CS 344: Design & Analysis of Algorithms

Lecture 4

Sep 12, 2019

Maximum subarray problem

Given an array, find a non-empty contiguous subarray that maximizes its sum:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

maximum subarray

Maximum subarray problem

4 -6 7 2 5 -1 3 2

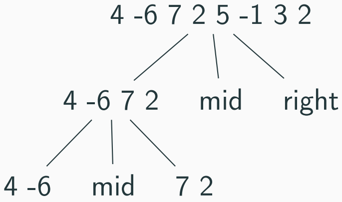
left mid right

Maximum subarray problem

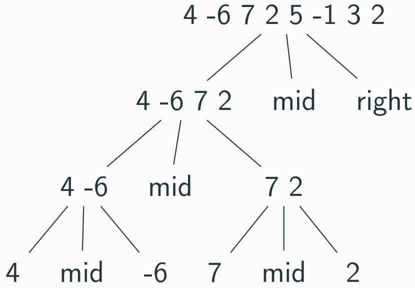
4 -6 7 2 5 -1 3 2

4 -6 7 2 mid right

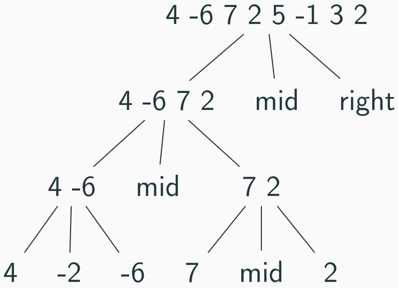
Maximum subarray problem



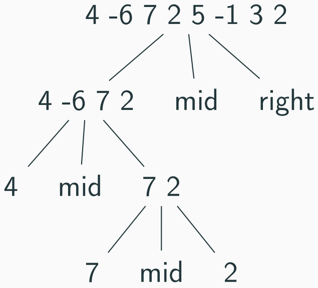
Maximum subarray problem



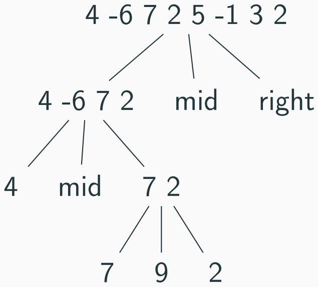
Maximum subarray problem



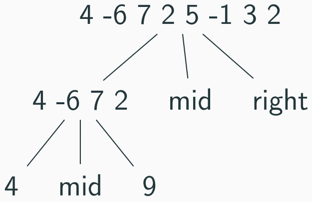
Maximum subarray problem



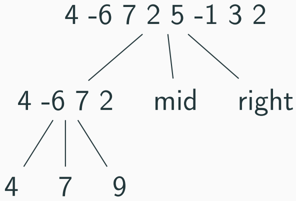
Maximum subarray problem



Maximum subarray problem



Maximum subarray problem



Maximum subarray problem

4 -6 7 2 5 -1 3 2

9 (7 2)   mid   right

Maximum subarray problem

4 -6 7 2 5 -1 3 2

9 (7 2)   mid   5 -1 3 2

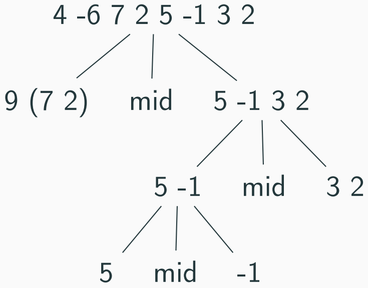
Maximum subarray problem

4 -6 7 2 5 -1 3 2

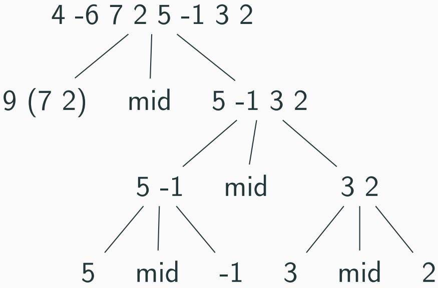
9 (7 2) mid 5 -1 3 2

5 -1 mid 3 2

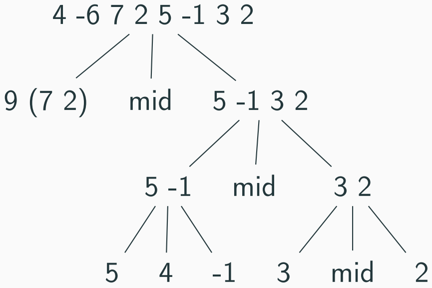
Maximum subarray problem



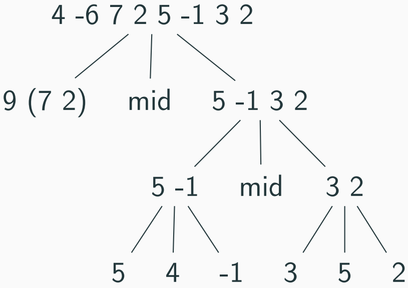
Maximum subarray problem



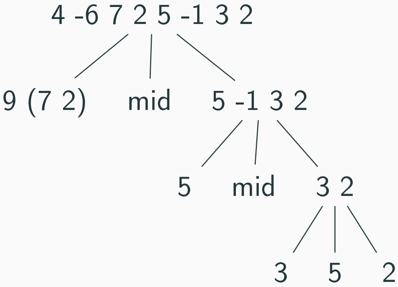
Maximum subarray problem



Maximum subarray problem



Maximum subarray problem



Maximum subarray problem

4 -6 7 2 5 -1 3 2

9 (7 2) mid 5 -1 3 2

5 mid 5

Maximum subarray problem

4 -6 7 2 5 -1 3 2

9 (7 2) mid 5 -1 3 2

5 9 5

Maximum subarray problem

4 -6 7 2 5 -1 3 2

9 (7 2)   mid   9 (5 -1 3 2)

Maximum subarray problem

4 -6 7 2 5 -1 3 2

9 (7 2)   18 (7 2 5 -1 3 2)   9 (5 -1 3 2)

Master theorem

Divide and conquer solves a problem of size n by:

- splitting it into a subproblems of size n/b
- combining the answers in $O(n^d)$ time

where $a, b, d > 0$

Master theorem

If $T(n) = aT(n/b) + O(n^d)$ and $a > 0, b > 1, d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

Maximum subarray problem

if *high* == *low*
    return (*low*, *high*, $A[low]$)
else $mid = \lfloor (low + high)/2 \rfloor$
    ($left\text{-}low$, $left\text{-}high$, $left\text{-}sum$) $=$
        FIND-MAXIMUM-SUBARRAY(A, low, mid)
    ($right\text{-}low$, $right\text{-}high$, $right\text{-}sum$) $=$
        FIND-MAXIMUM-SUBARRAY(A, $mid + 1$, $high$)
    ($cross\text{-}low$, $cross\text{-}high$, $cross\text{-}sum$) $=$
        FIND-MAX-CROSSING-SUBARRAY(A, low, mid, $high$)
    if $left\text{-}sum \geq right\text{-}sum$ and $left\text{-}sum \geq cross\text{-}sum$
        return ($left\text{-}low$, $left\text{-}high$, $left\text{-}sum$)
    elseif $right\text{-}sum \geq left\text{-}sum$ and $right\text{-}sum \geq cross\text{-}sum$
        return ($right\text{-}low$, $right\text{-}high$, $right\text{-}sum$)
    else return ($cross\text{-}low$, $cross\text{-}high$, $cross\text{-}sum$)

Maximum subarray problem

Step	Time
Solve the midpoint case	$O(n)$
Solve the left and right cases	$T(n/2)$ each
Find the max of those three	$O(1)$

Maximum subarray problem

We get the following recurrence:

$$T(n) = 2T(n/2) + O(n)$$

To match $T(n) = aT(n/b) + O(n^d)$:

- $a = 2$
- $b = 2$
- $d = 1$

Maximum subarray problem

If $T(n) = aT(n/b) + O(n^d)$ and $a > 0, b > 1, d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

- $a = 2$
- $b = 2$
- $d = 1$

$$\log_b a = \log_2 2 = 1 = d$$

Use the second case.

Maximum subarray problem

If $T(n) = aT(n/b) + O(n^d)$ and $a > 0, b > 1, d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

- $a = 2$
- $b = 2$
- $d = 1$

$$O(n^d \log n) = O(n^1 \log n) = O(n \log n)$$

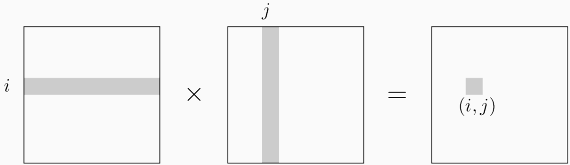
Matrix multiplication

If A and B are $n \times n$ matrices, we can multiply them to produce a $n \times n$ matrix C:

$$A \cdot B = C$$

then c_{ij} is the dot product of row i of A and column j of B:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$



Matrix multiplication

Naïve method:

```
for i from 1 to n:
    for j from 1 to n:
        c[i][j] = 0.0
        for k from 1 to n:
            c[i][j] += a[i][k] * b[k][j]
```

How long does this take?

Matrix multiplication

Can we view this as a divide and conquer algorithm?

Suppose we decompose each matrix into four $n/2 \times n/2$ blocks:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Matrix multiplication

Then we can define multiplication as such:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Matrix multiplication

So we have these subproblems to compute:

$$
\begin{aligned}
C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\
C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\
C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\
C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}
\end{aligned}
$$

Matrix multiplication

$n = A.rows$
let C be a new $n \times n$ matrix
if $n == 1$
    $c_{11} = a_{11} \cdot b_{11}$
else partition A, B, and C as in equations (4.9)
    $C_{11} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{11})$
        $+ \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{21})$
    $C_{12} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{12})$
        $+ \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{22})$
    $C_{21} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{11})$
        $+ \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{21})$
    $C_{22} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{12})$
        $+ \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{22})$
return C

Matrix multiplication

- Divide into 8 blocks of size $n/2 \times n/2$
- Recursively multiply
- Add (some of) the resulting matrices

Matrix multiplication

Step	Time
Divide into 8 blocks of size $n/2 \times n/2$	$O(1)$
Recursively multiply	$T(n/2)$ each
Add resulting matrices	$O(n^2)$

Matrix multiplication

Then the time required has the form:

$$T(n) = 8T(n/2) + O(n^2)$$

Matrix multiplication

$$T(n) = 8T(n/2) + O(n^2)$$

Master theorem: $T(n) = aT(n/b) + O(n^d)$

- $a = 8$
- $b = 2$
- $d = 2$

Matrix multiplication

If $T(n) = aT(n/b) + O(n^d)$ and $a > 0, b > 1, d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

- $a = 8$
- $b = 2$
- $d = 2$

$$\log_b a = log_2 8 = 3 > 2 = d$$

Use case 3.

Matrix multiplication

If $T(n) = aT(n/b) + O(n^d)$ and $a > 0, b > 1, d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

- $a = 8$
- $b = 2$
- $d = 2$

$$O(n^{\log_b a}) = O(n^3)$$

Matrix multiplication

So matrix multiplication is $O(n^3)$.

Is n^3 also a lower bound?

(Is it also $\Omega(n^3)$, and thus $\Theta(n^3)$?)

Actually, there is a clever rearrangement that can do better!

It was found in 1969 by Volker Strassen, and is known as Strassen's method.

Matrix multiplication

"Strassen's method is not at all obvious."

– CLRS

There are four steps to Strassen's method:

- Divide the matrices into $n/2 \times n/2$ matrices as before
- Add/subtract some of these to create 10 new temporary matrices $S_1, \ldots, S_{10}$
- Compute 7 matrix products $P_1, \ldots, P_7$
- Compute submatrices $C_{11}, C_{12}, C_{21}, C_{22}$

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Strassen's method

First compute the S matrices:

$$
\begin{aligned}
S_1 &= B_{12} - B_{22} , \\
S_2 &= A_{11} + A_{12} , \\
S_3 &= A_{21} + A_{22} , \\
S_4 &= B_{21} - B_{11} , \\
S_5 &= A_{11} + A_{22} , \\
S_6 &= B_{11} + B_{22} , \\
S_7 &= A_{12} - A_{22} , \\
S_8 &= B_{21} + B_{22} , \\
S_9 &= A_{11} - A_{21} , \\
S_{10} &= B_{11} + B_{12} .
\end{aligned}
$$

Strassen's method

Then compute the P matrices:

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 &&= A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \,, \\
P_2 &= S_2 \cdot B_{22} &&= A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \,, \\
P_3 &= S_3 \cdot B_{11} &&= A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \,, \\
P_4 &= A_{22} \cdot S_4 &&= A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \,, \\
P_5 &= S_5 \cdot S_6 &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \,, \\
P_6 &= S_7 \cdot S_8 &&= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22} \,, \\
P_7 &= S_9 \cdot S_{10} &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12} \,.
\end{aligned}
$$

Finally, combine these to get the C submatrices:

$$
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6 \\
C_{12} &= P_1 + P_2 \\
C_{21} &= P_3 + P_4 \\
C_{22} &= P_1 + P_5 - P_3 - P_7
\end{aligned}
$$

Strassen's method

Expanding C_{11}:

$$
\begin{array}{lllllll}
A_{11}\cdot B_{11} & + A_{11}\cdot B_{22} & + A_{22}\cdot B_{11} & + A_{22}\cdot B_{22} & & & \\
 & & - A_{22}\cdot B_{11} & & + A_{22}\cdot B_{21} & & \\
 & - A_{11}\cdot B_{22} & & & & - A_{12}\cdot B_{22} & \\
 & & & - A_{22}\cdot B_{22} & - A_{22}\cdot B_{21} & + A_{12}\cdot B_{22} & + A_{12}\cdot B_{21} \\
\hline
A_{11}\cdot B_{11} & & & & & & + A_{12}\cdot B_{21}\ ,
\end{array}
$$

Strassen's method

Expanding C_{12}:

$$
\begin{array}{l}
A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\
\qquad\qquad + A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\
\hline
A_{11} \cdot B_{12} \qquad\qquad\qquad + A_{12} \cdot B_{22} \,,
\end{array}
$$

Expanding C_{21}:

$$
\begin{array}{l}
A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\
\qquad\qquad\; - A_{22} \cdot B_{11} + A_{22} \cdot B_{21} \\
\hline
A_{21} \cdot B_{11} \qquad\qquad\qquad\;\; + A_{22} \cdot B_{21} \;,
\end{array}
$$

Expanding C_{22}:

$$
\begin{array}{rrrrrrr}
A_{11}\cdot B_{11} & + A_{11}\cdot B_{22} & + A_{22}\cdot B_{11} & + A_{22}\cdot B_{22} & & & \\
 & - A_{11}\cdot B_{22} & & & + A_{11}\cdot B_{12} & & \\
 & & - A_{22}\cdot B_{11} & & & - A_{21}\cdot B_{11} & \\
- A_{11}\cdot B_{11} & & & & - A_{11}\cdot B_{12} & + A_{21}\cdot B_{11} & + A_{21}\cdot B_{12} \\
\hline
 & & & A_{22}\cdot B_{22} & & & + A_{21}\cdot B_{12}\ ,
\end{array}
$$

Strassen's method

Step	Time
Divide the matrices	$O(1)$
Add/subtract to get $S_1, \ldots, S_{10}$	$O(n^2)$
Multiply to get $P_1, \ldots, P_7$	$T(n/2)$ each
Compute $C_{11}, C_{12}, C_{21}, C_{22}$	$O(n^2)$

Strassen's method

So the overall time is

$$T(n) = 7T(n/2) + O(n^2)$$

Strassen's method

$$T(n) = 7T(n/2) + O(n^2)$$

- $a = 7$
- $b = 2$
- $d = 2$

$$\log_2 7 \approx 2.807 > d$$

So using case 3 of master theorem, $T(n) = O(n^{log_2 7}) \approx O(n^{2.807})$

Matrix multiplication

So is matrix multiplication $\Theta(n^{2.807})$?

Let ω denote the exponent.

- $2 \leq \omega \leq 3$ (why?)
- $O(n^{2.807})$ (Strassen, 1969)
- $O(n^{2.376})$ algorithm found in 1990 (Coopersmith-Winograd)
- $O(n^{2.3729})$ in 2013 (Virginia Williams)
- $O(n^{2.3728639})$ in 2014 (François Le Gall)

$2 \leq \omega < 2.373$, but we still don't know if $\omega = 2$ is possible!

Another clever rearrangement

Suppose we have two complex numbers, $a + bi$, and $c + di$.

We can multiply them as such:

$$\begin{aligned}(a + bi)(c + di) &= ac + adi + bci + bdi^2 \\ &= ac + (ad + bc)i - bd \\ &= ac - bd + (ad + bc)i\end{aligned}$$

Another clever rearrangement

$$(a + bi)(c + di) = ac - bd + (ad + bc)i$$

So we have to do four multiplications.

Another clever rearrangement

$$(a + bi)(c + di) = ac - bd + (ad + bc)i$$

But Gauss noticed that

$$ad + bc = (a + b)(c + d) - ac - bd$$

Another clever rearrangement

$$
\begin{aligned}
(a + bi)(c + di) &= ac - bd + (ad + bc)i \\
ad + bc &= (a + b)(c + d) - ac - bd
\end{aligned}
$$

So we can compute this with only three multiplications!

Another clever rearrangement

Both are $\Theta(1)$, so does this matter?

Let's consider multiplying n-bit binary numbers...

Multiplying n-bit numbers

Let's consider multiplying n-bit binary numbers x and y.

And let's assume n is a power of 2, for simplicity.

Can we divide and conquer this problem?

Multiplying n-bit numbers

Divide: split x and y into two pieces of length $n/2$

$$x = \boxed{\quad x_L \quad}\ \boxed{\quad x_R \quad} = 2^{n/2} x_L + x_R$$

$$y = \boxed{\quad y_L \quad}\ \boxed{\quad y_R \quad} = 2^{n/2} y_L + y_R.$$

E.g., if $x = 10110110_2$, this is also equal to $2^4 \cdot 1011_2 + 0110_2$.

Multiplying n-bit numbers

Divide: split x and y into two pieces of length $n/2$

$$\begin{aligned}(2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) &= 2^n x_L y_L + 2^{n/2} x_R y_L + 2^{n/2} x_L y_R + x_R y_R \\ &= 2^n x_L y_L + 2^{n/2}(x_R y_L + x_L y_R) + x_R y_R\end{aligned}$$

We need four recursive multiplications.

Multiplying n-bit numbers

This gives a running time of

$$T(n) = 4T(n/2) + O(n)$$

which is $O(n^2)$ by the master method.

Multiplying n-bit numbers

Let's try Gauss's trick:

$$\begin{aligned}(2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) &= 2^n x_L y_L + 2^{n/2}(x_R y_L + x_L y_R) + x_R y_R \\ x_R y_L + x_L y_R &= (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R\end{aligned}$$

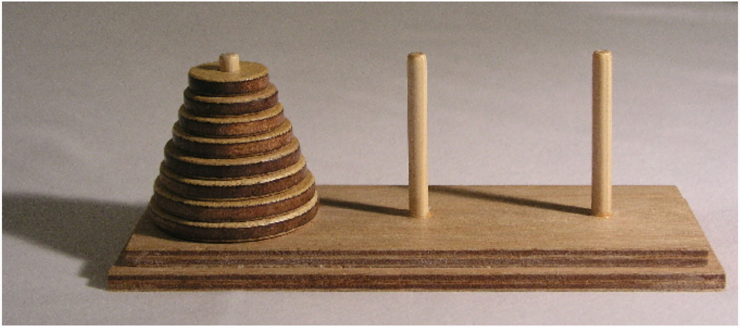
Multiplying n-bit numbers

Now with only three recursive multiplications, this gives a running time of

$$T(n) = 3T(n/2) + O(n)$$

which is $O(n^{log_2 3}) \approx O(n^{1.59})$.

Tower of Hanoi



The goal is to move all discs from peg A to C, but a larger disc can never go on top of a smaller one.

Image credit: Evanherk

https://commons.wikimedia.org/wiki/File:Tower_of_Hanoi.jpeg

Tower of Hanoi

The idea behind the recursive solution:

- If we could somehow get all $n - 1$ discs from A to B, then we could move the largest disc to C.
- Then we could move all $n - 1$ discs from B to C.

Tower of Hanoi

```
moveDiscs(src, dest, n):
    if n == 1:
        move top disc from src to dest
    otherwise:
        moveDiscs(src, otherPeg, n - 1)
        move remaining disc from src to dest
        moveDiscs(otherPeg, dest, n - 1)
```

Tower of Hanoi

Step	Time
Move from src to other	$T(n-1)$
Move one disc	$O(1)$
Move from other to dest	$T(n-1)$

Tower of Hanoi

So we get a time of

$$T(n) = 2T(n-1) + O(1)$$

But it's not in the form of the master theorem. Now what?

Tower of Hanoi

We can solve this using another method, called recursion trees.

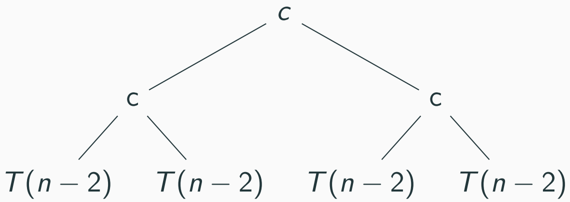
$$T(n)$$

Tower of Hanoi

We can solve this using another method, called recursion trees.

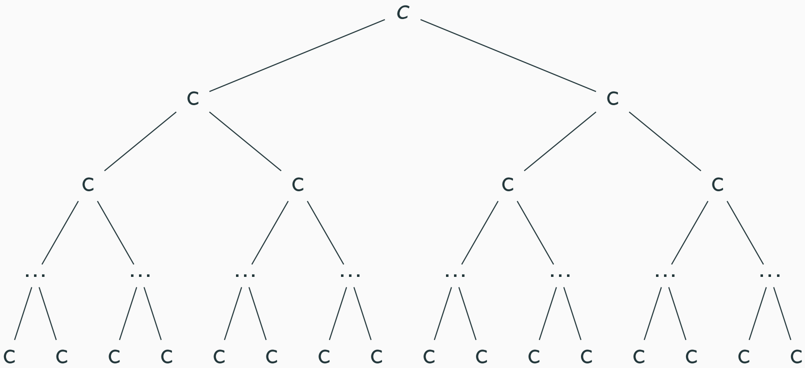
$$
\begin{tikzcd}[column sep=small]
 & c \arrow[dl, dash] \arrow[dr, dash] & \\
T(n-1) & & T(n-1)
\end{tikzcd}
$$

Expanding one level:



Continue until we reach the leaves.

Tower of Hanoi



- How many levels are there?
- How much work is done at each level?
- What is the sum of all levels?

Tower of Hanoi

- How much work is done at each level?
  - 2^k nodes, each with c work, so $c2^k$
- What is the sum of all levels?

$$\sum_{k=0}^{n-1} c2^k = c(2^n - 1)$$

This implies that the recursive algorithm takes $O(2^n)$ steps.

Tower of Hanoi

Another way to see this:

n	#steps
1	1
2	3
3	7
4	15

Tower of Hanoi

We can guess that this is $2^n - 1 = O(2^n)$, but we have to prove it.

Use induction!

Claim: The recursive algorithm takes $2^n - 1$ steps.

- Base case: $n = 1$, easy to check.
- Inductive case:
  - Assume $T(n-1) = 2^{n-1} - 1$
  - Use the inductive hypothesis to prove $T(n) = 2^n - 1$:

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

Summary

- Divide and conquer is a powerful tool for designing algorithms
- Often get a recurrence of the form $T(n) = aT(n/b) + O(n^d)$
- If so, use the master theorem
- If not, draw the recursion tree and sum the work at each node
- Sometimes you'll get a general answer with some coefficients
- Use induction to solve and verify