

Example

$$1) n^3 + 2^{1000}n^2 + 2^{100000}n = O(n^3) = \Omega(n^3) = \Theta(n^3)$$

2) In general for any polynomial

$$p(n) = \sum_{i=0}^d a_i n^i$$

where a_i are constants and $a_d > 0$, we have $p(n) = \Theta(n^d)$.

$$3) n = O(n^2)$$

$$4) n^{1000} = O(1.1^n)$$

$$5) e^x = 1 + x + \Theta(x^2), \text{ as } x \rightarrow 0.$$

6)

$$f(n) = \begin{cases} n^2 & : \text{ n is even} \\ n^3 & : \text{ n is odd} \end{cases}$$

$$g(n) = n^3$$

$$f(n) = O(g(n))$$

$$\limsup_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq 1.$$

Example

Theorem 1. *Let $T(n)$ be the best running time of an algorithm for finding the maximum of n numbers stored in an unsorted array, then $T(n) = \Theta(n)$.*

Proof.

1. Lower Bound: If the algorithm makes less than n comparisons, then an *adversary* can make sure that an unseen element is the largest.

2. Upper Bound: The following algorithm finds the maximum in no more than n comparisons:

1. $Max = A[1]; I = 1;$

2. While $I < n$ do

(a) $I := I + 1;$

(b) If $MAX < A[I]$ then $MAX := A[I];$

□

More notation

$g(n) = o(f(n))$ if for any positive constant c there is a constant n_c such that for any $n > n_c$, $g(n) < cf(n)$.

$$\limsup_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0.$$

$g(n) = \omega(f(n))$ if for any positive constant c there is a constant n_c such that for any $n > n_c$, $f(n) > cg(n)$.

$$\liminf_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty.$$

Example

1) $1/n = o(1)$

2) $n^{1000} = o(1.1^n)$

3) $n^2 = \omega(n)$

4) $n = (?)n^{1+\sin n}$

$$\limsup_{n \rightarrow \infty} \frac{n}{n^{1+\sin n}} = \infty$$

$$\liminf_{n \rightarrow \infty} \frac{n}{n^{1+\sin n}} = 0$$

5) Finding the $\lfloor \frac{n}{2} \rfloor$ -largest element in an array of n elements takes $o(n^2)$ steps.

We can sort in $O(n \log n)$ time.

Recurrences

Algorithmic analysis often involves solving recurrences. This is especially typical in recursive algorithms.

$$\text{Solve } T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

Guess $T(n) = O(n)$. Let's use **induction** to show that $T(n) \leq cn$.

$$T(n) \leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1$$

$= cn + 1$ which does not prove our hypothesis.

Guess $T(n) \leq cn - b$. Then

$$T(n) \leq (c\lfloor n/2 \rfloor - b) + (c\lceil n/2 \rceil - b) + 1$$

$$= cn - 2b + 1$$

$$\leq cn - b, \text{ if } b \geq 1.$$

And c should be chosen large enough to satisfy the boundary conditions.

Another Example

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

Guess and verify that $T(n) \leq cn \log n$, for some constant c .

$$T(n) \leq 2cn/2 \log n/2 + n$$

$$= cn \log n - cn \log 2 + n$$

$$= cn \log n - cn + n$$

$$\leq cn \log n \text{ if } c \geq 1.$$

However, if you try to show $T(n) \leq cn$ by arguing:

$$T(n) \leq 2cn/2 + n = (c + 1)n = O(n)$$

This is incorrect!

Change of Variables

Solve $T(n) = 2T(\sqrt{n}) + 1$

Renaming $m = \log n$:

$$T(2^m) = 2T(2^{m/2}) + 1$$

Renaming $S(m) = T(2^m)$:

$$S(m) = 2S(m/2) + 1$$

Thus, $S(m) = O(m)$ and $T(n) = T(2^m) = S(m) = O(m) = O(\log n)$.

A General Theorem for “Divide and Conquer” Recurrences

Consider recurrences of the form

$T(n) = aT(n/b) + f(n)$ where a and $b > 1$ are constants and $f(n)$ is some function.

Theorem 2. [“Master” Theorem] *The solution for the above recurrence $T(n)$ is:*

1) *If $af(n/b) = cf(n)$ for some constant $c < 1$ then $T(n) = \Theta(f(n))$.*

2) *If $af(n/b) = cf(n)$ for some constant $c > 1$ then $T(n) = \Theta(n^{\log_b a})$.*

3) *If $af(n/b) = f(n)$ then $T(n) = \Theta(f(n) \log_b n)$.*

Examples

1. Randomized Selection: $T(n) = T(3n/4) + 2n$

Here $af(n/b) = 2(3n/4) = (3/4)f(n)$

Hence $T(n) = \Theta(n)$.

2. Strassen's Matrix Multiplication: $T(n) = 7T(n/2) + \Theta(n^2)$

That is, $T(n) = 7T(n/2) + c_1n^2$, for some positive constant c_1 .

$af(n/b) = 7c_1(n/2)^2 = (7/4)c_1n^2 = (7/4)f(n)$

Hence, $T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$

3. MergeSort: $T(n) = 2T(n/2) + n$

Here $af(n/b) = f(n)$ and hence $T(n) = \Theta(n \log n)$.