Design and Analysis of Algorithm

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

REF BOOK: THOMAS CORMEN

Algorithm Design Techniques/Strategies

- Brute force
- Divide and conquer
- Space and time tradeoffs
- Greedy approach
- Dynamic programming
- Backtracking
- Branch and bound

Divide & Conquer

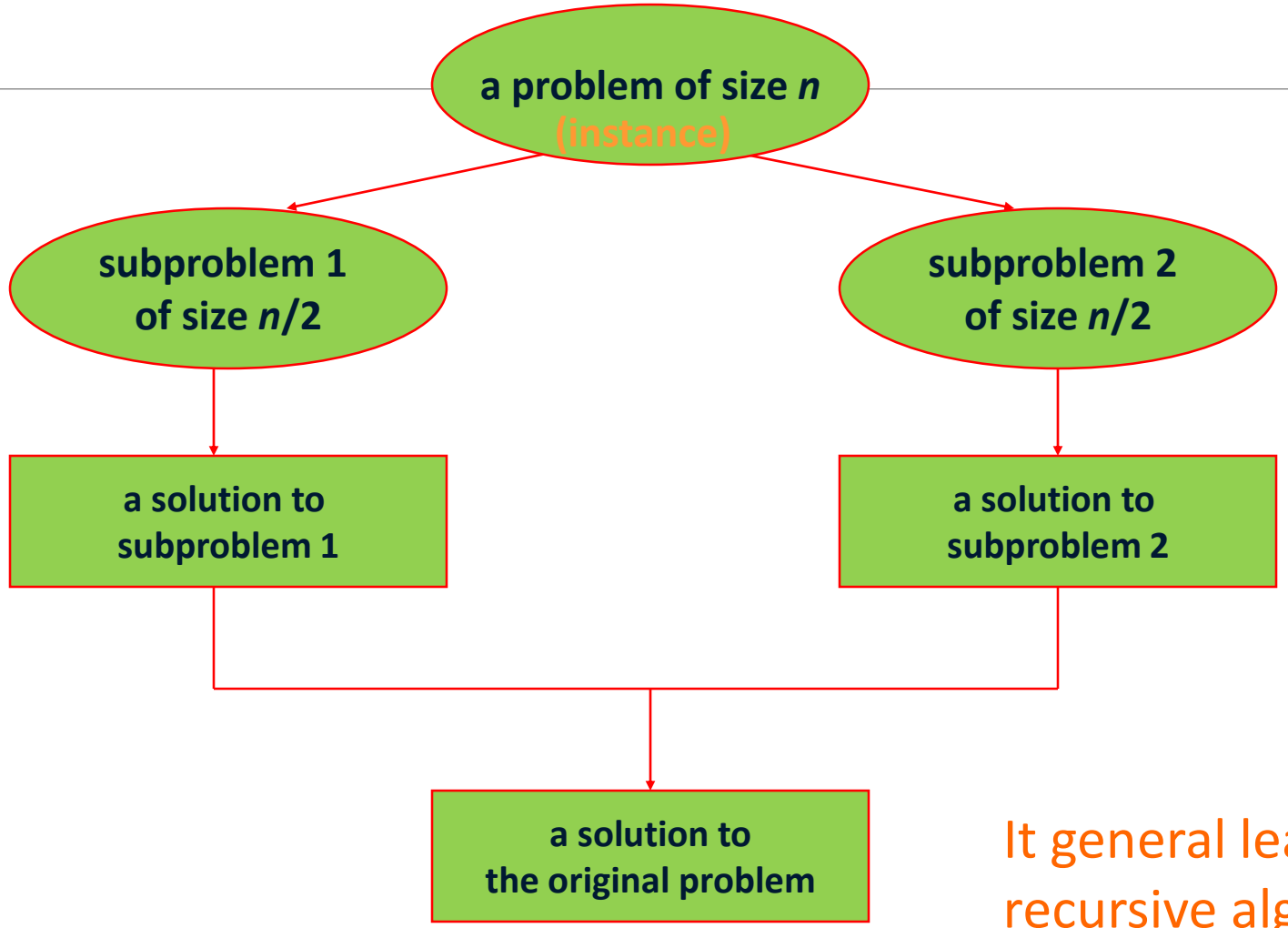
Control Abstraction

```
DANDC (P)
{
if SMALL (P) then return S (p);
else
{
divide p into smaller instances p1, p2, …. Pk, k >= 1;
apply DANDC to each of these sub problems;
return (COMBINE (DANDC (p1) , DANDC (p2),…., DANDC (pk));
}
}
```

Divide the problem into smaller sub problems
Conquer the sub problems by solving them recursively.
Combine the solutions to the sub problems into the solution of the original problem.

Divide-and-Conquer Technique (cont.)

REF BOOK: INTERNET



It general leads to a recursive algorithm!

Time complexity of the general algorithm

REF BOOK: THOMAS CORMEN

A Recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs

Special techniques are required to analyze the space and time required

$$T(n)=\begin{cases} aT(n/b)+D(n)+C(n) & , n \geq c \\ O(1) & , n < c \end{cases}$$

Time complexity (recurrence relation):

where D(n) : time for splitting

C(n) : time for conquer

c : a constant

REF BOOK: THOMAS CORMEN

Methods for Solving recurrences

- Substitution Method
  - We guess a bound and then use mathematical induction to prove our guess correct.
- Recursion Tree
  - Convert recurrence into tree
- Master Method

REF BOOK: INTERNET

Math You need to Review

properties of logarithms:

$\log_b(xy) = \log_b x + \log_b y$

$\log_b (x/y) = \log_b x - \log_b y$

$\log_b xa = a\log_b x$

$\log_b a = \log_x a/\log_x b$

- **properties of exponentials**:

$a^{(b+c)} = a^b a^c$

$a^{bc} = (a^b)^c$

$a^b / a^c = a^{(b-c)}$

$b = a^{\log_a b}$

$b^c = a^{c*\log_a b}$

REF BOOK: THOMAS CORMEN

Substitution Method

- Sum of n Natural Numbers

```
int sum(int n) {
    int s = 0;
    for(; n > 0; --n)
        s = s + n;
    return s;
}
```

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

Solve Recurrence in Long hand:

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

- Time $T(n)$ (additions)
- Space $S(n) = 2$

- Factorial (Recursive)

```
int fact(int n) {
    if (0 != n) return n*fact(n-1);
    return 1;
}
```

- Time $T(n)$ (multiplication)

$$\begin{aligned} T(n) &= T(n-1)+1, & n>0 \\ &= 0, & n=0 \end{aligned}$$

$$T(n) = n$$

Quick Sort

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

$$= 2(2(n/2^2) + (n/2)) + n$$

$$= 2^2\, T(n/2^2) + n + n$$

$$= 2^2\, (T(n/2^3) + (n/2^2)) + n + n$$

$$= 2^3\, T(n/2^3) + \underline{n + n + n}$$

$$= \boldsymbol{n \log n}$$

Binary Search

EXAMPLE 2: BINARY SEARCH

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

$T(1)=1$

Above is another example of recurrence relation and the way to solve it is by Substitution.

$T(n)=T(n/2) +1$

$= T(n/2^2)+1+1$

$= T(n/2^3)+1+1+1$

$= \log n$

T(n)= O(logn)

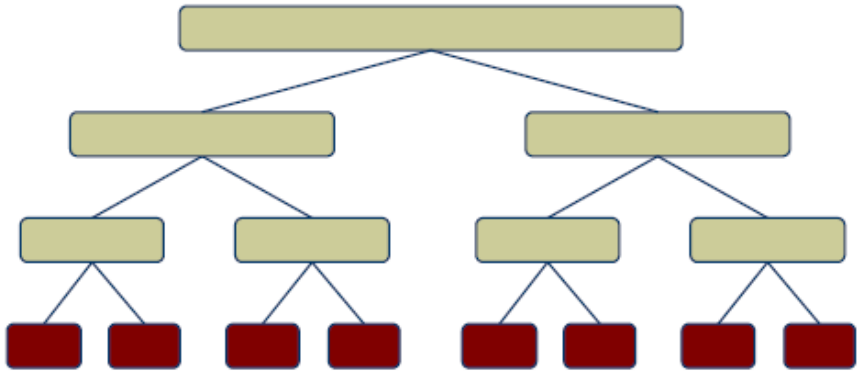
REF BOOK: THOMAS CORMEN

The Recursion Tree

Draw the recursion tree for the recurrence relation and look for a pattern:

$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(n/2) + bn & \text{if } n \geq 2 \end{cases}$$

depth	T's	size	time
0	1	n	bn
1	2	$n/2$	bn
i	2^i	$n/2^i$	bn
...	...	...	...



Total time = $bn + bn \log n$

(last level plus all previous levels)

REF BOOK: THOMAS CORMEN

Master Theorem

Master method provides a "cookbook" method for solving recurrences of the following form

$$T(n) = a\,T(n/b) + f(n)$$

where $a \geq 1$, $b > 1$. If f(n) is asymptotically positive function. T(n) has following asymptotic bounds:

There are 3 cases:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ with[1] $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ with $\epsilon > 0$, and $f(n)$ satisfies the regularity condition, then $T(n) = \Theta(f(n))$.
   Regularity condition: $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

Example of Master Method

REF BOOK: THOMAS CORMEN

To use the master theorem, we simply plug the numbers into the formula

Example 1: $T(n) = 9T(n/3)+n$. Here $a = 9$, $b = 3$, $f(n) = n$, and $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$. Since $f(n) = O(n^{\log_3 9-\epsilon})$ for $\epsilon = 1$, case 1 of the master theorem applies, and the solution is $T(n) = \Theta(n^2)$.

Example 2: $T(n) = T(2n/3)+1$. Here $a = 1$, $b = 3/2$, $f(n) = 1$, and $n^{\log_b a} = n^0 = 1$. Since $f(n) = \Theta(n^{\log_b a})$, case 2 of the master theorem applies, so the solution is $T(n) = \Theta(\log n)$.

Example 3: $T(n) = 3T(n/4) + n\log n$. Here $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$. For $\epsilon = 0.2$, we have $f(n) = \Omega(n^{\log_4 3+\epsilon})$. So case 3 applies if we can show that $af(n/b) \leq cf(n)$ for some $c < 1$ and all sufficiently large n. This would mean $3\frac{n}{4}\log\frac{n}{4} \leq cn\log n$. Setting $c = 3/4$ would cause this condition to be satisfied.

Example 4: $T(n) = 2T(n/2)+n\log n$. Here the master method does not apply. $n^{\log_b a} = n$, and $f(n) = n\log n$. Case 3 does not apply because even though $n\log n$ is asymptotically larger than n, it is not polynomially larger. That is, the ratio $f(n)/n^{\log_b a} = \log n$ is asymptotically less than n^ϵ for all positive constants ϵ.

REF BOOK: SRIDHAR (OXFORD PUBLICATION)

Master Method (Simplified)

Let T(n) be a monotonically increasing function that satisfies

$$T(n) = a\,T(n/b) + f(n)$$

$$T(1) = c$$

where $a \geq 1$, $b \geq 2$, $c>0$. If $f(n)$ is $\Theta(n^d)$ where $d \geq 0$ then solution to recurrence relation is given as

Case 1: $T(n) \in \Theta(n^d)$ -------- if $a < b^d$

Case 2: $T(n) \in \Theta(n^d \log n)$ -------- if $a = b^d$

Case 3: $T(n) \in \Theta(n^{\log_b a})$ -------- if $a > b^d$

REF BOOK: THOMAS CORMEN/SAHNI

Divide-and-Conquer Examples

Sorting : Mergesort and Quicksort

Binary tree traversals

Binary search

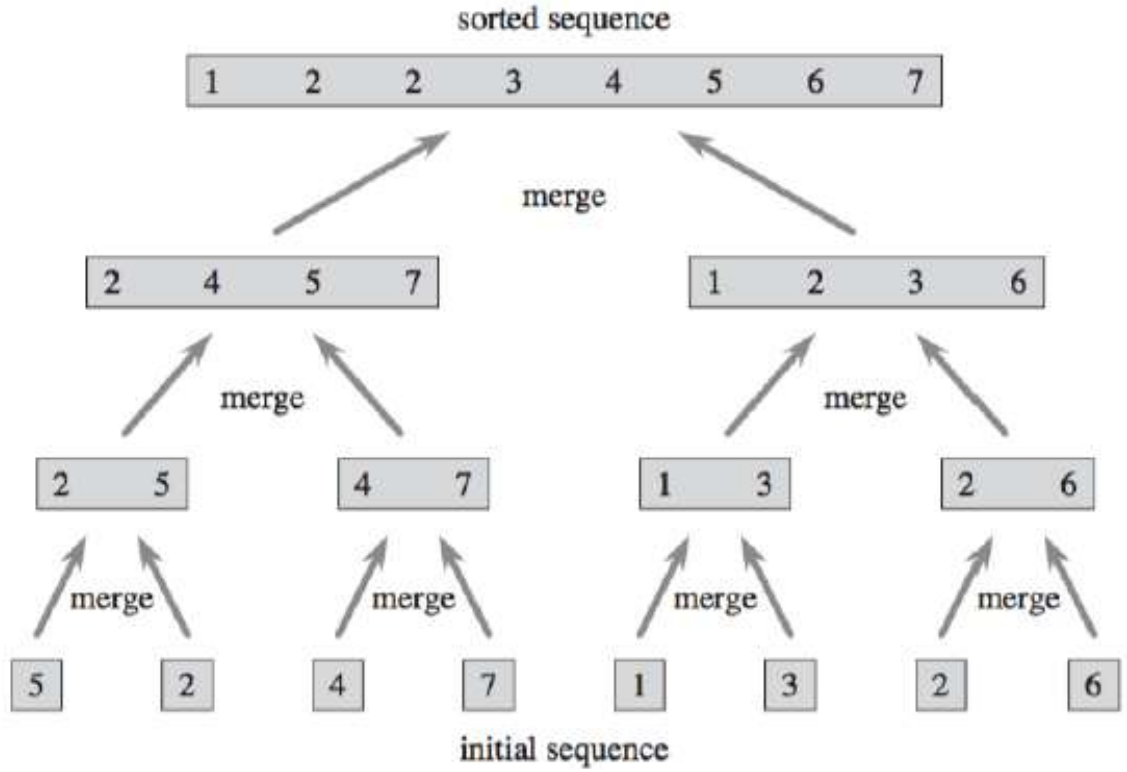
Multiplication of large integers

Matrix multiplication: Strassen’s algorithm

Closest-pair and convex-hull algorithms

Mergesort

Merge sort is a divide and conquer algorithm for sorting arrays. To sort an array, first you split it into two arrays of roughly equal size. Then sort each of those arrays using merge sort, and merge the two sorted arrays



Pseudocode of Merge-Sort (A, p, r)

INPUT: a sequence of *n* numbers stored in array A

OUTPUT: an ordered sequence of *n* numbers

```
MergeSort (A, p, r)   // sort A[p..r] by divide & conquer
1   if p < r
2       then q ← ⌊(p+r)/2⌋
3            MergeSort (A, p, q)
4            MergeSort (A, q+1, r)
5            Merge (A, p, q, r) // merges A[p..q] with A[q+1..r]
```

Initial Call: MergeSort(*A*, 1, *n*)

Procedure Merge

REF BOOK: THOMAS CORMEN

```
Merge(A, p, q, r)
1  n1 ← q – p + 1
2  n2 ← r – q
3      for i ← 1 to n1
4          do L[i] ← A[p + i – 1]
5      for j ← 1 to n2
6          do R[j] ← A[q + j]
7      L[n1+1] ← ∞
8      R[n2+1] ← ∞
9      i ← 1
10     j ← 1
11     for k ←p to r
12         do if L[i] ≤ R[j]
13             then A[k] ← L[i]
14                  i ← i + 1
15             else A[k] ← R[j]
16                  j ← j + 1
```

Input: Array containing sorted subarrays $A[p..q]$ and $A[q+1..r]$.

Output: Merged sorted subarray in $A[p..r]$.

Sentinels, to avoid having to check if either subarray is fully copied at each step.

Analysis of Mergesort

Running time $T(n)$ of Merge Sort:

Divide: computing the middle takes $\Theta(1)$

Conquer: solving 2 sub problems takes $2T(n/2)$

Combine: merging n elements takes $\Theta(n)$

Total:

$$T(n) = \Theta(1) \quad \text{if } n = 1$$
$$T(n) = 2T(n/2) + \Theta(n) \quad \text{if } n > 1$$

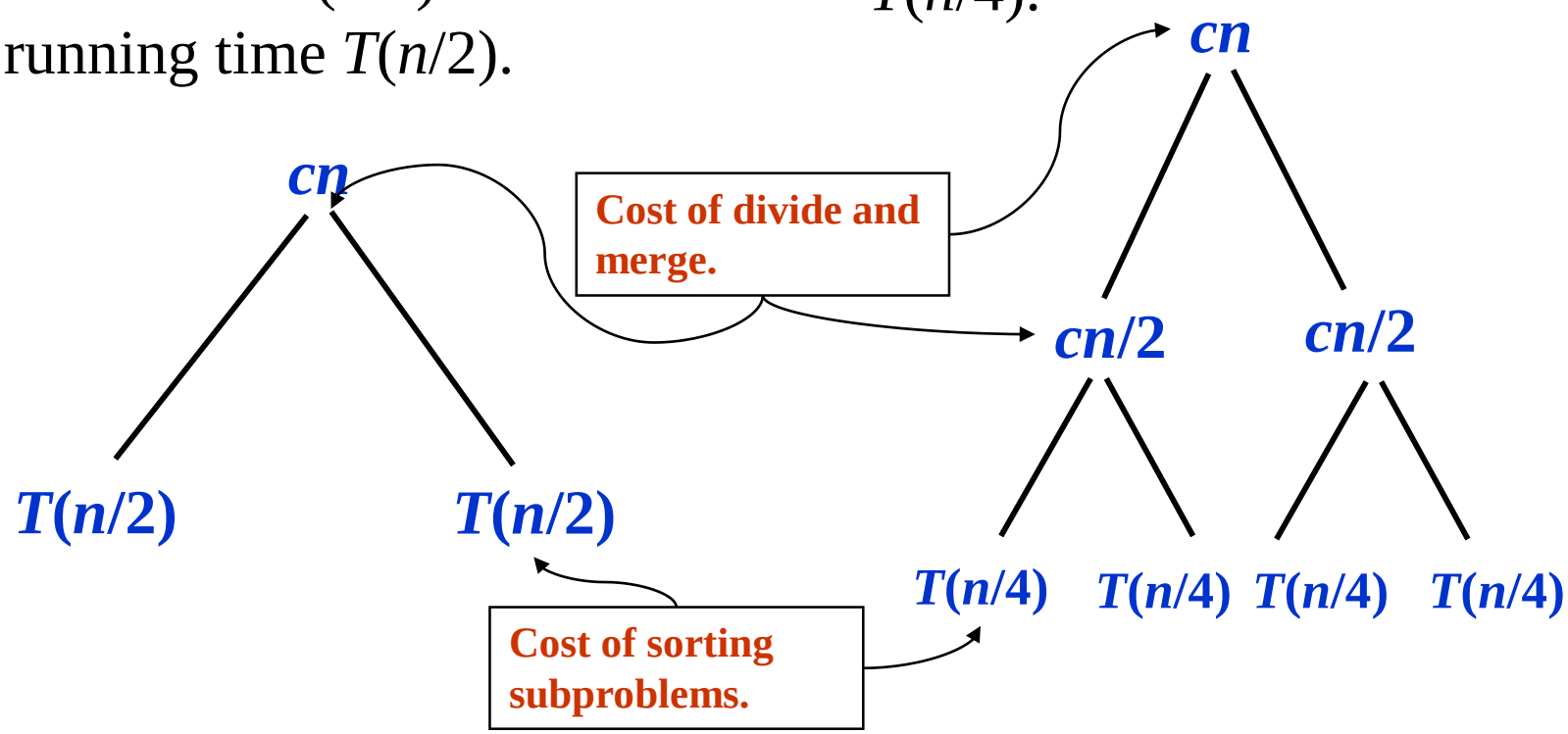
$\Rightarrow T(n) = \Theta(n \lg n)$

$\Rightarrow$Space requirement: $\Theta(n)$ (not in-place)

Recursion Tree for Merge Sort

For the original problem, we have a cost of *cn*, plus two subproblems each of size ($n/2$) and running time $T(n/2)$.

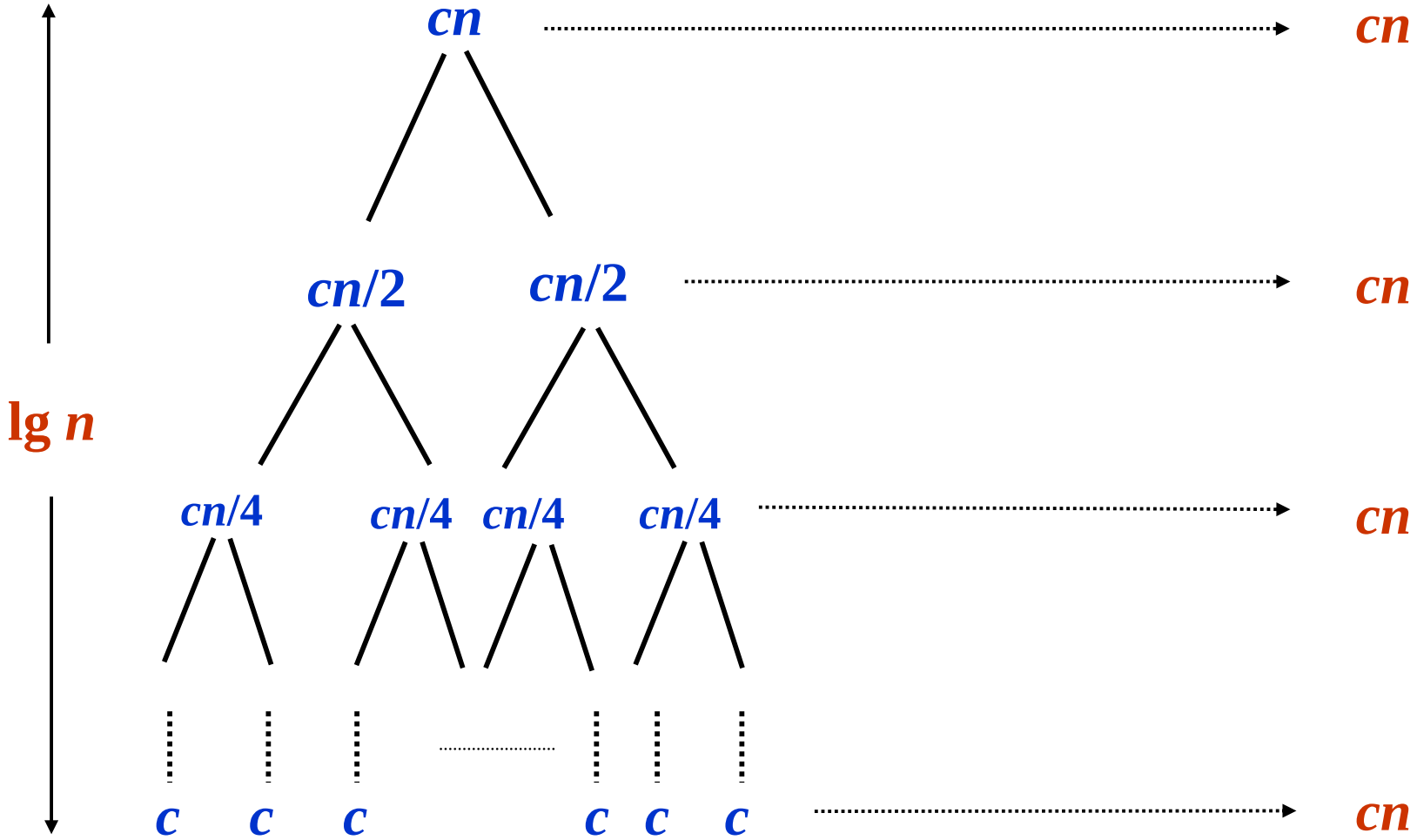
Each of the size $n/2$ problems has a cost of $cn/2$ plus two subproblems, each costing $T(n/4)$.



Recursion Tree for Merge Sort

Ref Book: Thomas Cormen

Continue expanding until the problem size reduces to 1.

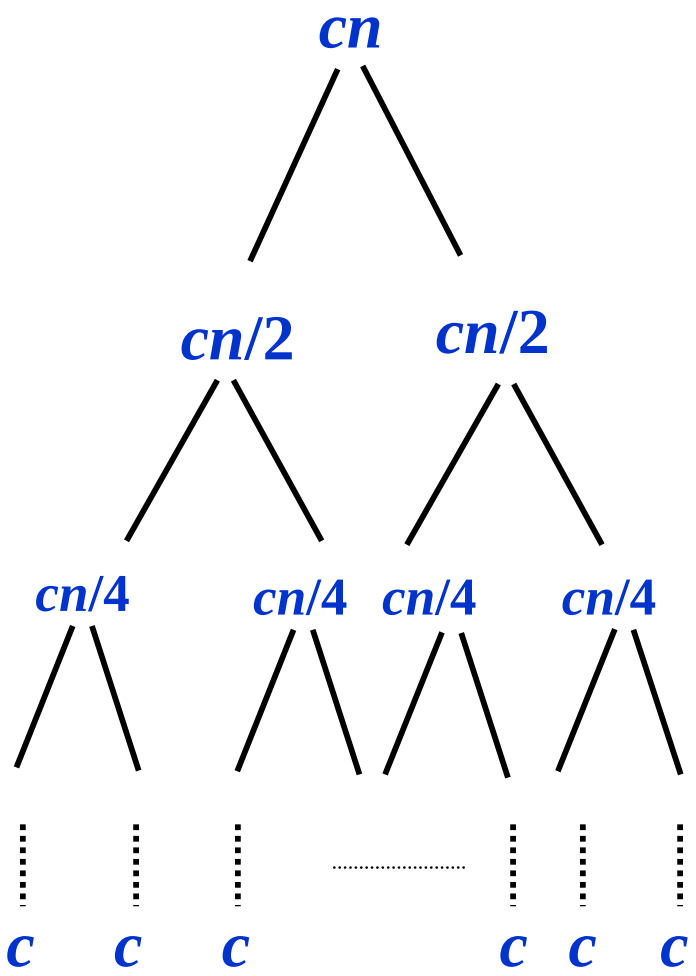


Total : $cn\lg n + cn$

Recursion Tree for Merge Sort

Ref Book: Thomas Cormen

Continue expanding until the problem size reduces to 1.



- Each level has total cost ***cn***.
- Each time we go down one level, the number of subproblems doubles, but the cost per subproblem halves
  $\Rightarrow$ *cost per level remains the same*.
- There are lg n + 1 levels, height is lg n. (Assuming n is a power of 2.)
  - Can be proved by induction.
- Total cost = sum of costs at each level = $(\lg n + 1)cn = cn\lg n + cn = \Theta(n \lg n)$.

Ref Book: Thomas Cormen

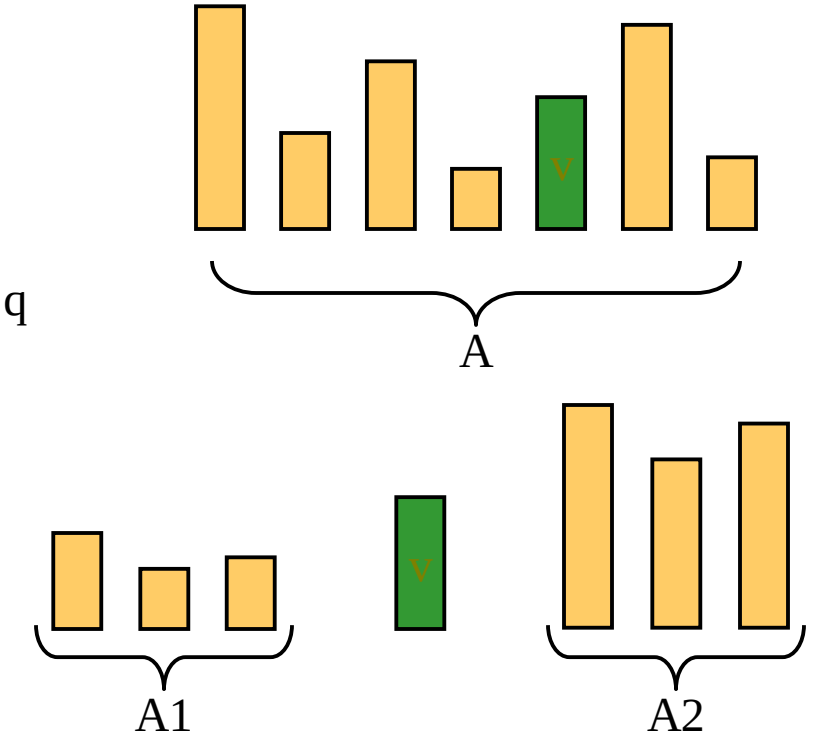
Quicksort

Ref Book: Thomas Cormen

- Divide :
  - Pick any element (***pivot***) v in array A[p…r]
  - Partition array A into two groups

    A1[p - - -q-1] , A2 [q+1----r] Compute the index q

    A1 element < A[q] < A2 element
- Conquer step: recursively sort A1 and A2
- Combine step: the sorted A1 (by the time returned from recursion), followed by A[q], followed by the sorted A2 (i.e., nothing extra needs to be done)



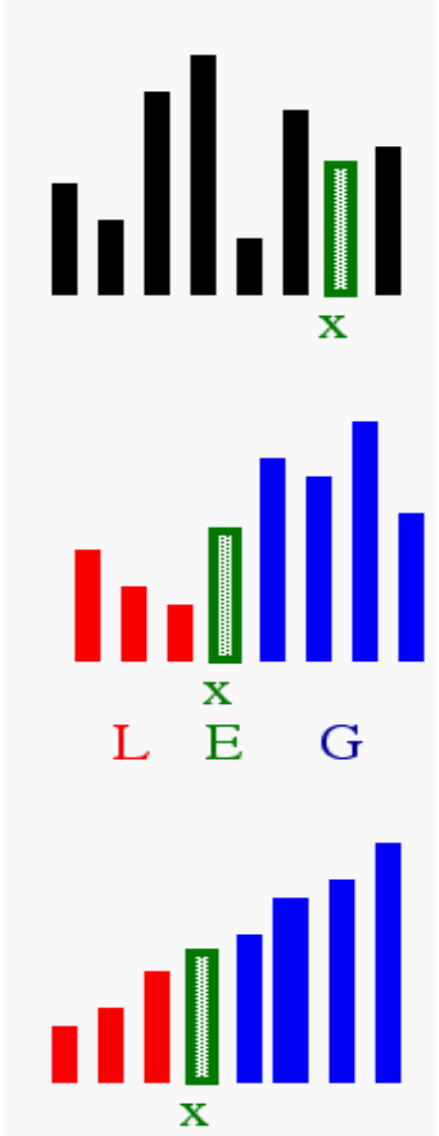
Idea of Quick Sort

Ref Book: Internet

1) **Select**: pick an element

2) **Divide**: rearrange elements so that x goes to its final position E

3) **Recurse and Conquer**: recursively sort



Ref Book: Thomas Cormen

Quicksort Pseudo-code

INPUT: a sequence of *n* numbers stored in array A
OUTPUT: an ordered sequence of *n* numbers

```
QuickSort (A, p, r)   // sort A[p..r] by divide & conquer
1    if p < r
2        then q = Partition (A, p, r)
3             QuickSort (A, p, q-1)
4             QuickSort (A, q+1, r)
```

Initial Call: QuickSort(A, 1, n)

Procedure Partitioning the array

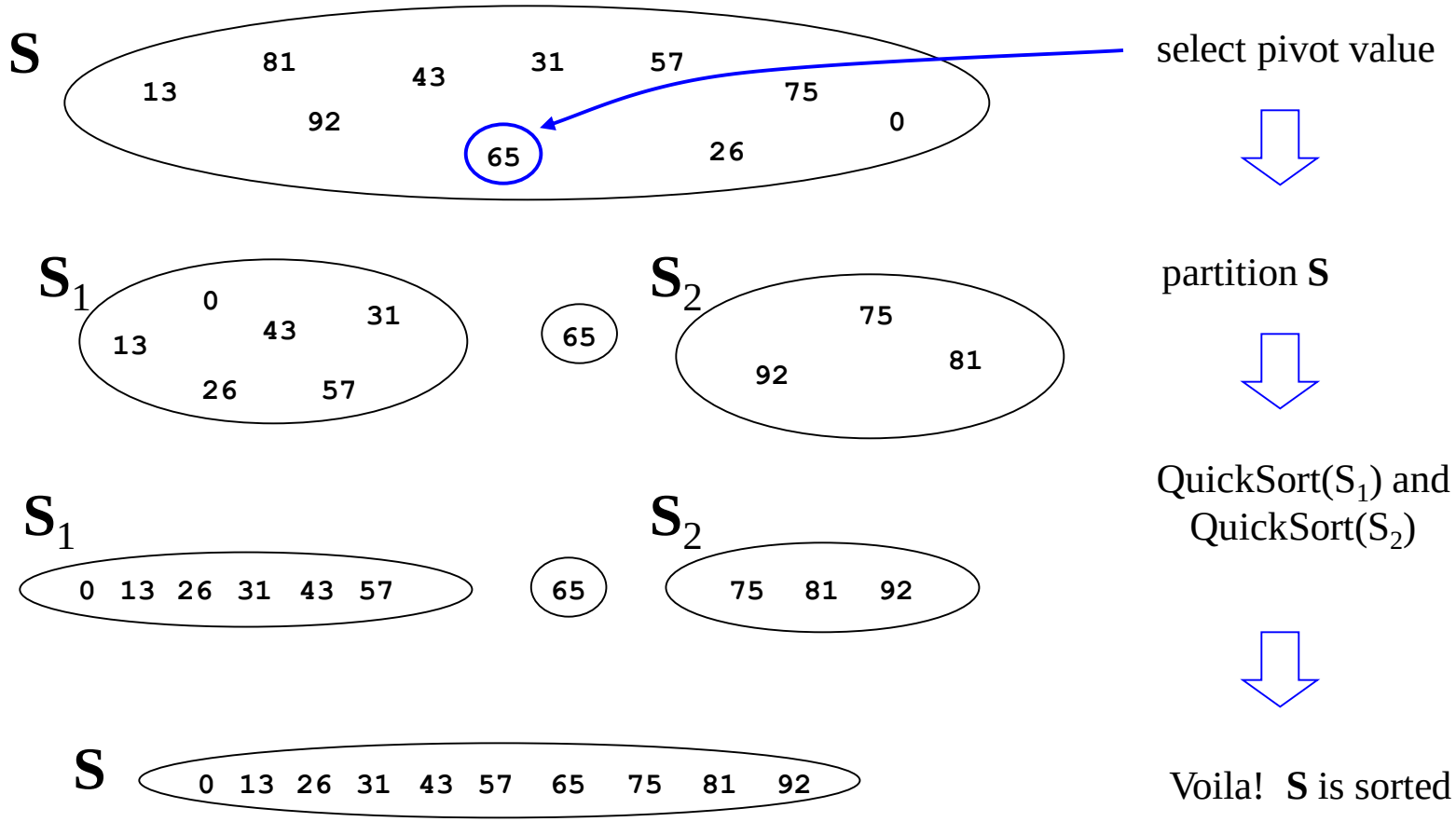
Ref Book: Thomas Cormen

```
Partition(A, p, r)
1  x = A[r]
2  i = p - 1
3      for j ← p to r-1
4          if A[j] ≤ x
5              i= i +1
6          exchange A[i] with A[j]
7      exchange A[i] with A[r]
8      Return i + 1
```

Input: Array containing sorted subarrays $A[p..q]$ and $A[q+1..r]$.

Output: sorted subarray in $A[p..r]$.

The steps of QuickSort



Quicksort Analysis

Assumptions:

- A random pivot (no median-of-three partitioning)
- No cutoff for small arrays

Running time

- pivot selection: constant time, i.e. O(1)
- partitioning: linear time, i.e. O(N)
- running time of the two recursive calls

T(N)=T(i)+T(N-i-1)+cN where c is a constant

- i: number of elements in S1

Worst-Case Analysis

worst case Partition?

- The pivot is the smallest element, all the time
- Partition is always unbalanced

$$
\begin{aligned}
T(N) &= T(N-1) + cN \\
T(N-1) &= T(N-2) + c(N-1) \\
T(N-2) &= T(N-3) + c(N-2) \\
&\vdots \\
T(2) &= T(1) + c(2) \\
T(N) &= T(1) + c\sum_{i=2}^{N} i = O(N^2)
\end{aligned}
$$

Best-case Analysis

best case Partitioning?

- Partition is perfectly balanced.
- Pivot is always in the middle (median of the array)

$$
\begin{aligned}
T(N) &= 2T(N/2) + cN \\
\frac{T(N)}{N} &= \frac{T(N/2)}{N/2} + c \\
\frac{T(N/2)}{N/2} &= \frac{T(N/4)}{N/4} + c \\
\frac{T(N/4)}{N/4} &= \frac{T(N/8)}{N/8} + c \\
&\vdots \\
\frac{T(2)}{2} &= \frac{T(1)}{1} + c \\
\frac{T(N)}{N} &= \frac{T(1)}{1} + c \log N \\
T(N) &= cN \log N + N = O(N \log N)
\end{aligned}
$$

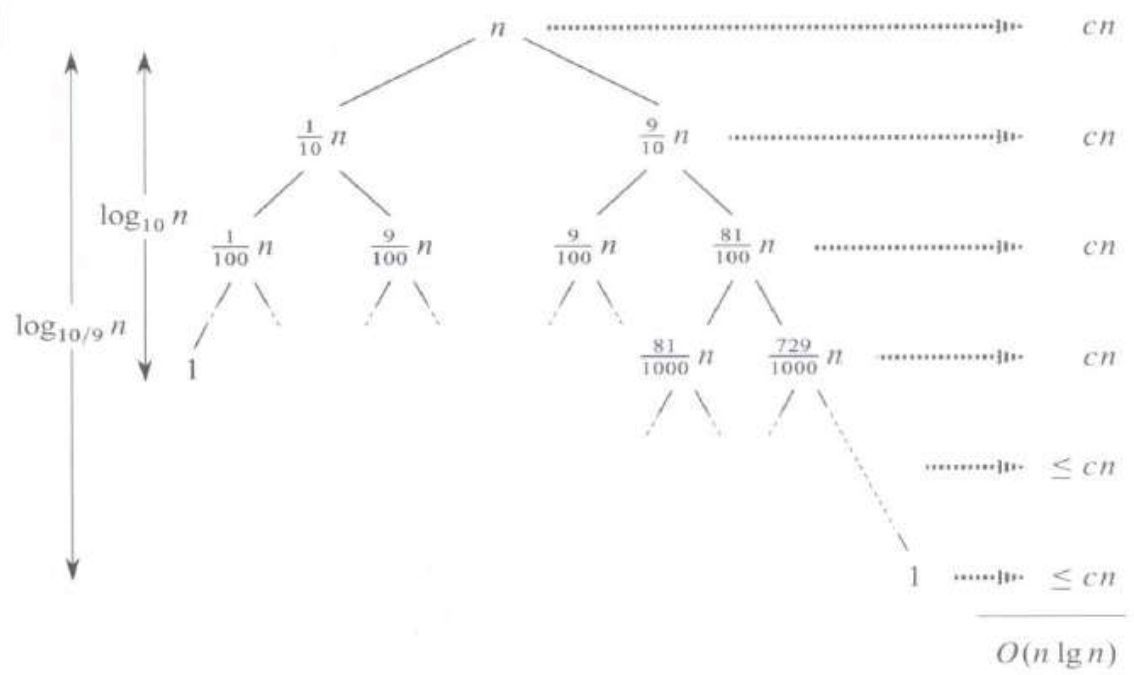
Average-Case Analysis

Intution for Average Case : We can get an idea of average case by considering the case when partition puts O(n/9) elements in one set and O(9n/10) elements in other set.

Following is recurrence for this case.

T(n) = T(n/9) + T(9n/10) + Θ(n)

Solution of above recurrence is also O(nlogn)



REF BOOK: THOMAS CORMEN

Summary Analysis of Quicksort

Best case: split in the middle — $\Theta(n \log n)$

Worst case: sorted array! — $\Theta(n^2)$

Average case: random arrays — $\Theta(n \log n)$

Improvements:

- better pivot selection: median of three partitioning
- switch to insertion sort on small subfiles

Multiplication of Large Integers

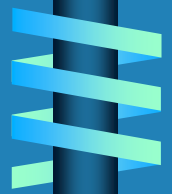
Consider the problem of multiplying two (large) *n*-digit integers represented by arrays of their digits such as:

A = 12345678901357986429 B = 87654321284820912836

The grade-school algorithm:

$$
\begin{array}{r}
a_1\ a_2 \ldots\ a_n \\
b_1\ b_2 \ldots\ b_n \\
\hline
(d_{10})\, d_{11} d_{12} \ldots d_{1n} \\
(d_{20})\, d_{21} d_{22} \ldots d_{2n} \\
\ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots \\
(d_{n0})\, d_{n1} d_{n2} \ldots d_{nn}
\end{array}
$$

Efficiency: $\Theta(n^2)$ single-digit multiplications



First Divide-and-Conquer Algorithm

A small example: $A * B$ where $A = 2135$ and $B = 4014$

$A = (21 \cdot 10^2 + 35)$, $B = (40 \cdot 10^2 + 14)$

So, $A * B = (21 \cdot 10^2 + 35) * (40 \cdot 10^2 + 14)$

$= 21 * 40 \cdot 10^4 + (21 * 14 + 35 * 40) \cdot 10^2 + 35 * 14$

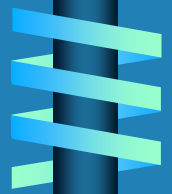
In general, if $A = A_1A_2$ and $B = B_1B_2$ (where A and B are *n*-digit, A_1, A_2, B_1, B_2 are *n*/2-digit numbers),

$A * B = A_1 * B_1 \cdot 10^n + (A_1 * B_2 + A_2 * B_1) \cdot 10^{n/2} + A_2 * B_2$

Recurrence for the number of one-digit multiplications M(*n*):

$$M(n) = 4M(n/2), \quad M(1) = 1$$

Solution: $M(n) = n^2$



Second Divide-and-Conquer Algorithm

$A * B = A_1 * B_1 \cdot 10^n + (A_1 * B_2 + A_2 * B_1) \cdot 10^{n/2} + A_2 * B_2$

The idea is to decrease the number of multiplications from 4 to 3:

$$(A_1 + A_2) * (B_1 + B_2) = A_1 * B_1 + (A_1 * B_2 + A_2 * B_1) + A_2 * B_2,$$

I.e., $(A_1 * B_2 + A_2 * B_1) = (A_1 + A_2) * (B_1 + B_2) - A_1 * B_1 - A_2 * B_2$, which requires only 3 multiplications at the expense of (4-1) extra add/sub.

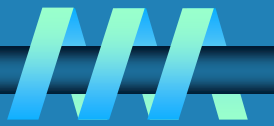
Recurrence for the number of multiplications $M(n)$:

$$M(n) = 3M(n/2), \quad M(1) = 1$$

Solution: $M(n) = 3^{\log_2 n} = n^{\log_2 3} \approx n^{1.585}$

What if we count both multiplications and additions?

Example of Large-Integer Multiplication



2135 * 4014

= (21*10^2 + 35) * (40*10^2 + 14)

= (21*40)*10^4 + c1*10^2 + 35*14

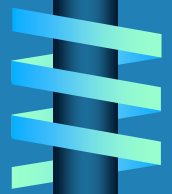
where c1 = (21+35)*(40+14) - 21*40 - 35*14, and

21*40 = (2*10 + 1) * (4*10 + 0)

= (2*4)*10^2 + c2*10 + 1*0

where c2 = (2+1)*(4+0) - 2*4 - 1*0, etc.

This process requires 9 digit multiplications as opposed to 16.



References

1. Thomas H Cormen and Charles E.L Leiserson, "Introduction to Algorithm" PHI Third Edition
2. Horowitz and Sahani, "Fundamentals of Computer Algorithms", 2ND Edition. University Press, ISBN: 978 81 7371 6126, 81 7371 61262.