

GPS 5 due Mon Oct 5th

HW 5 due Tue Oct 6th (available by this evening)

Algorithms

Model: "Real RAM"

C-based language

Read about "describe an algorithm" on the website

- Unambiguous English description of how to solve the problem.

Consider pseudocode.

Not programming code!

- Run time (Big-Oh) -

- Follow rubrics.
- Greedy algorithms must have a formal proof of correctness.

Reductions: Solve an instance of problem A by using a black-box solution to some problem B.

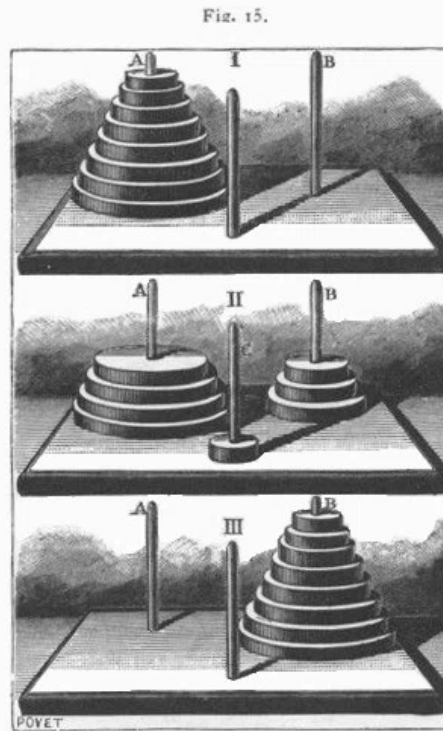
Recursion: Solve an instance of problem A by solving smaller instances of A.

Don't think about what is happening during recursive calls.

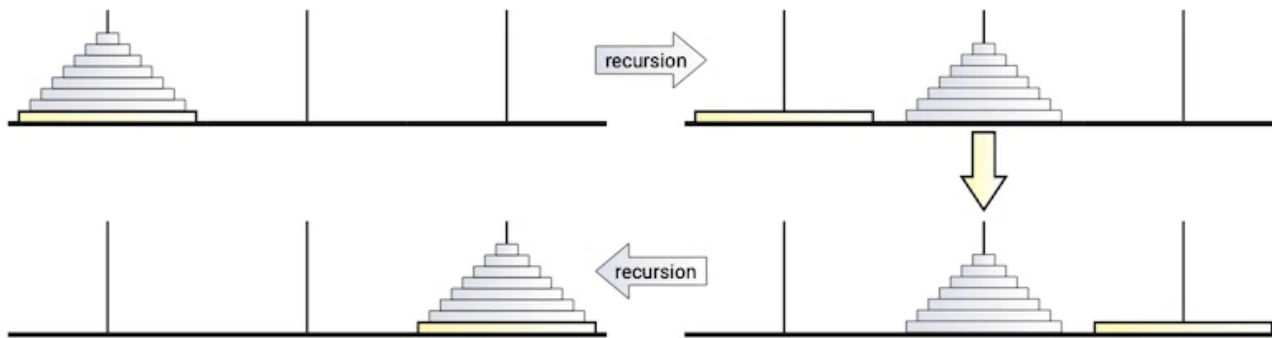
Trust the Recursion Fairy.

Tower of Hanoi

move disks between
pegs. one disk
at a time, always
stack big $\uparrow$ small.



La tour d'Hanoi.



Say we have n disks + three pegs
 src, dst, tmp .

Hanoi(n, src, dst, tmp):

if $n > 0$

Hanoi($n - 1, src, tmp, dst$)

move disk n from src to dst

Hanoi($n - 1, tmp, dst, src$)

moves n disk
(with disk n being
the largest)
from peg src to
peg dst .

Run time (here, # moves):

$T(n)$: # move for n disks

$$T(n) = \begin{cases} 0 & \text{if } n=0 \\ 2T(n-1) + 1 & \text{otherwise} \end{cases}$$

n	0	1	2	4
$T(n)$	0	1	3	7

Guess: $T(n) = 2^n - 1$

Proof: Let $n \geq 0$.

Assume $T(k) = 2^k - 1$ for all $0 \leq k < n$.

If $n = 0$, $T(n) = T(0) = 2^0 - 1 = 0$ ✓

If $n \geq 1$, then

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2^{n-1} - 1) + 1 \\ &= 2^n - 1 \end{aligned}$$

Sorting algorithms:

Given an array $A[1..n]$ of numbers/
chars/etc.

Want to modify A so its elements are
in increasing order.

Mergesort (von Neumann '45):

Input:	S	O	R	T	I	N	G	E	X	A	M	P	L	
Divide:	S	O	R	T	I	N		G	E	X	A	M	P	L
Recurse Left:	I	N	O	R	S	T		G	E	X	A	M	P	L
Recurse Right:	I	N	O	R	S	T		A	E	G	L	M	P	X
Merge:	A	E	G	I	L	M	N	O	P	R	S	T	X	

← merge two sorted subarrays
into one big sorted array

Do nothing if array too small to divide
into smaller
pieces.

```
MergeSort(A[1 .. n]):
```

```
  if  $n > 1$ 
```

```
     $m \leftarrow \lfloor n/2 \rfloor$ 
```

```
    MergeSort(A[1 .. m])
```

```
    MergeSort(A[m + 1 .. n])
```

```
    Merge(A[1 .. n], m)
```

← Sort $A[1..n]$ assuming
 $A[1..m]$ & $A[m+1..n]$ are
sorted.

Merge: grab smaller of $A[i]$ & $A[m+1]$...
& recursively sort what remains!

```
Merge(A[1 .. n], m):
```

```
   $i \leftarrow 1; j \leftarrow m + 1$ 
```

```
  for  $k \leftarrow 1$  to  $n$ 
```

```
    for  $j > n$ 
```

```
      if  $j > n$ 
```

```
         $B[k] \leftarrow A[i]; i \leftarrow i + 1$ 
```

```
      else if  $i > m$ 
```

```
         $B[k] \leftarrow A[j]; j \leftarrow j + 1$ 
```

```
      else if  $A[i] \leq A[j]$ 
```

```

else if  $A[i] < A[j]$ 
     $B[k] \leftarrow A[i]; i \leftarrow i + 1$ 
else
     $B[k] \leftarrow A[j]; j \leftarrow j + 1$ 
for  $k \leftarrow 1$  to  $n$ 
     $A[k] \leftarrow B[k]$ 

```

$T(n)$: time to mergesort an array of size n .

$$T(1) = T(0) = O(1)$$

$$T(n) = O(n) + T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)$$

$$\approx O(n) + 2T(n/2)$$

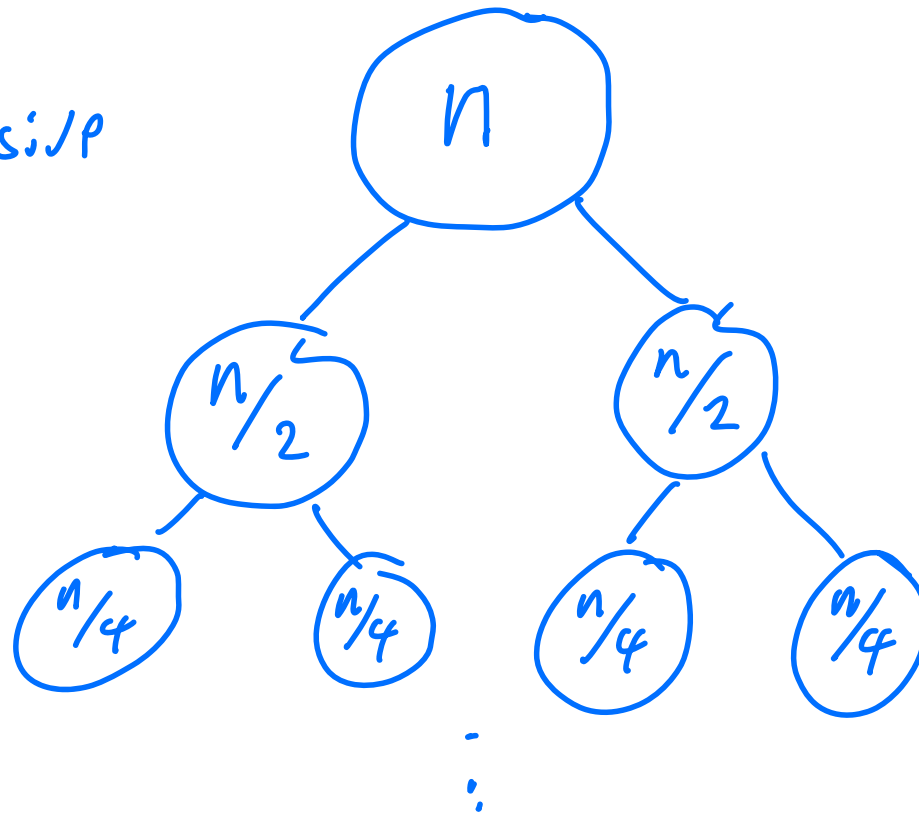
Recursion tree: rooted tree

each node is a subproblem

used when running the
algorithm

root: the top-level call

a child for
each recursive
call



$$\Sigma = n$$

$$\Sigma =$$

$$n$$

base
cases



to solve recurrence: each node gets a value τ
how long spent on that subproblem.

Total time: sum of the values

by summing the individual levels' sums

Merge sort: all levels = n

levels: # times we divide n by 2 = $\log_2 n$

$$T(n) = O(n \cdot \log_2 n) = O(n \log n)$$

Other typical cases: the level sums make
a decreasing geometric series

= $O(\text{value of biggest member}$
in geometric series)

= $O(\text{value at root})$

n
 $n/3$
 $n/9$
 $\vdots$

maybe an increasing series

= $O(\text{biggest term in series})$

= $O(\# \text{ leaves})$

$$\begin{array}{r} n \\ 3n \\ \hline 2 \\ 7n \\ \hline 2 \\ i \end{array}$$

Examples of 60th on Thursday!