Order of functions

An analogy between the asymptotic comparison of two functions f and g and the comparison of two real numbers a and b:

$$\begin{array}{ccccccc} f(n) & = & O(g(n)) & \approx & \mathsf{a} & \leq & \mathsf{b} \\ f(n) & = & \Omega(g(n)) & \approx & \mathsf{a} & \geq & \mathsf{b} \\ f(n) & = & \Theta(g(n)) & \approx & \mathsf{a} & = & \mathsf{b} \end{array}$$

Order of functions (cont'd)

Question:

What's the order of the following widely used functions:
lgn, n, n^2, 1, n^3, 2^n, $n2^n$, $(n+1)!$, 2^{2^n}, $(lgn)!$, e^n, $n!$

Answer:

$1 \leq lgn \leq n \leq n^2 \leq n^3 \leq (lgn)! \leq 2^n$

$\leq n2^n \leq e^n \leq n! \leq (n+1)! \leq 2^{2^n}$, where $a \leq b$ means $a = O(b)$

Order of functions (Cont'd)

Suppose one basic operation needs CPU time 0.000001 second.

	10	20	30	40	50	60
n	0.00001 s	0.00002 s	0.00003 s	0.00004 s	0.00005 s	0.00006 s
n^2	0.0001 s	0.0004 s	0.0009 s	0.016 s	0.025 s	0.036 s
n^3	0.001 s	0.008 s	0.027 s	0.064 s	0.125 s	0.216 s
n^5	0.1 s	3.2 s	24.3 s	1.7 min	5.2 min	13.0 min
2^n	0.001 s	1.0 s	17.9 min	12.7 days	35.7 years	366 cent
3^n	0.59 s	58 min	6.5 years	3855 cent	2×10^8 cent	1.3×10^13 cent

A Recurrence Example: Merge Sort

Merge sort is a good example to show how divide and conquer works. The idea is: Given an array A[1..n], divide it into two sub-array A[1..n/2] and A[n/2+1..n]. Each sub-array is individually sorted, and the resulting sub-arrays are merged to produce a single sorted array of n elements. The algorithm:

```
MERGE-SORT(A, p, r)
    if (p == r) return;
    q = (p + r)/2;
    Merge-Sort(A, p, q);
    Merge-Sort(A, q+1, r);
    Merge(A, p, q, r);
```

To sort the whole array, Merge-Sort(A, 1, n) is called.

The operation of Merge Sort

Input: 5, 2, 4, 7, 1, 3, 2, 6

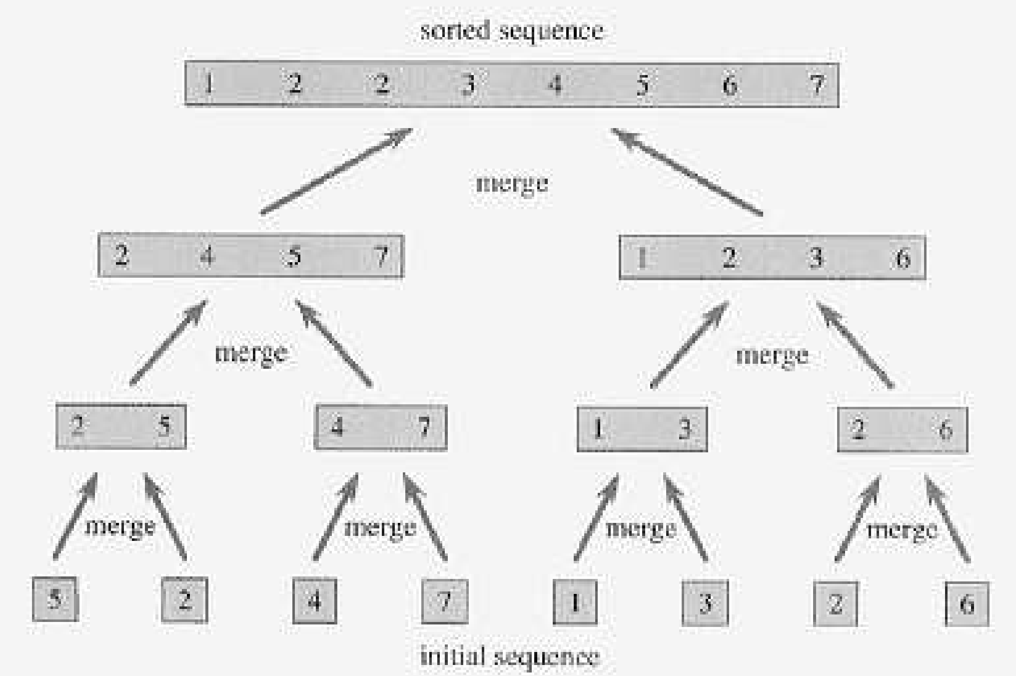

Figure 2.4 The operation of merge sort on the array $A = \langle 5, 2, 4, 7, 1, 3, 2, 6 \rangle$. The lengths of the sorted sequences being merged increase as the algorithm progresses from bottom to top.

Complexity of Merge Sort

Divide: The divide step only compute the middle, takes constant time. $D(n) = \Theta(1)$.

Conquer: Recursively sort 2 subarrays. $C(n) = 2T(n/2)$.

Combine: Merge two $n/2$-element subarrays, takes linear time $\Theta(n)$.

Overall:

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

$$= \begin{cases} C_1 & \text{if } n = 1 \text{ (or smallsize)} \\ 2T(n/2) + C_2 n & \text{if } n > 1 \text{ (or smallsize)} \end{cases}$$

How to solve this recurrence?

Solution 1: Substitution method

1. Guess the form of the solution.
2. Use mathematical induction to find the constants and show the solution works.

Example: Merge Sort

$$T(n) = \begin{cases} C_1 & \text{if } n = 1 \\ 2T(n/2) + C_2 n & \text{if } n > 1 \end{cases}$$

Step 1: give a guess: **T(n) = O(n lg n)**

Step 2: to show $\exists$ const c and n_0, such that $T(n) \leq c \cdot nlgn$ for all $n \geq n_0$

Base case:

T(1) = $C_1 \leq$ c 1 lg 1 = 0.... Impossible

Take T(2) as the base case.

T(2) = $2C_1 + 2C_2 \leq$ c 2 lg 2 = 2c

as long as c $\geq (C_1 + C_2)$.

Cont'd

Induction Step:

Suppose there exist a constant c such that

$T(n) \leq c \cdot nlgn$ for all n = 2, 3,...., k-1

We want to show $T(n) \leq c \cdot nlgn$

holds for n = k.

T(k) = 2T(k/2) + C_2k   Note: k/2 is in {2, 3. .., k-1}

$\leq$ 2(c (k/2) lg (k/2)) + C_2k

= ck lg k - ck lg2 + C_2k

= ck lg k - ck + C_2k

$\leq$ ck lg k   as long as $c \geq C_2$.

So we pick n_0 = 2, $c = C_1 + C_2$,

$T(n) \leq c \cdot nlgn$ for all $n \geq n_0 \Longrightarrow T(n) = O(nlgn)$.

Where to get the good guess?

Solution 2: Iteration/Recursion tree method: used to generate a good guess; can also be used as a direct proof.

Example:

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

$$\begin{aligned}
T(n) &= 2T(\tfrac{n}{2}) + n \\
&= 2(2T(\tfrac{n}{4}) + \tfrac{n}{2}) + n \\
&= 2^2T(\tfrac{n}{2^2}) + n + n \\
&= 2^2(2T(\tfrac{n}{2^3}) + \tfrac{n}{2^2}) + 2n \\
&= 2^3T(\tfrac{n}{2^3}) + 3n \\
&\ldots \\
&= 2^iT(\tfrac{n}{2^i}) + in
\end{aligned}$$

Cont'd

Question: When will the iteration procedure reach the boundary condition (hit the ground)?

Answer: $(n/2^i) = 1 \iff i = lgn$

Then
$$
\begin{aligned}
T(n) &= 2^{lgn}\, T(1) + lgn \times n \\
&= n + nlgn \\
&= \Theta(nlgn).
\end{aligned}
$$