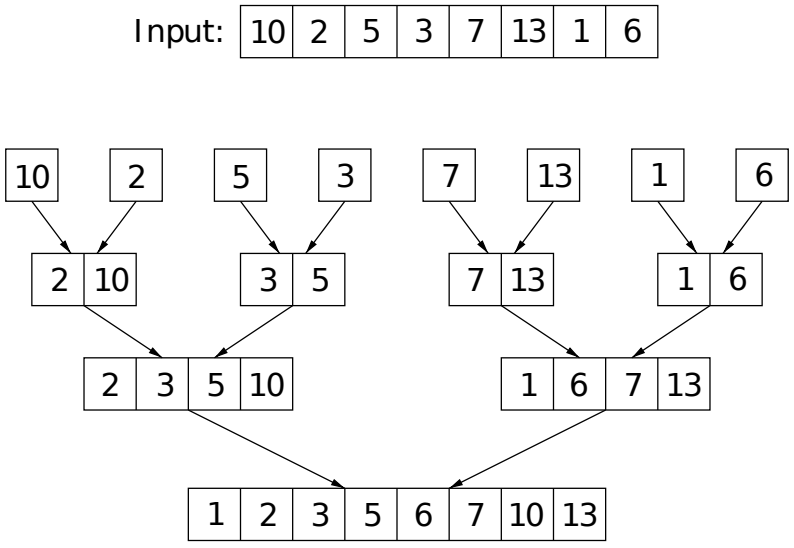
CS3383 Unit 1 Lecture 1: Divide and Conquer intro

David Bremner David Bremner

2024-01-12

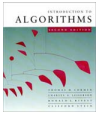
Outline

Merge Sort


Input: 10 2 5 3 7 13 1 6
10 2 5 3 7 13 1 6
2 10 3 5 7 13 1 6
2 3 5 10 1 6 7 13
1 2 3 5 6 7 10 13

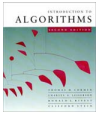
Merge Sort

```
def merge_sort(A):
    n = len(A)
    if (n <= 1):
        return A

    split = max(n//2,1)
    left=merge_sort(A[0:split])
    right=merge_sort(A[split:])
    return merge(left,right)
```

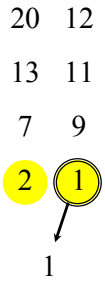


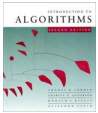
Merging two sorted arrays

20	12
13	11
7	9
2	1

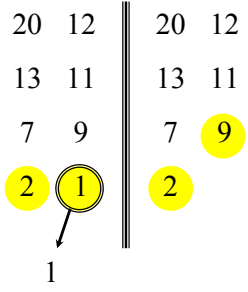


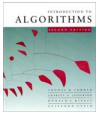
Merging two sorted arrays



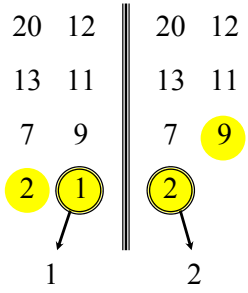


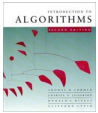
Merging two sorted arrays



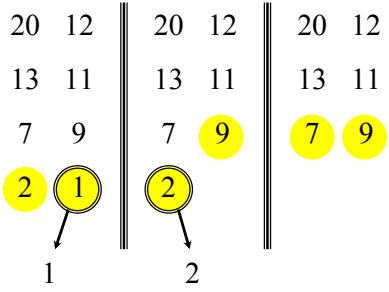


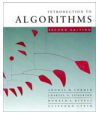
Merging two sorted arrays



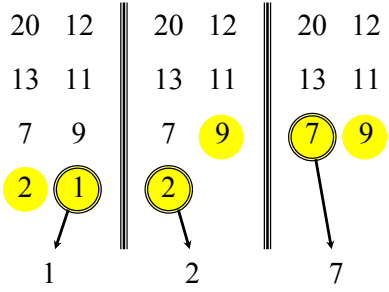


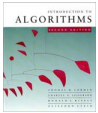
Merging two sorted arrays



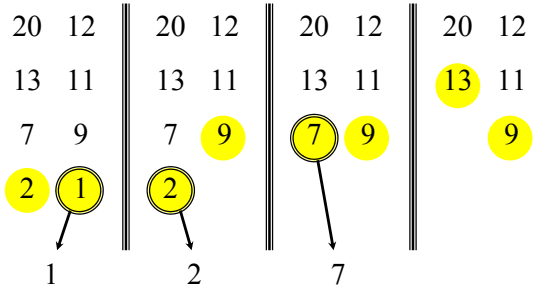


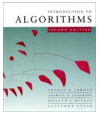
Merging two sorted arrays



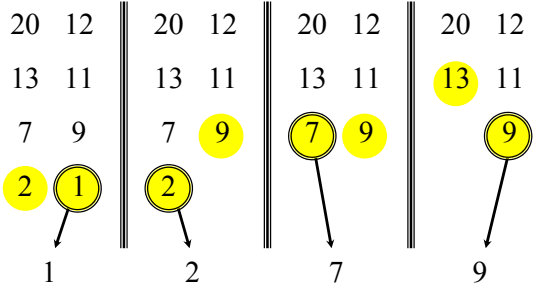


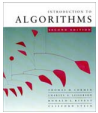
Merging two sorted arrays



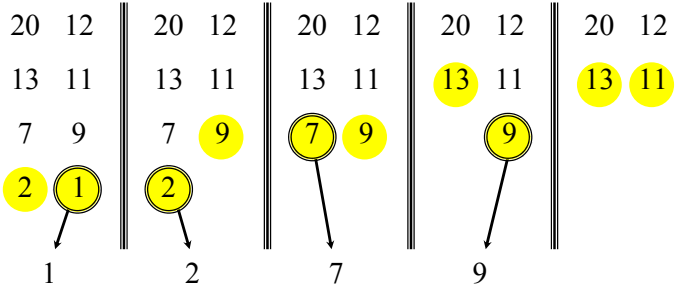


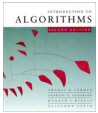
Merging two sorted arrays



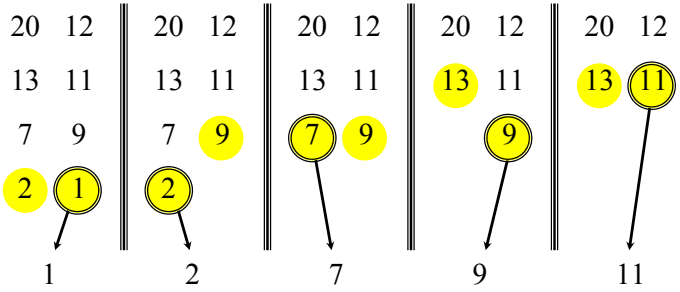


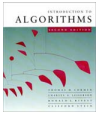
Merging two sorted arrays



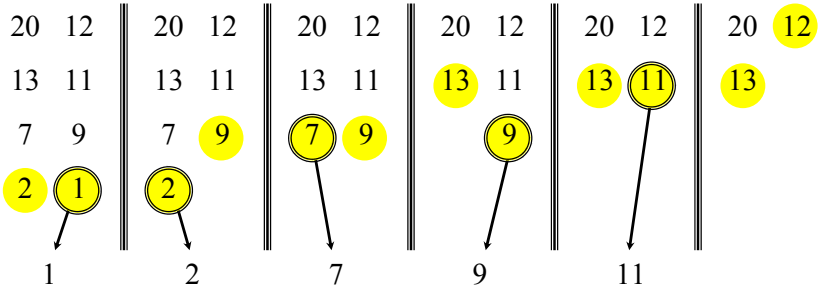


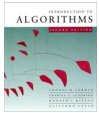
Merging two sorted arrays



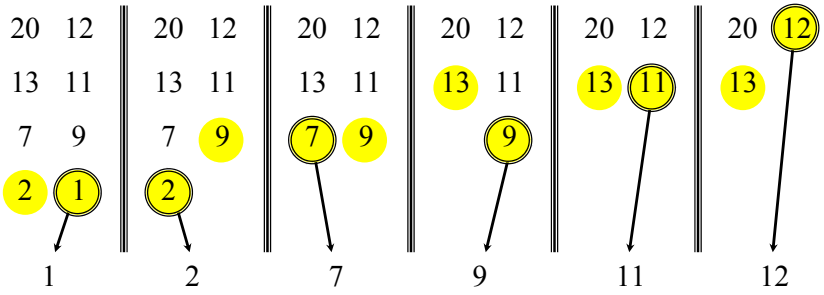


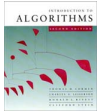
Merging two sorted arrays



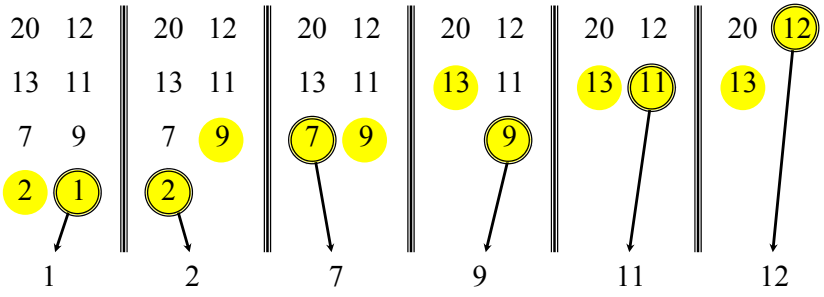


Merging two sorted arrays





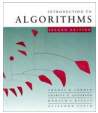
Merging two sorted arrays



Time = $\Theta(n)$ to merge a total of n elements (linear time).

Analyzing Merge sort

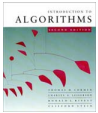
```
def merge_sort(A):
    n = len(A)                          # Θ(1)
    if (n <= 1):                        # Θ(1)
        return A                        # Θ(1)
    split = max(n//2,1)                 # Θ(1)

    left=merge_sort(A[0:split])         # T(n/2)
    right=merge_sort(A[split:])         # T(n/2)
    return merge(left,right)            # Θ(n)
```



Recurrence for merge sort

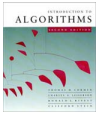
$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

- We shall usually omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small n, but only when it has no effect on the asymptotic solution to the recurrence.
- We will see several ways starting with "Rec. Tree" to find a good upper bound on $T(n)$.



Recursion tree

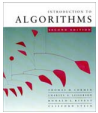
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



Recursion tree

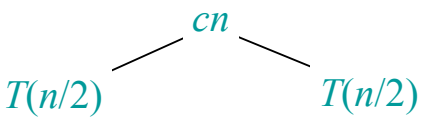
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

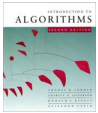
$T(n)$



Recursion tree

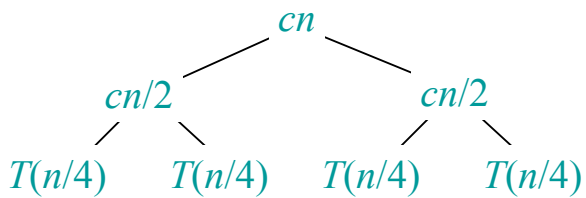
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

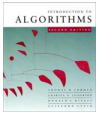




Recursion tree

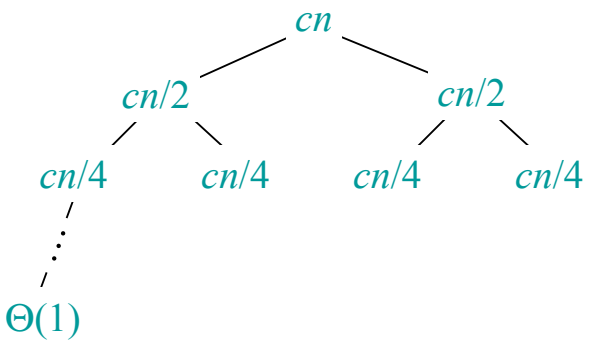
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

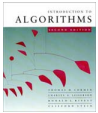




Recursion tree

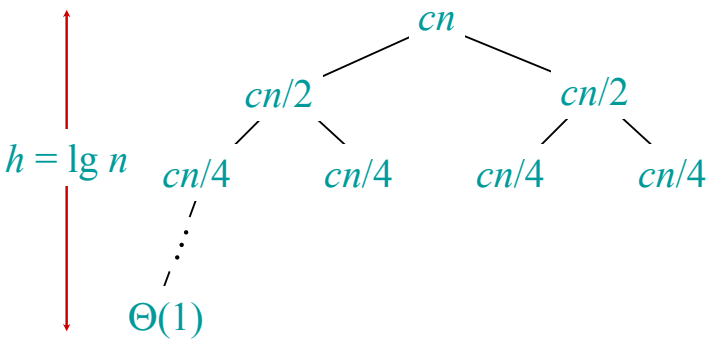
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

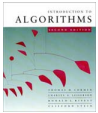




Recursion tree

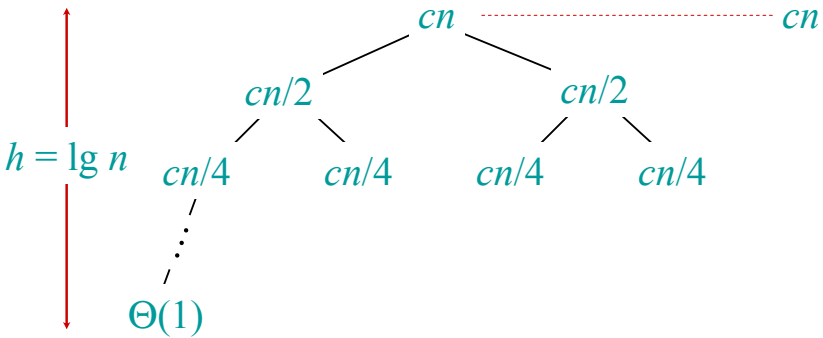
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

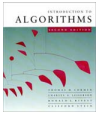




Recursion tree

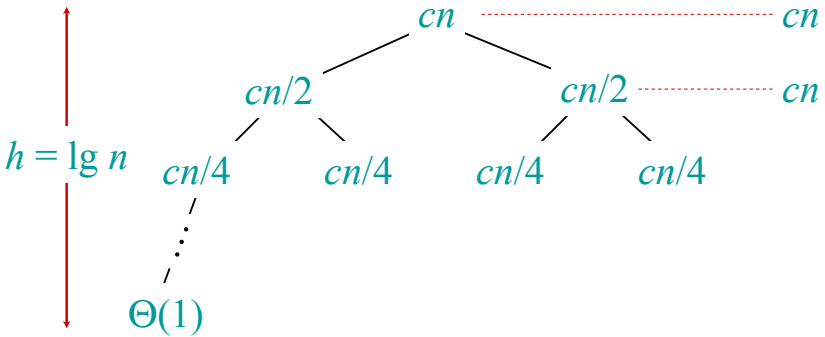
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

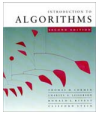




Recursion tree

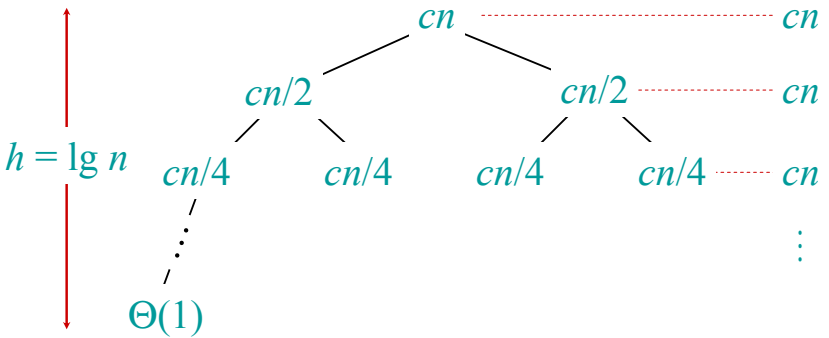
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

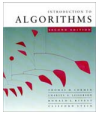




Recursion tree

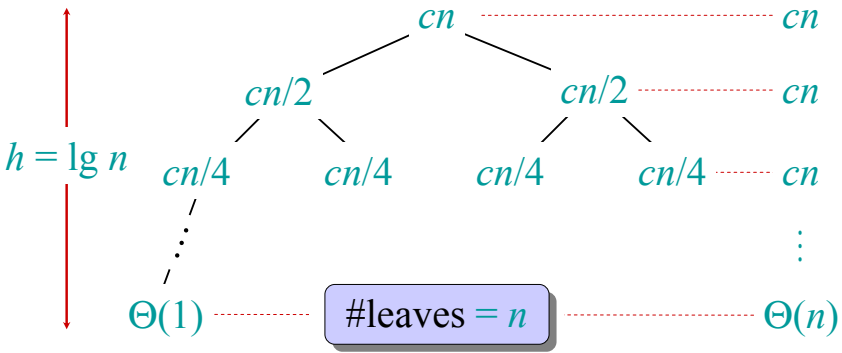
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

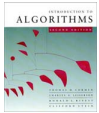




Recursion tree

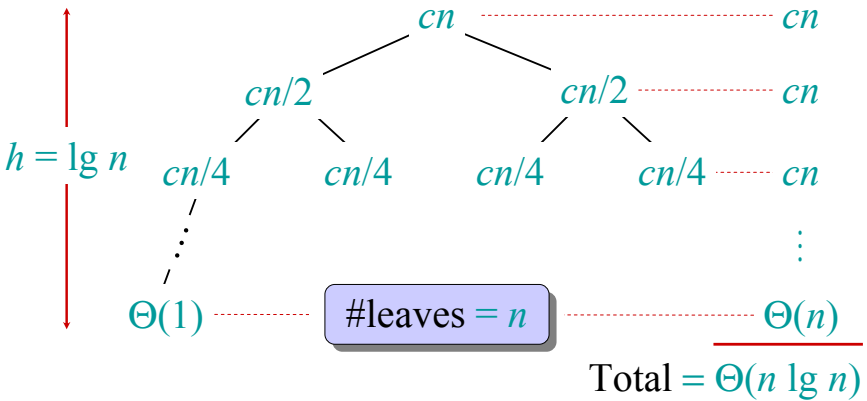
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

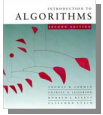




Recursion tree

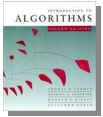
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.





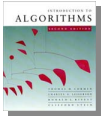
Recursion-tree method

- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion-tree method can be unreliable, just like any method that uses ellipses (…).
- The recursion-tree method promotes intuition, however.
- The recursion tree method is good for generating guesses for the substitution method.



Example of recursion tree

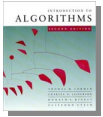
Solve $T(n) = T(n/4) + T(n/2) + n^2$:



Example of recursion tree

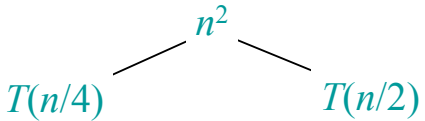
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

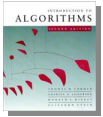
$T(n)$



Example of recursion tree

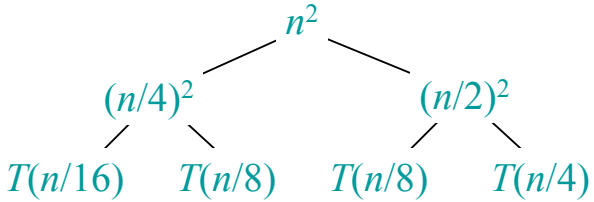
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

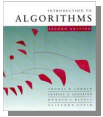




Example of recursion tree

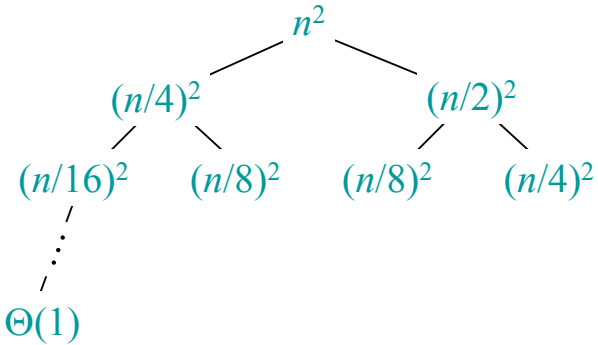
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

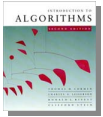




Example of recursion tree

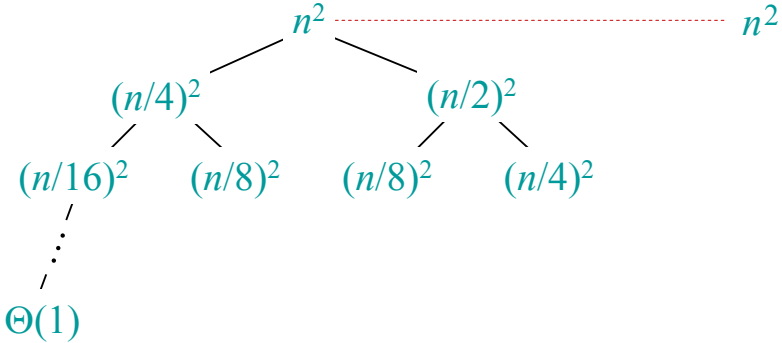
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

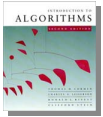




Example of recursion tree

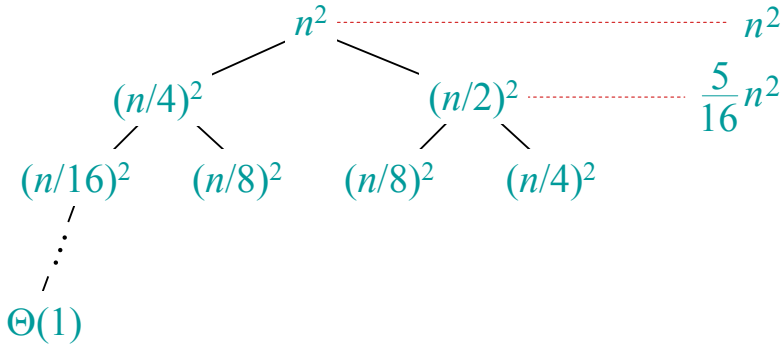
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

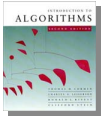




Example of recursion tree

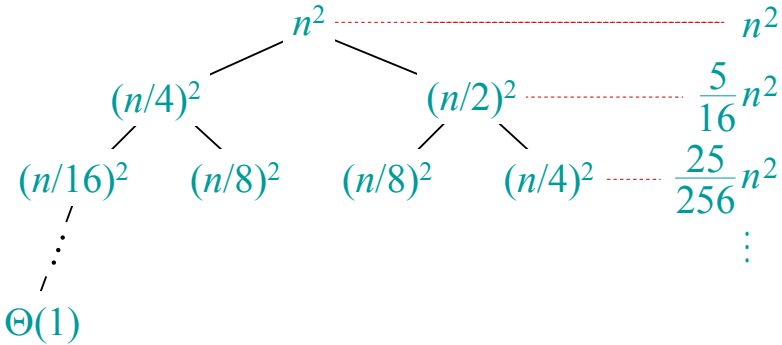
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

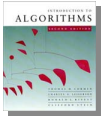




Example of recursion tree

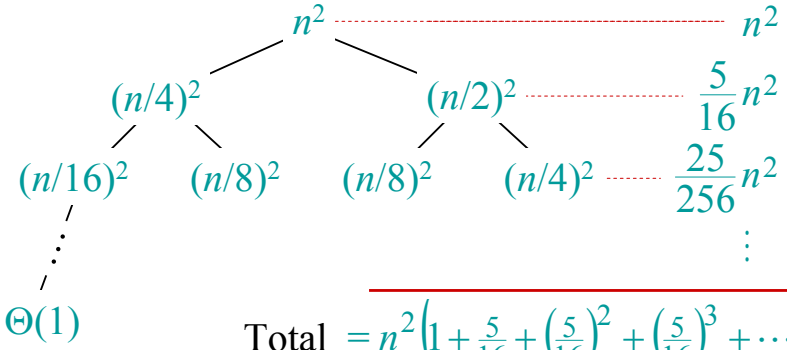
Solve $T(n) = T(n/4) + T(n/2) + n^2$:



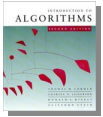


Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



$$\text{Total} = n^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \cdots\right)$$
$$= \Theta(n^2) \quad \textit{geometric series}$$



Appendix: geometric series

$$1 + x + x^2 + \cdots + x^n = \frac{1 - x^{n+1}}{1 - x} \quad \text{for } x \neq 1$$

$$1 + x + x^2 + \cdots = \frac{1}{1 - x} \quad \text{for } |x| < 1$$

Return to last slide viewed.

Naive D&C multiplication

Solve $T(n) = 4T(\frac{n}{2}) + cn$, via recursion tree.

Naive D&C multiplication

Solve $T(n) = 4T(\frac{n}{2}) + cn$, via recursion tree.

$$T(n) = \sum_{i=0}^{\lg n} c \cdot \frac{n}{2^i} \cdot 4^i$$

Naive D&C multiplication

Solve $T(n) = 4T(\frac{n}{2}) + cn$, via recursion tree.

$$T(n) = \sum_{i=0}^{\lg n} c \cdot \frac{n}{2^i} \cdot 4^i$$

$$= cn \cdot \sum_{i=0}^{\lg n} \frac{4^i}{2^i} = cn \cdot \sum_{i=0}^{\lg n} 2^i$$

Naive D&C multiplication

Solve $T(n) = 4T(\frac{n}{2}) + cn$, via recursion tree.

$$T(n) = \sum_{i=0}^{\lg n} c \cdot \frac{n}{2^i} \cdot 4^i$$

$$= cn \cdot \sum_{i=0}^{\lg n} \frac{4^i}{2^i} = cn \cdot \sum_{i=0}^{\lg n} 2^i$$

$$= cn \cdot \frac{2^{\lg n+1} - 1}{2 - 1} = 2cn \cdot 2^{\lg n} - cn$$

Naive D&C multiplication

Solve $T(n) = 4T(\frac{n}{2}) + cn$, via recursion tree.

$$T(n) = \sum_{i=0}^{\lg n} c \cdot \frac{n}{2^i} \cdot 4^i$$

$$= cn \cdot \sum_{i=0}^{\lg n} \frac{4^i}{2^i} = cn \cdot \sum_{i=0}^{\lg n} 2^i$$

$$= cn \cdot \frac{2^{\lg n+1} - 1}{2 - 1} = 2cn \cdot 2^{\lg n} - cn$$

$$= 2cn \cdot n - cn = 2cn^2 - cn \in \Theta(n^2)$$

Smart D&C multiplication, sloppy analysis

Solve $T(n) = 3T(\frac{n}{2}) + cn$, via recursion tree.

Smart D&C multiplication, sloppy analysis

Solve $T(n) = 3T(\frac{n}{2}) + cn$, via recursion tree.

$$T(n) = \sum_{i=0}^{\lg n} c \cdot \frac{n}{2^i} \cdot 3^i$$

Smart D&C multiplication, sloppy analysis

Solve $T(n) = 3T(\frac{n}{2}) + cn$, via recursion tree.

$$T(n) = \sum_{i=0}^{\lg n} c \cdot \frac{n}{2^i} \cdot 3^i$$

$$= cn \cdot \sum_{i=0}^{\lg n} \frac{3^i}{2^i} = cn \cdot \sum_{i=0}^{\lg n} (3/2)^i$$

Smart D&C multiplication, sloppy analysis

Solve $T(n) = 3T(\frac{n}{2}) + cn$, via recursion tree.

$$T(n) = \sum_{i=0}^{\lg n} c \cdot \frac{n}{2^i} \cdot 3^i$$

$$= cn \cdot \sum_{i=0}^{\lg n} \frac{3^i}{2^i} = cn \cdot \sum_{i=0}^{\lg n} (3/2)^i$$

$$= cn \cdot \frac{(3/2)^{\lg n+1} - 1}{(3/2) - 1} = 3cn \cdot (3/2)^{\lg n} - 2cn$$

Smart D&C multiplication, sloppy analysis

Solve $T(n) = 3T(\frac{n}{2}) + cn$, via recursion tree.

$$T(n) = \sum_{i=0}^{\lg n} c \cdot \frac{n}{2^i} \cdot 3^i$$

$$= cn \cdot \sum_{i=0}^{\lg n} \frac{3^i}{2^i} = cn \cdot \sum_{i=0}^{\lg n} (3/2)^i$$

$$= cn \cdot \frac{(3/2)^{\lg n+1} - 1}{(3/2) - 1} = 3cn \cdot (3/2)^{\lg n} - 2cn$$

$$= 3cn \cdot n^{\lg 3/2} - 2cn \in O(n^{1.6})$$

recursion tree example I/II

$$T(n) = 2T(3n/8) + n^2$$

recursion tree example I/II

$$T(n) = 2T(3n/8) + n^2$$

$$n^2, (3n/8)^2, [(3/8)^2 n]^2, [(3/8)^3 n]^2, \ldots, (3/8)^{2k} n^2$$

(call at depth k)

recursion tree example I/II

$$T(n) = 2T(3n/8) + n^2$$

$$n^2, (3n/8)^2, [(3/8)^2 n]^2, [(3/8)^3 n]^2, \ldots, (3/8)^{2k} n^2$$

(call at depth k)

There are 2^k nodes on level k, so work on level k is

$$2^k \cdot (3/8)^{2k} n^2 =$$

$$=$$

$$=$$

recursion tree example I/II

$$T(n) = 2T(3n/8) + n^2$$

$$n^2, (3n/8)^2, [(3/8)^2 n]^2, [(3/8)^3 n]^2, \ldots, (3/8)^{2k} n^2$$

(call at depth k)

There are 2^k nodes on level k, so work on level k is

$$2^k \cdot (3/8)^{2k} n^2 = 2^k \frac{3^{2k}}{2^{6k}} n^2$$

$$=$$

$$=$$

recursion tree example I/II

$$T(n) = 2T(3n/8) + n^2$$

$$n^2, (3n/8)^2, [(3/8)^2 n]^2, [(3/8)^3 n]^2, \ldots, (3/8)^{2k} n^2$$

(call at depth k)

There are 2^k nodes on level k, so work on level k is

$$2^k \cdot (3/8)^{2k} n^2 = 2^k \frac{3^{2k}}{2^{6k}} n^2$$

$$= 2^k \frac{9^k}{64^k} n^2$$

$$=$$

recursion tree example I/II

$$T(n) = 2T(3n/8) + n^2$$

$$n^2, (3n/8)^2, [(3/8)^2 n]^2, [(3/8)^3 n]^2, \ldots, (3/8)^{2k} n^2$$

(call at depth k)

There are 2^k nodes on level k, so work on level k is

$$\begin{aligned} 2^k \cdot (3/8)^{2k} n^2 &= 2^k \frac{3^{2k}}{2^{6k}} n^2 \\ &= 2^k \frac{9^k}{64^k} n^2 \\ &= \left(\frac{9}{32}\right)^k n^2 \end{aligned}$$

recursion tree example II/II

Total work

$$T(n) = n^2 \sum_{i=0}^{\log_{8/3} n} \left(\frac{9}{32}\right)^k$$

$$\leq$$

$$=$$

$$= \quad n^2$$

recursion tree example II/II

Total work

$$
\begin{aligned}
T(n) &= n^2 \sum_{i=0}^{\log_{8/3} n} \left(\frac{9}{32}\right)^k \\
&\leq n^2 \sum_{i=0}^{\infty} \left(\frac{9}{32}\right)^k \\
&= \\
&= \quad n^2
\end{aligned}
$$

recursion tree example II/II

Total work

$$
\begin{aligned}
T(n) &= n^2 \sum_{i=0}^{\log_{8/3} n} \left(\frac{9}{32}\right)^k \\
&\leq n^2 \sum_{i=0}^{\infty} \left(\frac{9}{32}\right)^k \\
&= n^2 \frac{1}{1 - 9/32} \\
&= \quad n^2
\end{aligned}
$$

recursion tree example II/II

Total work

$$
\begin{aligned}
T(n) &= n^2 \sum_{i=0}^{\log_{8/3} n} \left(\frac{9}{32}\right)^k \\
&\leq n^2 \sum_{i=0}^{\infty} \left(\frac{9}{32}\right)^k \\
&= n^2 \frac{1}{1 - 9/32} \\
&= \frac{32}{23} n^2
\end{aligned}
$$