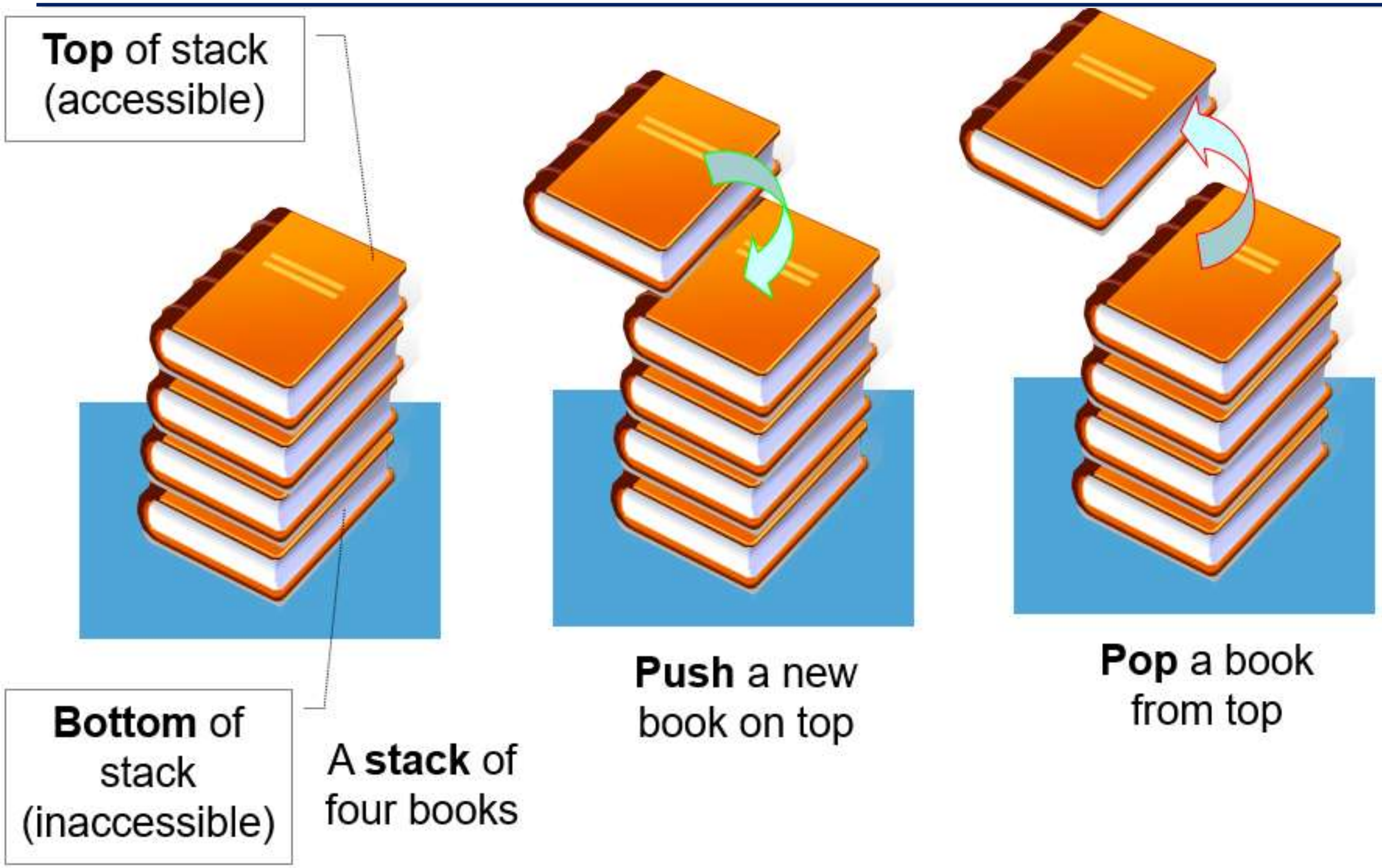
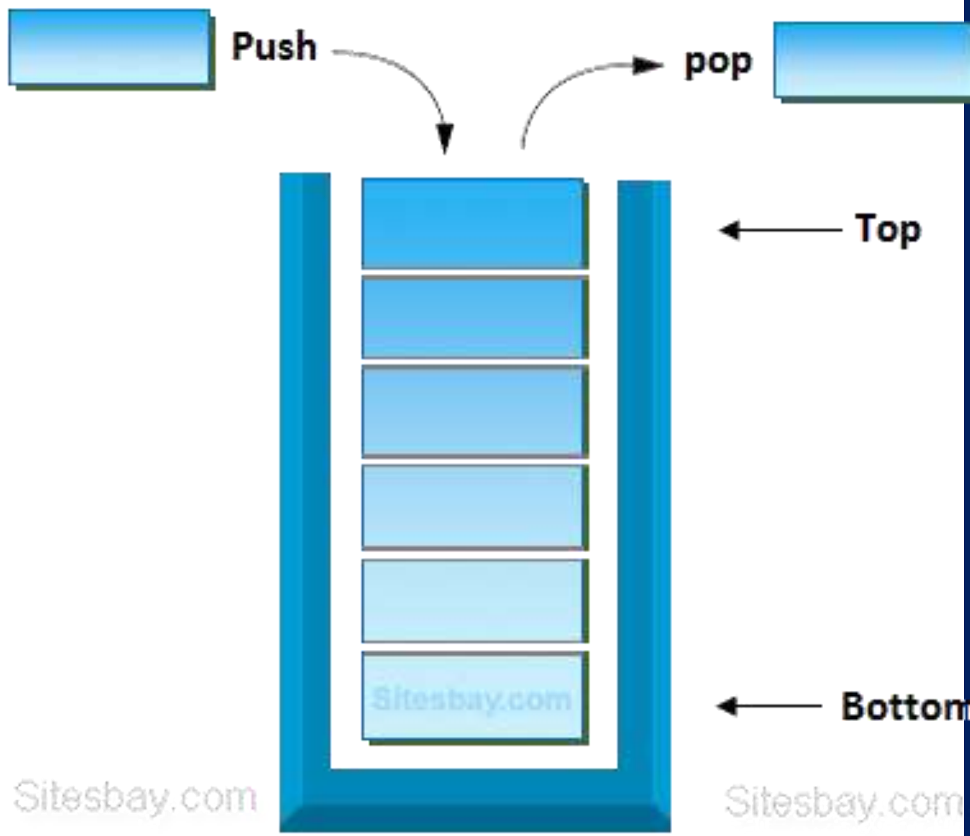
Greedy Algorithms

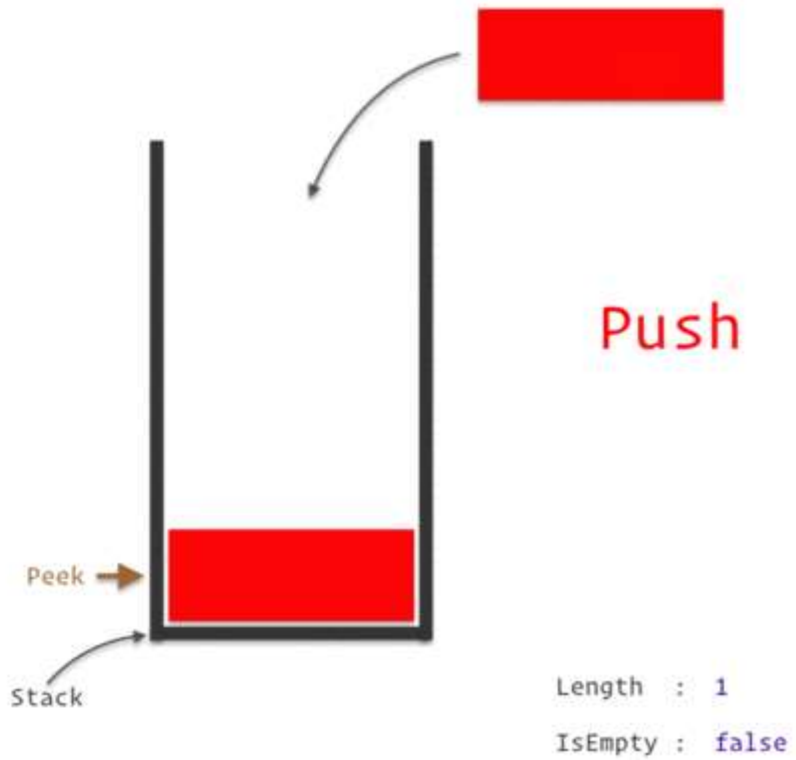
Stack





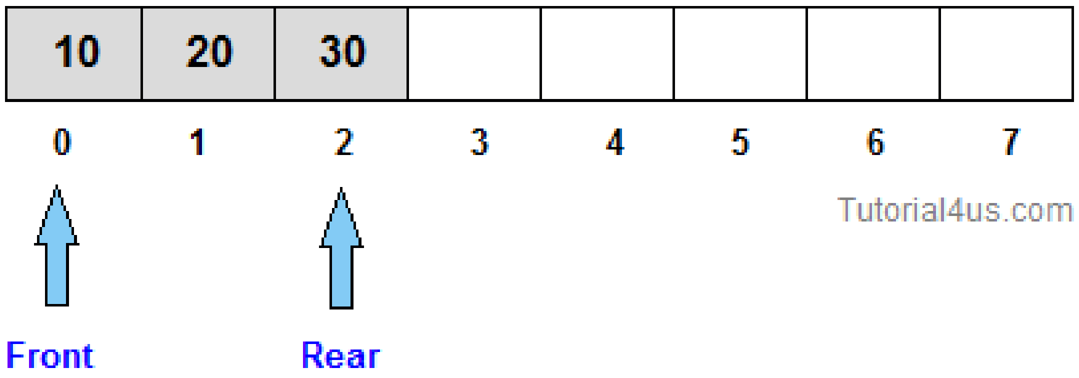
[VisuAlgo - Linked List (Single, Doubly), Stack, Queue, Deque]()

[Stack in C | Real Life Example of Stack (sitesbay.com)]()



[JavaScript — Stacks and Queues. Today we will learn about the basics of… | by Dhara Patel | Medium]

Queue



[Insert Element in Queue in C++ | Insert Element in Queue Program in C++ (sitesbay.com)]

Example

- **Largest Number**
- What is the largest number that consists of digits 3, 9, 5, 9, 7, 1? Use all the digits.

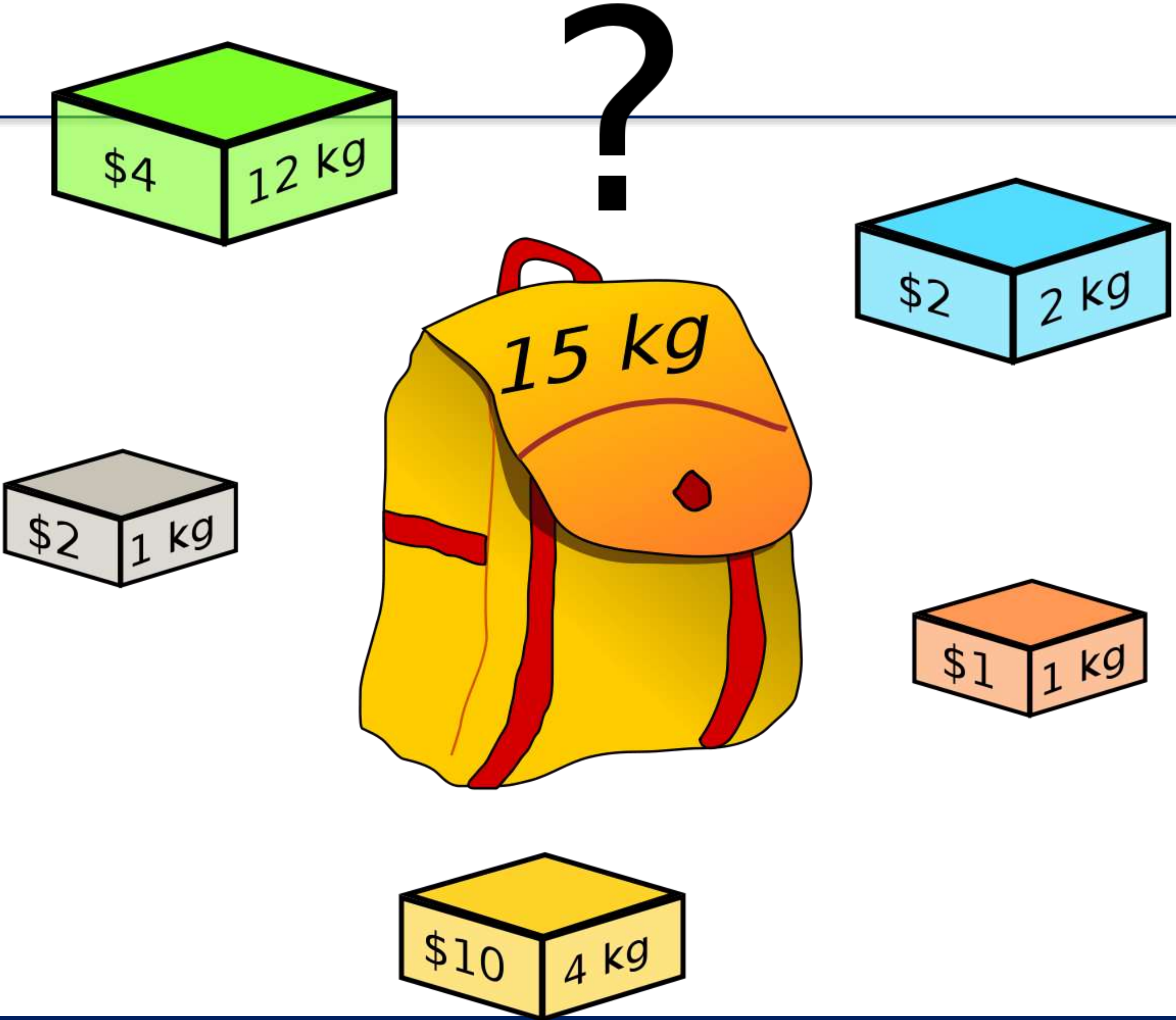
1. Find max digit
2. Append it to the number
3. Remove it from the list of digits
4. Repeat while there are digits in the list

- 997531

Money Change

Optimization

- Maximization
- Minimization



Knapsack problem

- Objective function: value in the knapsack
- Constraints: sum of the weights of the chosen items cannot exceed the knapsack capacity.
  - Factional
    - You can take any fraction of any item.
  - 0-1
    - You either take the whole item or leave it.

Brute-force knapsack problems

- Enumerate all possible combination/ subset (not permutation –order not matter)
- How many subset 2^n
- Running time at least $\Omega(2^n)$

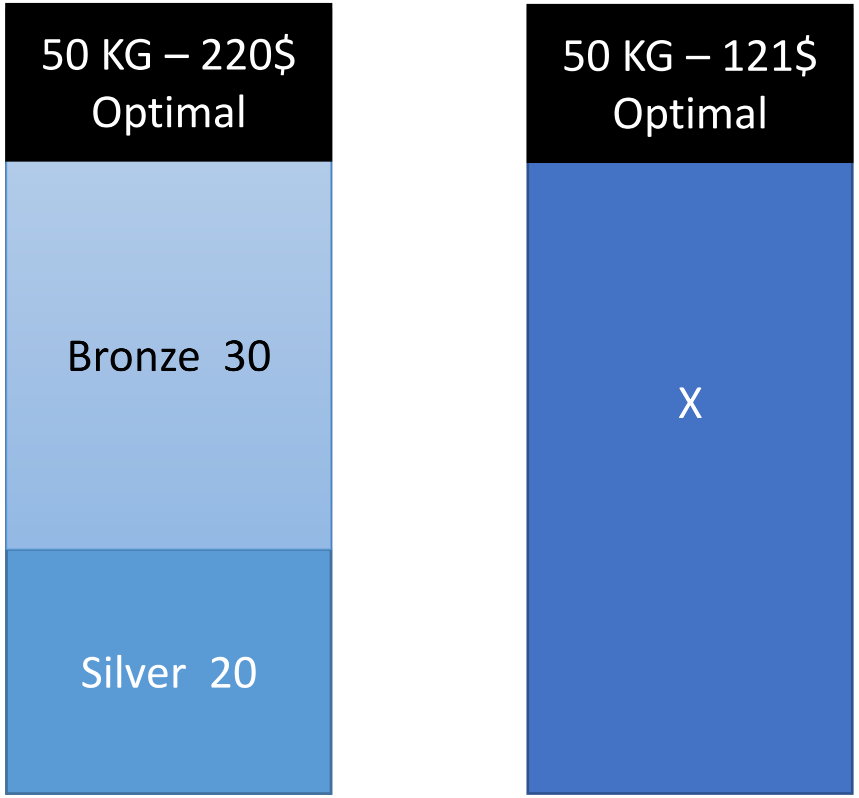
1) Identify feasible solution(a solution that satisfy the problem constraints) (compare with the capacity)
2) Calculate the sum of the values
3) Keep track of maximum value

Knapsack problem - Example

Item	Weight KG	Value$	Value$/KG
• Gold	10	60	6$
• Silver	20	100	5$
• Bronze	30	120	4$

Knapsack problem - Example

Item	Weight KG	Value$
Gold	10	60
Silver	20	100
Bronze	30	120
X	**50**	**121**



Another greedy Strategy. The item with the highest value

There is no known greedy algorithm solves 0-1 Knapsack problem

Greedy fails because no future insight and no back track

Greedy

- Definition": Iteratively make greedy decisions, hope everything works out at the end.
- Pros:
  - 1. Easy to propose multiple greedy algorithms for many problems.
  - 2. Easy running time analysis. (Contrast with Master method etc.)
  - 3.Hard to establish correctness.(Contrast with straightforward inductive correctness proofs.)
- DANGER: Most greedy algorithms are NOT correct. (Even if your intuition says otherwise!)

https://www.coursera.org/learn/algorithms-greedy

Divide and Conquer

MERGE SORT

Merge Sort

6 5 3 1 8 7 2 4

https://en.wikipedia.org/wiki/Merge_sort

Merge Sort

Analyzing merge sort

MERGE-SORT $A[1 \,..\, n]$	$T(n)$
1. If $n = 1$, done.	$\Theta(1)$
2. Recursively sort $A[\,1 \,..\, \lceil n/2 \rceil\,]$ and $A[\,\lceil n/2 \rceil + 1 \,..\, n\,]$.	$2T(n/2)$
3. "Merge" the two sorted lists	$\Theta(n)$

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

Merge sort dance

- https://www.youtube.com/watch?v=XaqR3G_NVoo

Recurrence solving

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.

$$T(n)$$

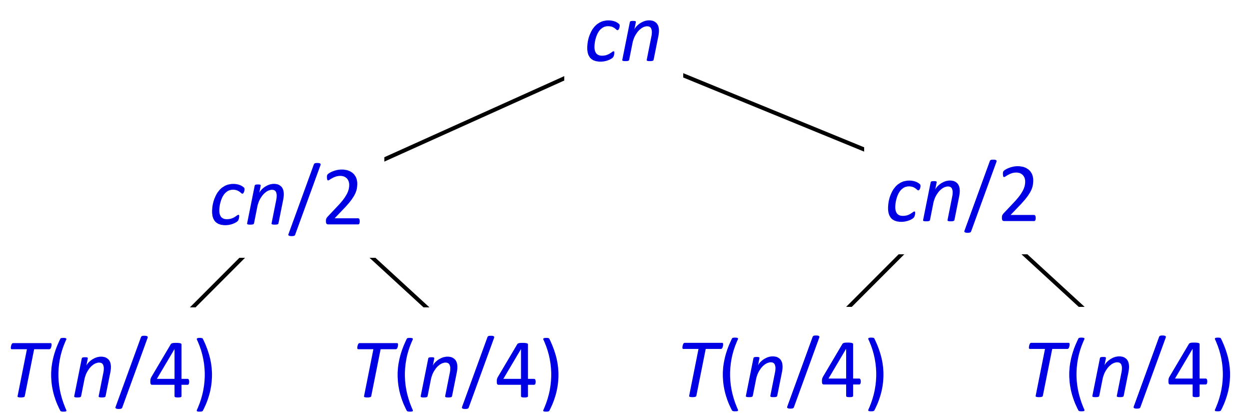
Recursion tree

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

$$
\begin{array}{ccc}
 & cn & \\
\swarrow & & \searrow \\
T(n/2) & & T(n/2)
\end{array}
$$

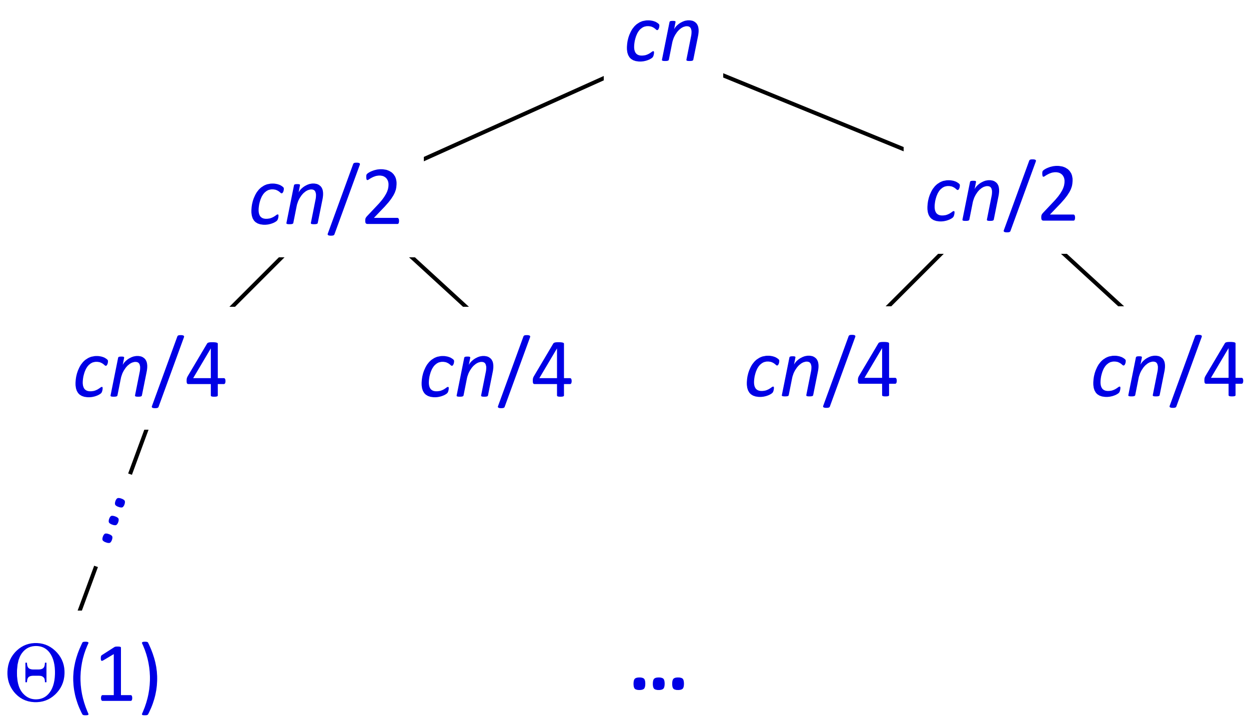
Recursion tree

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



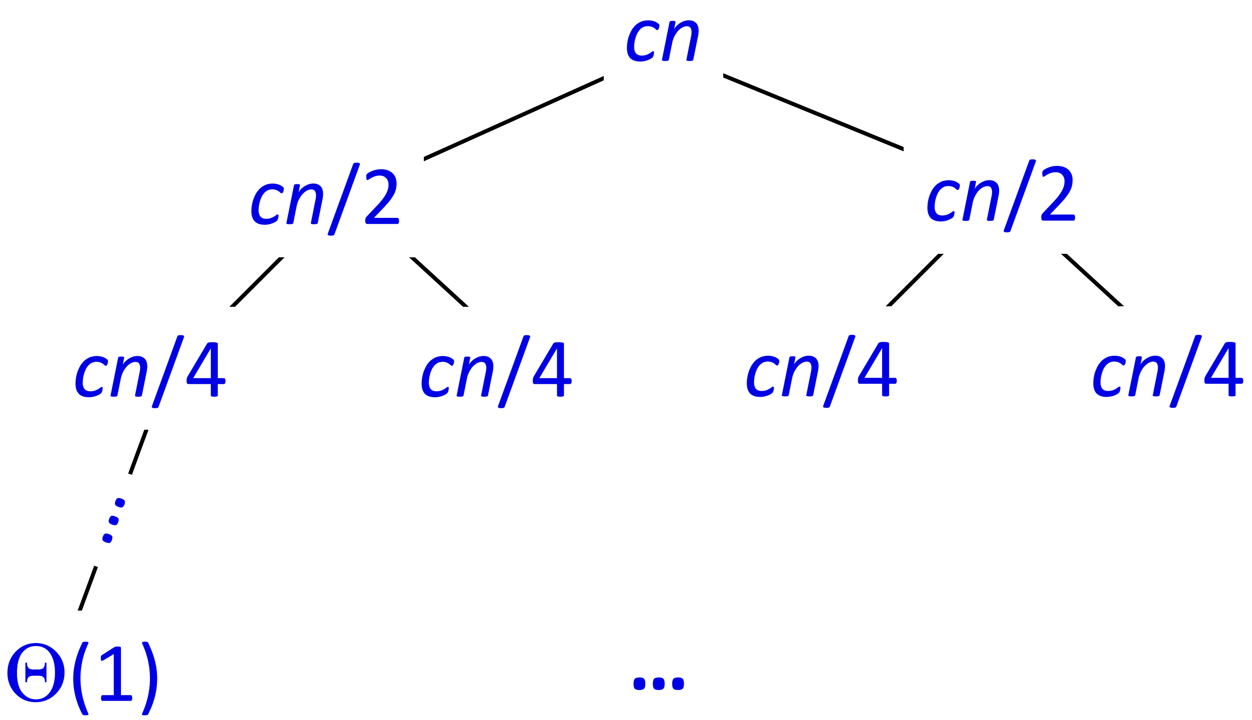
Recursion tree

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



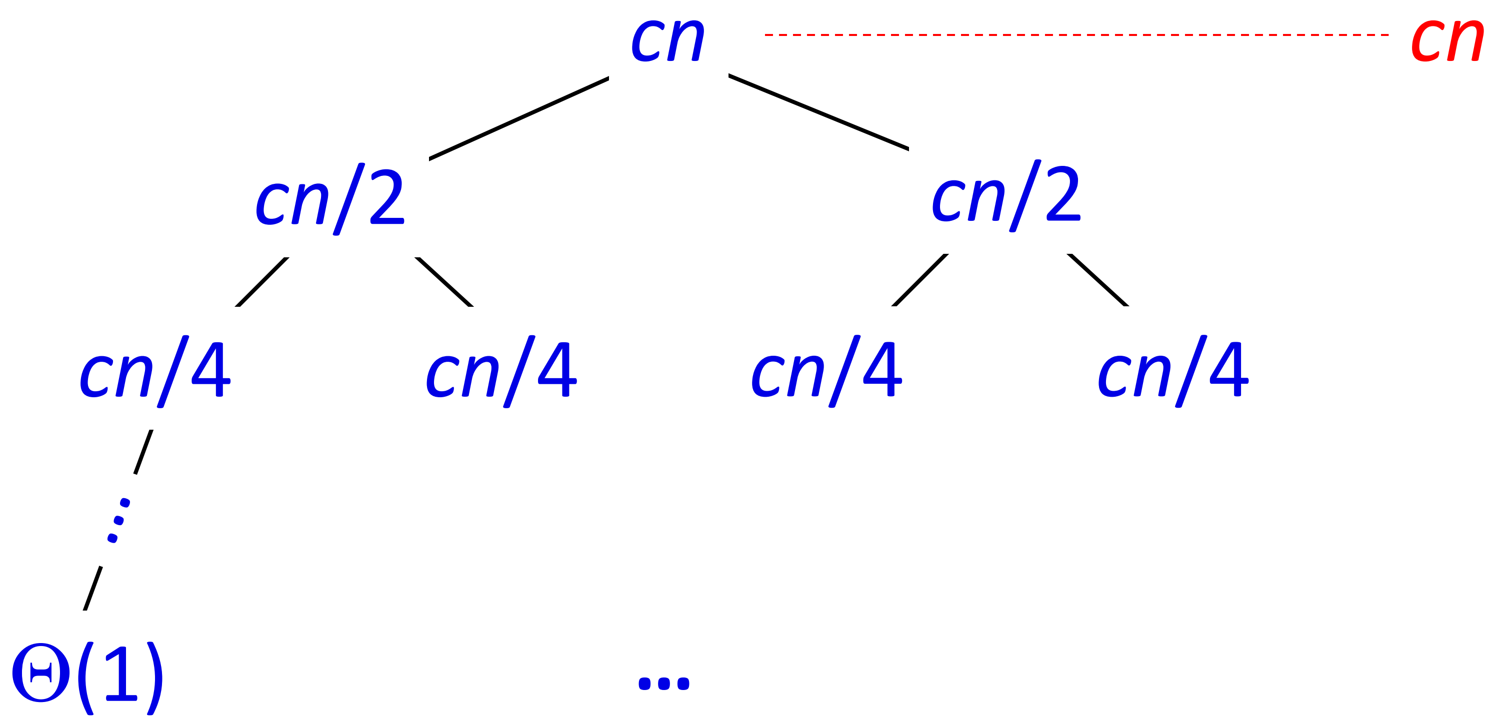
Recursion tree

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



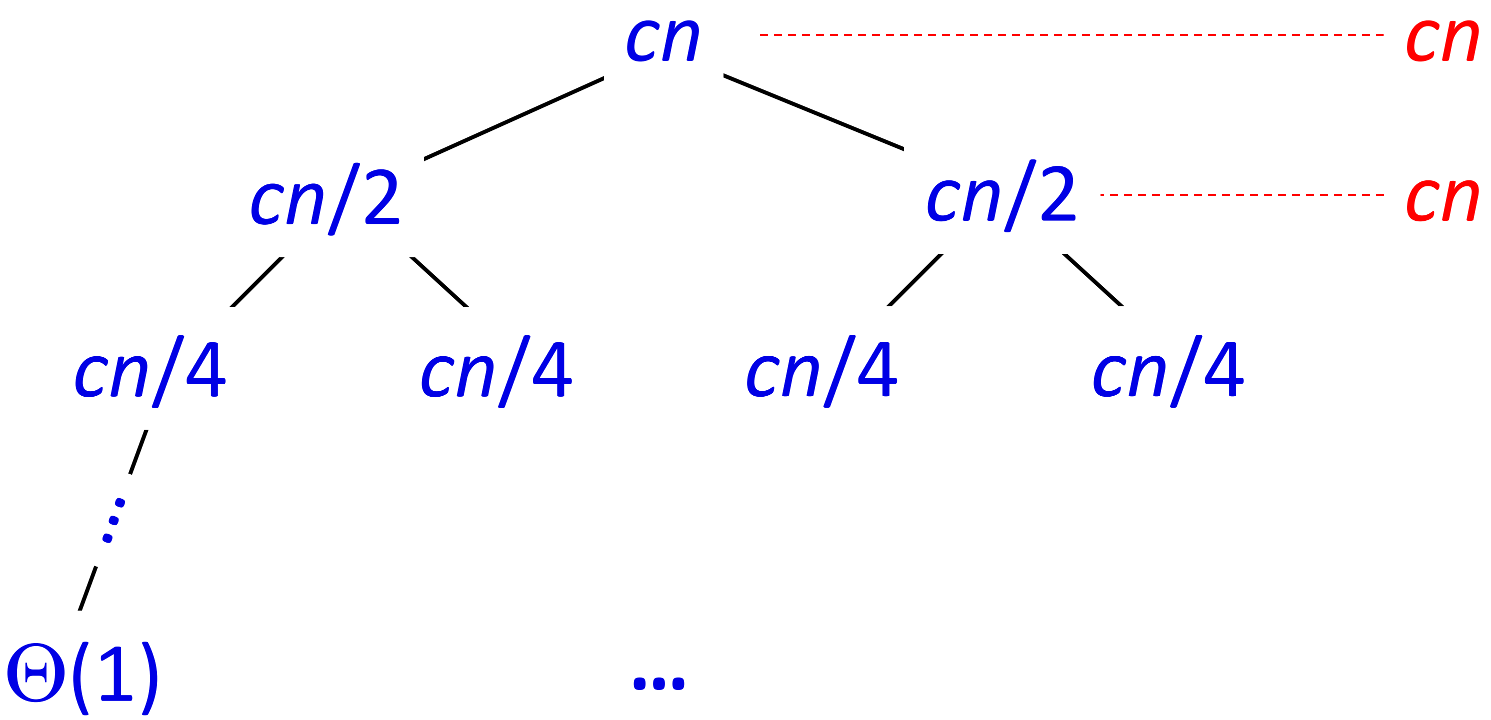
Recursion tree

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



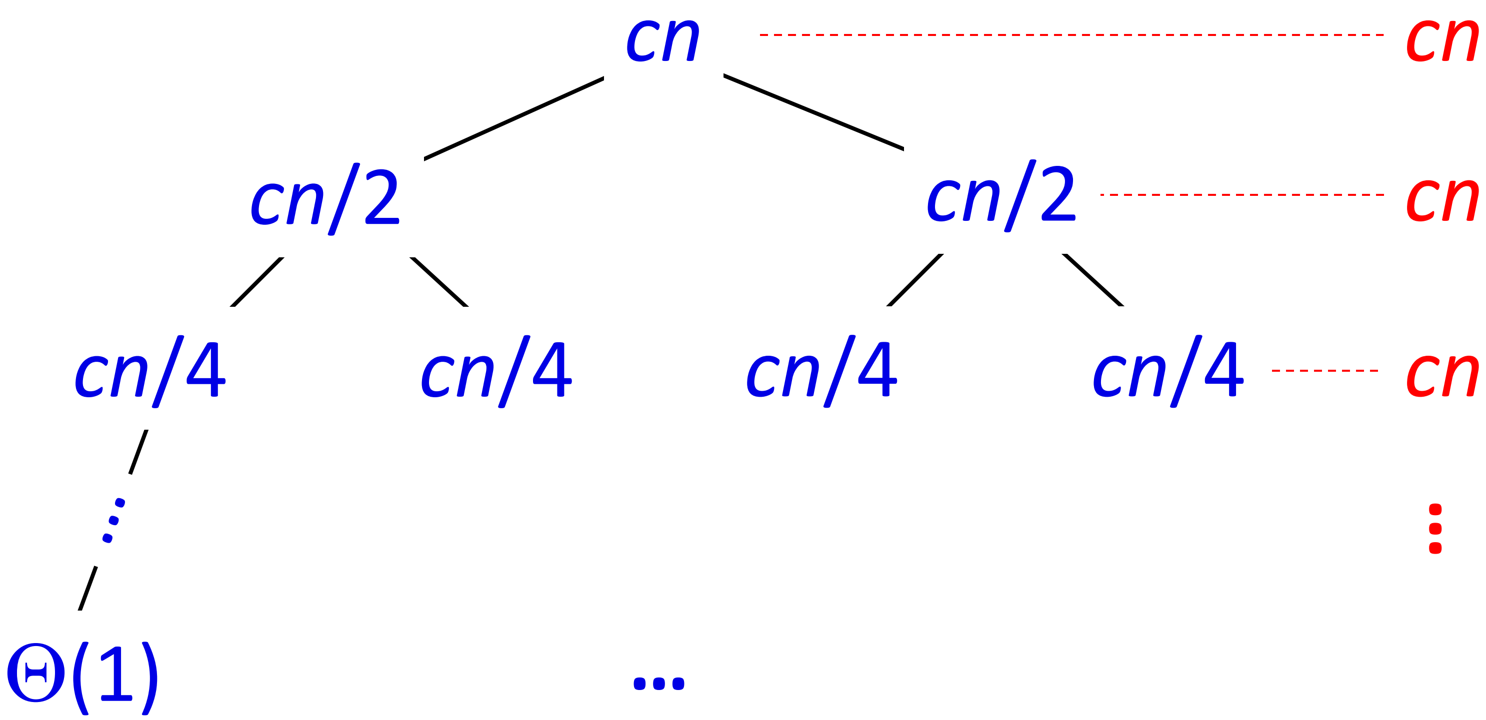
Recursion tree

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



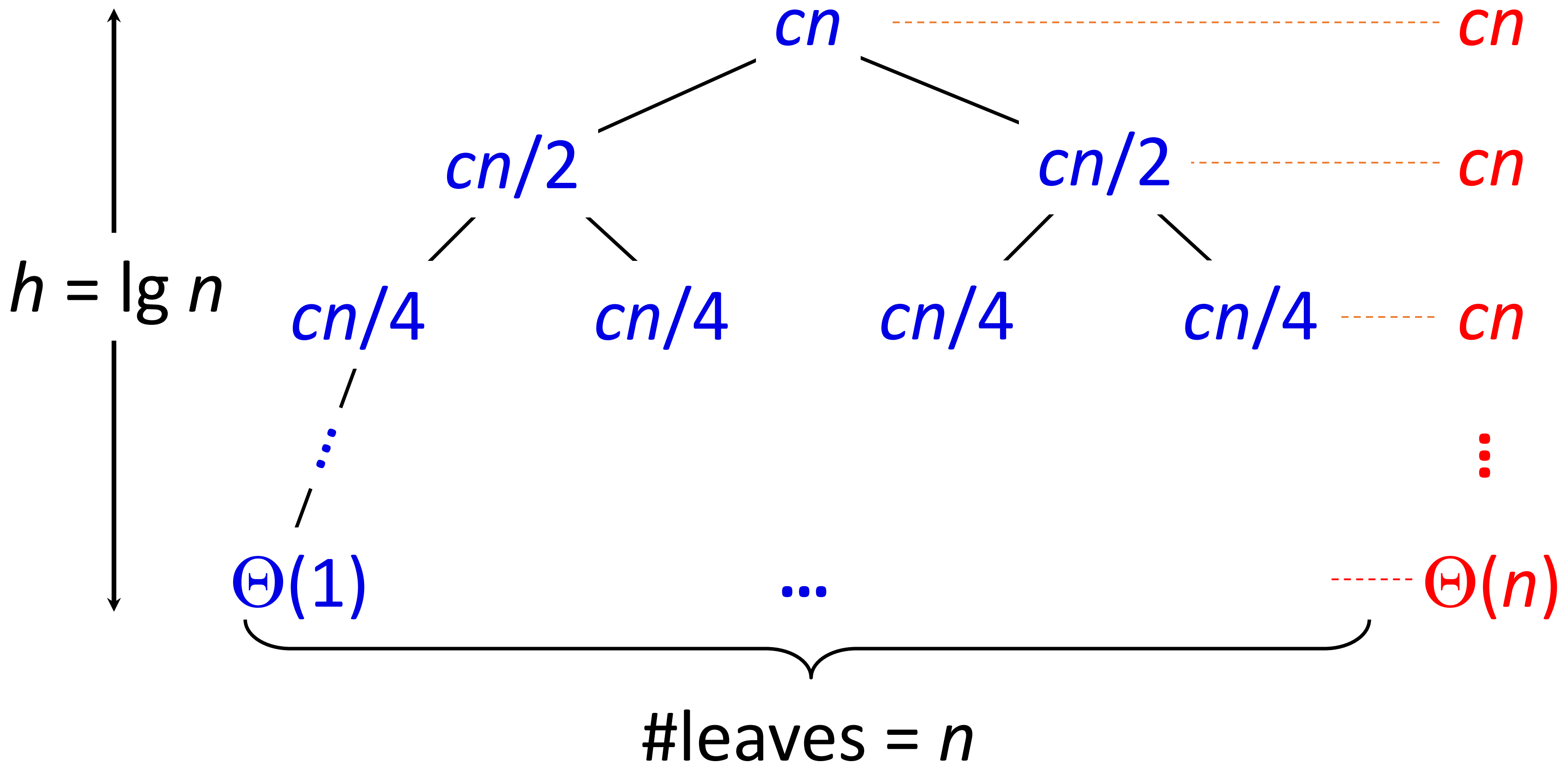
Recursion tree

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



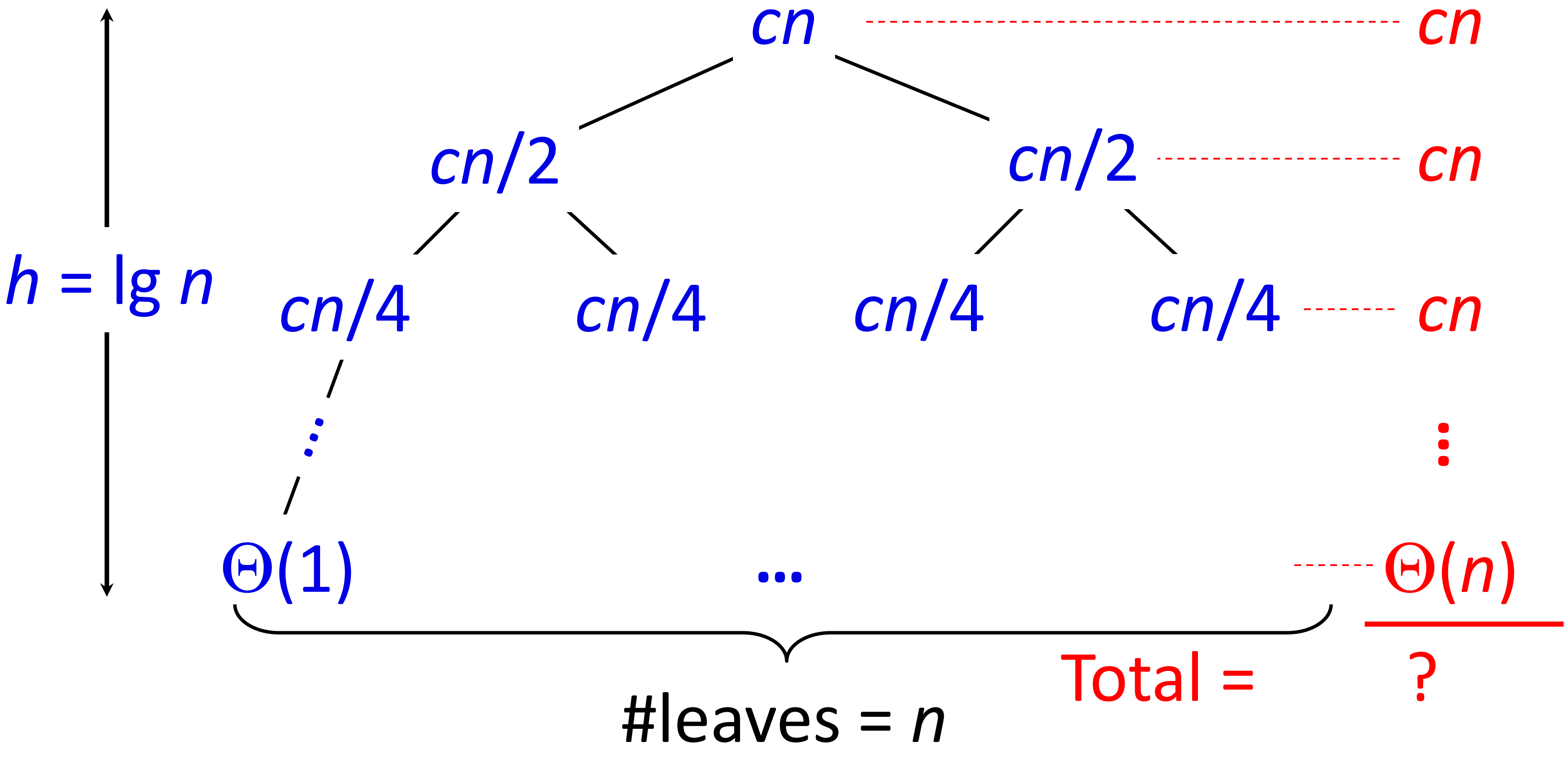
Recursion tree

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



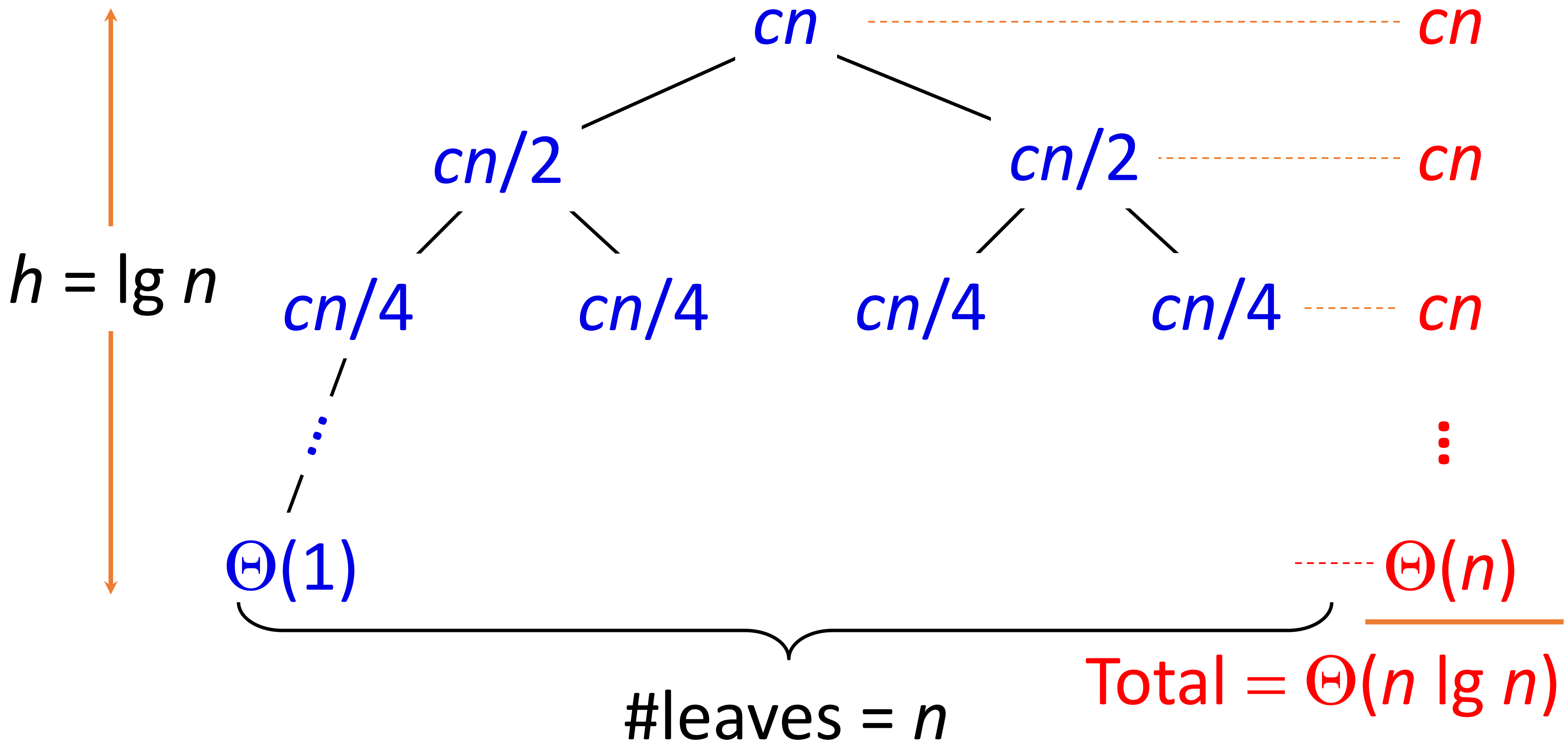
Recursion tree

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.



Divide-and-Conquer

1. Divide the problem into a number of subproblems that are smaller instances of the same problem.
2. Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
3. Combine the solutions to the subproblems into the solution for the original problem.

Divide-and-Conquer

- When the subproblems are large enough to solve recursively, we call that the recursive case. Once the subproblems become small enough that we no longer recurse,
- we say that the recursion “bottoms out” and that we have gotten down to the base case. Sometimes, in addition to subproblems that are smaller instances of the same problem, we have to solve subproblems that are not quite the same as the original problem.
- We consider solving such subproblems as part of the combine step.

Merge sort Divide-and-Conquer

- The merge sort algorithm follows the divide-and-conquer paradigm.

1. Divide: Divide the n-element sequence to be sorted into two subsequences of n/2 elements.
2. Conquer: Sort the two subsequences recursively using merge sort.
3. Combine: Merge the two sorted subsequences to produce the sorted answer.

- The recursion “bottoms out” when the sequence to be sorted has length 1.Every sequence of length 1 is already in sorted order

Recurrences

- The *master method* provides bounds for recurrences of the form

- $$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

- where a≥1, b>1 and f (n) is a given function. a subproblems, each of which is b the size of the original problem, and in which the divide and combine steps together take f (n) time.

Master Theorem

- If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ (for constants a > 0, b > 1, d ≥ 0), then:

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

Example 1

- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
- $T(n) = 4T\left(\frac{n}{2}\right) + O(n)$
- a=4
- b=2
- d=1
- Since d < $log_b a$
- T(n) = O($n^{log_b a}$) = O(n^2)

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

https://www.coursera.org/learn/algorithmic-toolbox

Example 2

- $$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$
- $$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$
- a=3
- b=2
- d=1
- Since d < $log_b a$
- T(n) = O($n^{log_b a}$) = O($n^{log_2 3}$)

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

https://www.coursera.org/learn/algorithmic-toolbox

Example 3

- $$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$
- $$T(n) = T\left(\frac{n}{2}\right) + O(1)$$
- a=1
- b=2
- d=0
- Since d =$log_b a$
- T(n) = O($n^d log\ n$) = O($n^0 log\ n$)= O($log\ n$)

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

https://www.coursera.org/learn/algorithmic-toolbox

Example 4

- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
- $T(n) = 2T\left(\frac{n}{2}\right) + O(n^2)$
- a=2
- b=2
- d=2
- Since d > $log_b a$
- T(n) = O(n^d) = O(n^2)

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

https://www.coursera.org/learn/algorithmic-toolbox

Integer Multiplication

- Input: Two n-digit nonnegative integers, x and y.
- Output: The product $x \times y$.

Integer Multiplication(The Grade-School Algorithm)

Analysis

- computing a partial product involves n multiplications(one per digit)
- at most n additions (at most one per digit)

- a total of at most 2n primitive operations.
- Since there are n partial products—one per digit of the second number—computing all of them requires at most $n \cdot 2n = 2n^2$ primitive operations.
- We still have to add them all up to compute the final answer.

total number of operations $=$constant(4)$\cdot n^2$.

```
5678
× 1234
--------
  22712
 17034
11356
5678
---------
7006652
```

A Recursive Algorithm

- In general, a number x with an even number n of digits can be expressed in terms of two n/2-digit numbers, its first half(a) and second half (b):

$$x = 10^{n/2}\, a + b.$$

  - for example (5678)
    - a=56   b=78
    - $x = \mathbf{10^{4/2}} \times \mathbf{56} + 78$
    - $x = \mathbf{10^{2}} \times \mathbf{56} + \mathbf{78}$
    - $x = \mathbf{10^{2}} \times \mathbf{56} + \mathbf{78}$
    - $x = 100 \times \mathbf{56} + \mathbf{78}$
    - $x = \mathbf{56}00 + \mathbf{78} = 5678$

- Similarly, we can write

$$\boldsymbol{y = 10^{n/2}\, c + d}$$

A Recursive Algorithm

- To compute the product of x and y:
- $x.y \;=\; (10^{\frac{n}{2}}\, a \;+\; b) \times \left(\mathbf{10^{\frac{n}{2}}\, \boldsymbol{c} + \boldsymbol{d}}\right)$**=**

$$10^{n} \cdot (a \cdot c) + 10^{n/2} \cdot (a \cdot d + b \cdot c) + b \cdot d.$$

RecIntMult Algorithm

Input: two n-digit positive integers x and y.

Output: the product $x \cdot y$.

Assumption: n is a power of 2.

if $n = 1$ then // base case

  compute $x \cdot y$ in one step and return the result

else // recursive case

  a, b := first and second halves of x

  c, d := first and second halves of y

  recursively compute $ac := a \cdot c$, $ad := a \cdot d$, $bc := b \cdot c$, and $bd := b \cdot d$

  compute $10^n \cdot ac + 10^{n/2} \cdot (ad + bc) + bd$

Integer Multiplication(Karatsuba Multiplication)

- Discovered in 1960 by Anatoly Karatsuba, who at the time was a 23-year-old student.
- The first and second halves of x are named a and b, a=56 and b=78, Similarly, c=12 and d=34

1) Compute $a \cdot c = 56 \cdot 12$, which is 672
2) Compute $b \cdot d = 78 \cdot 34 = 2652$.
3) Compute $(a + b) \cdot (c + d) = 134 \cdot 46 = 6164$.
4) Subtract the results of the first two steps from the result of the third step: $6164 - 672 - 2652 = 2840$.
5) Finally, we add up the results of steps 1, 2, and 4, but only after adding four trailing zeroes to the answer in step 1 and 2 trailing zeroes to the answer in step 4.

   Compute $10^4 \cdot 672 + 10^2 \cdot 2840 + 2652 = 6720000 + 284000 + 2652 = 70066552$.

Karatsuba Multiplication

- The RecIntMult algorithm uses four recursive calls, one for each of the products.
- But we don't really care about $a \cdot d$ or $b \cdot c$, except inasmuch as we care about their sum $a \cdot d + b \cdot c$

Karatsuba Multiplication

Input: two n-digit positive integers x and y.

Output: the product $x \cdot y$.

Assumption: n is a power of 2.

if $n = 1$ then // base case

compute $x \cdot y$ in one step and return the result

else // recursive case

a, b := first and second halves of x

c, d := first and second halves of y

compute $p := a + b$ and $q := c + d$

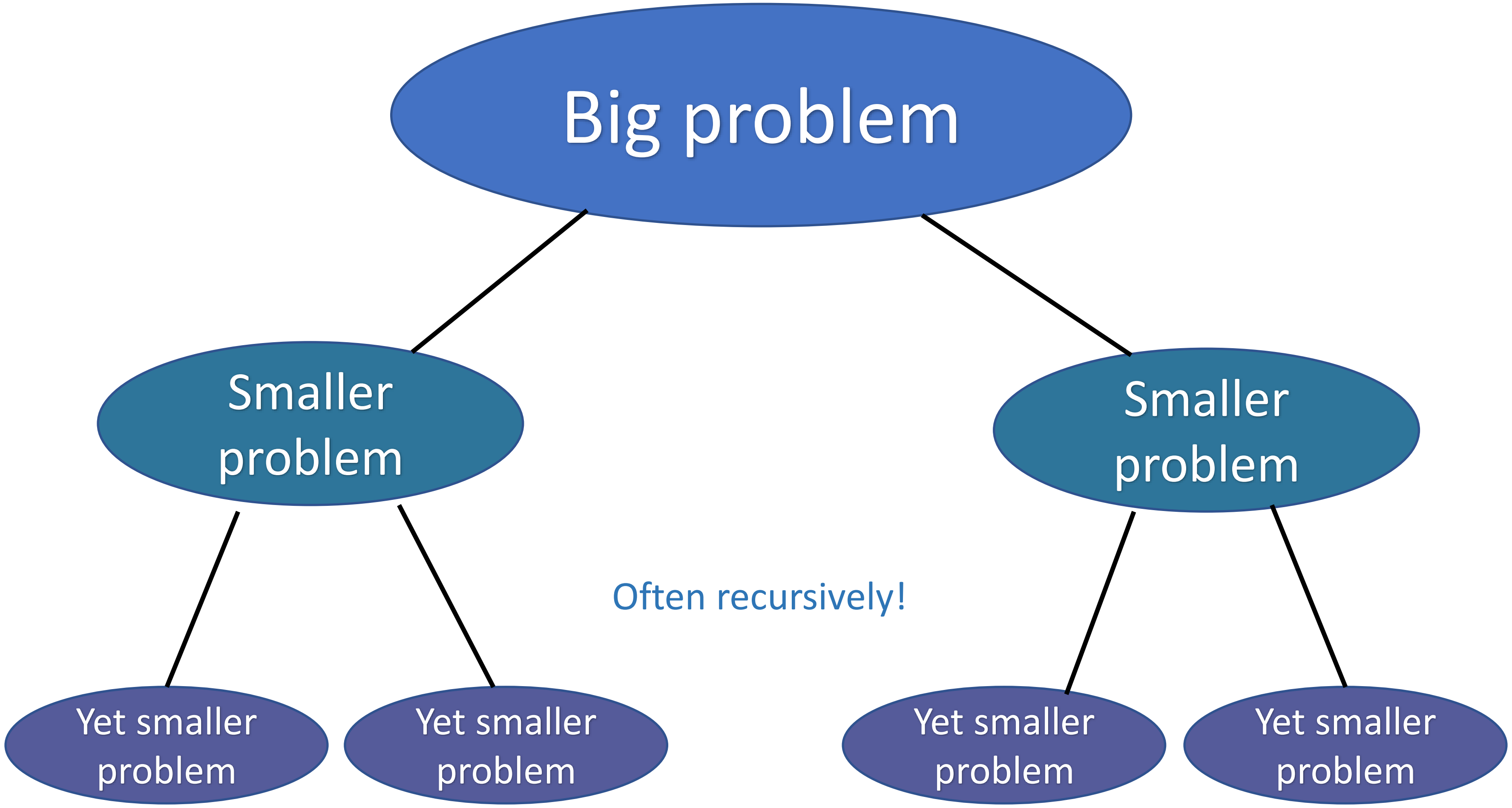
recursively compute $ac := a \cdot c$, $bd := b \cdot d$, and $pq := p \cdot q$

compute $adbc := pq - ac - bd$

compute $10^n \cdot ac + 10^{n/2} \cdot adbc + bd$

Divide and conquer

- Break problem up into smaller (easier) sub-problems



http://web.stanford.edu/class/archive/cs/cs161/cs161.1182/

Integer Multiplication

$$\begin{array}{r} 44 \\ \times\ 97 \\ \hline \end{array}$$

Integer Multiplication

$$\begin{array}{r} 1233925720752752384623764283568364918374523856298 \\ \times \quad 4562323582342395285623467235019130750135350013753 \\ \hline \end{array}$$

- At most n^2 multiplications
- At most n^2 additions (for carries)
- adding n different 2n-digit numbers…

Divide and conquer for multiplication

Break up an integer:

$$1234 = 12 \times 100 + 34$$

$$
\begin{aligned}
&1234 \times 5678 \\
&= (12 \times 100 + 34)(56 \times 100 + 78) \\
&= \underbrace{(12 \times 56)}_{1} 10000 + (\underbrace{34 \times 56}_{2} + \underbrace{12 \times 78}_{3}) 100 + \underbrace{(34 \times 78)}_{4}
\end{aligned}
$$

One 4-digit multiply → Four 2-digit multiplies

http://web.stanford.edu/class/archive/cs/cs161/cs161.1182/

More generally

Suppose n is even

Break up an n-digit integer:

$$[x_1 x_2 \cdots x_n] = [x_1 x_2 \cdots x_{n/2}] \times 10^{n/2} + [x_{n/2+1} x_{n/2+2} \cdots x_n]$$

$$x \times y = (a \times 10^{n/2} + b)(c \times 10^{n/2} + d)$$

$$= \underbrace{(a \times c)}_{1} 10^n + (\underbrace{a \times d}_{2} + \underbrace{c \times b}_{3}) 10^{n/2} + \underbrace{(b \times d)}_{4}$$

One n-digit multiply → Four (n/2)-digit multiplies

http://web.stanford.edu/class/archive/cs/cs161/cs161.1182/

Divide and conquer algorithm

x,y are n-digit numbers

Multiply(x, y):

Base case: I've memorized my 1-digit multiplication tables…

- **If** n=1:
  - **Return** xy

Say n is even…

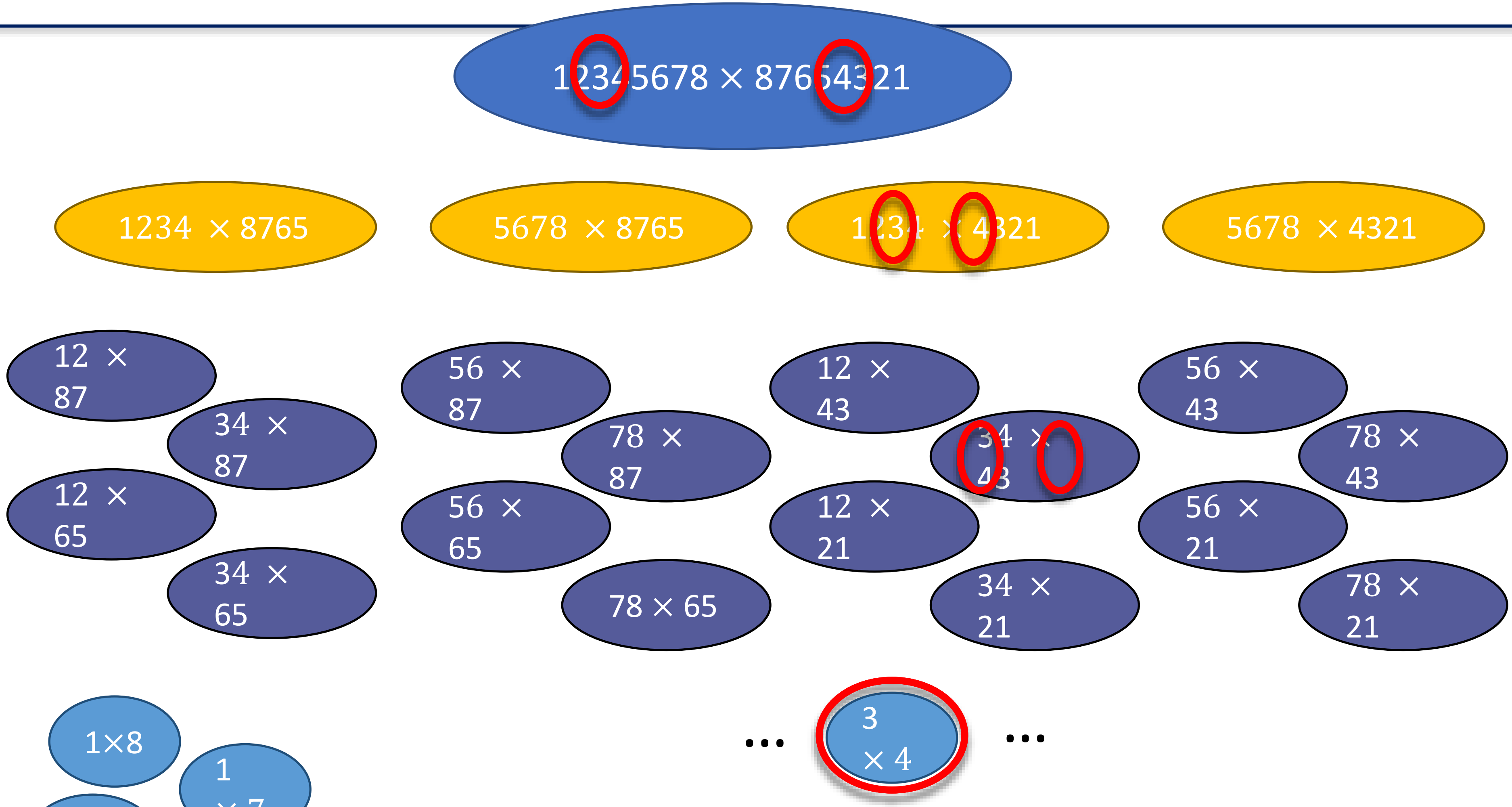
- Write $x = a\ 10^{\frac{n}{2}} + b$

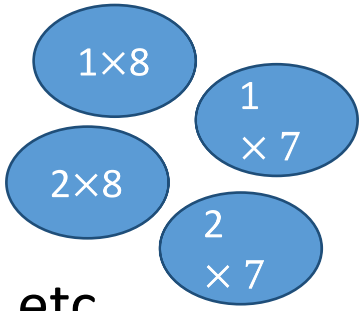
a, b, c, d are n/2-digit numbers

- Write $y = c\ 10^{\frac{n}{2}} + d$
- Recursively compute ac, ad, bc, bd:
  - ac = **Multiply**(a, c), etc…
- Add them up to get xy:
  - $xy = ac\ 10^n + (ad + bc)\ 10^{n/2} + bd$

http://web.stanford.edu/class/archive/cs/cs161/cs161.1182/

How many one-digit multiplies?





etc…

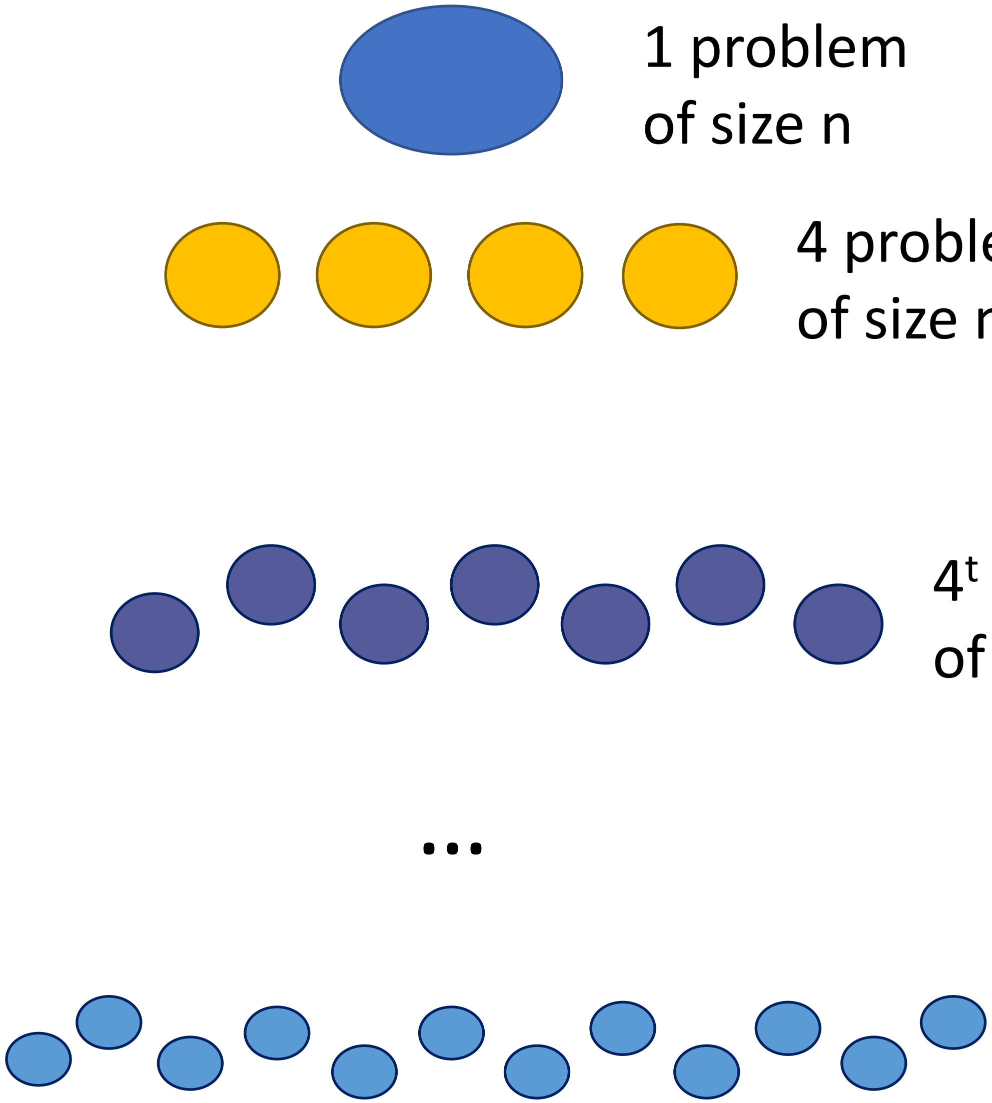
Claim: there are n^2 one-digit problems.

Every pair of digits still gets multiplied together separately.

So the running time is still at least n^2.

http://web.stanford.edu/class/archive/cs/cs161/cs161.1182/

Another way to see this



1 problem of size n

4 problems of size n/2

4^t problems of size $n/2^t$

…

$\underline{\ n^2\ }$ problems of size 1

- If you cut n in half $\log_2(n)$ times, you get down to 1.
- So we do this $\log_2(n)$ times and get…

$4^{\log\_2(n)} = n^2$

problems of size 1.

This is just a lower bound – we're just counting the number of size-1 problems!

http://web.stanford.edu/class/archive/cs/cs161/cs161.1182/

Divide and conquer can actually make progress

- Karatsuba figured out how to do this better!

$$xy = (a \cdot 10^{n/2} + b)(c \cdot 10^{n/2} + d)$$
$$= ac \cdot 10^{n} + (ad + bc)10^{n/2} + bd$$

Need these three things

- If only we recurse three times instead of four…

http://web.stanford.edu/class/archive/cs/cs161/cs161.1182/

Karatsuba integer multiplication

- Recursively compute these THREE things:
  - ac
  - bd
  - (a+b)(c+d)

Subtract these off

get this

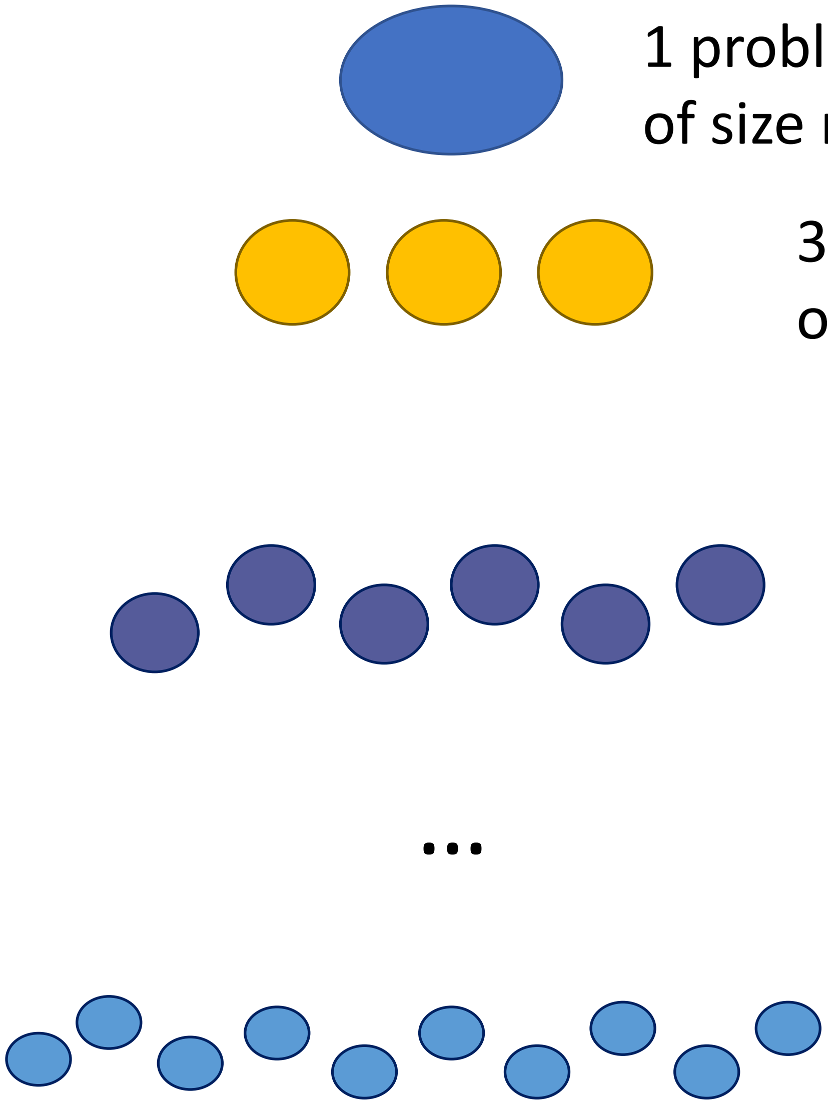
(a+b)(c+d) = ac + bd + bc + ad

- Assemble the product:

$$
\begin{aligned}
xy &= (a \cdot 10^{n/2} + b)(c \cdot 10^{n/2} + d) \\
&= ac \cdot 10^{n} + (ad + bc)10^{n/2} + bd
\end{aligned}
$$

✓ ✓ ✓

What's the running time?



1 problem
of size n

3 problems
of size n/2

3^t problems
of size $n/2^t$

…

$n^{1.6}$ problems
of size 1

- If you cut n in half $\log_2(n)$ times, you get down to 1.
- So we do this $\log_2(n)$ times and get…

$3^{\log\_2(n)} = n^{\log\_2(3)} \approx n^{1.6}$
problems of size 1.

We still aren't accounting for the work at the higher levels! But we'll see later that this turns out to be okay.

http://web.stanford.edu/class/archive/cs/cs161/cs161.1182/