 2.

This gives a general solution of

$$u_n = (A_1 + A_2 n)(-1)^n + A_3 2^n$$

Quick Summary

Recursion is a very useful technique in computer science.

It is very important to learn to think recursively,

by reducing the problem into smaller problems.

This is an essential skill to acquire to become a professional programmer.

Make sure you have more practice in setting up recurrence relations and generating functions.