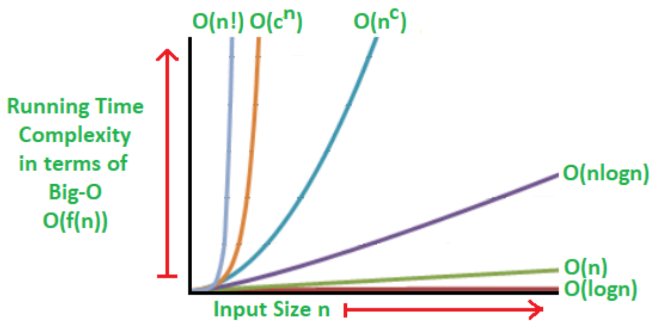
Asymptotic Complexity



O : upper bound

Ω : lower bound

Θ : tight bound

*Note: Big-O ≠ "worst case" and Big-Omega ≠ "best case"
A function can run in n^2 time, and $O(n!)$ is still a valid upper bound

Common Series:

Geometric: $x_n = x_0 r^{n-1}$

$$\begin{cases} r = 1 \rightarrow \Theta(n) & \Theta(n x_0) = \Theta(n) \\ r < 1 \rightarrow \Theta(1) & \text{first term dominates} \\ r > 1 \rightarrow \Theta(r^n) & \text{last term dominates} \end{cases}$$

$$1 + 2 + 3 + \ldots + n-1 + n = \frac{n(n+1)}{2} = \Theta(n^2)$$

*simplify expressions before dropping terms

1 Asymptotic Complexity Comparisons

(b) In each of the following, indicate whether $f = O(g)$, $f = \Omega(g)$, or both (in which case $f = \Theta(g)$). ***Briefly*** justify each of your answers. Recall that in terms of asymptotic growth rate, logarithmic < polynomial < exponential.

"f is upper bounded by g"

	$f(n)$	$g(n)$
(i)	$\log_3 n$	$\log_4(n)$
(ii)	$n\log(n^4)$	$n^2\log(n^3)$
(iii)	$\sqrt{n}$	$(\log n)^3$
(iv)	$n + \log n$	$n + (\log n)^2$

i) change of base formula

$\log_3 n = \frac{\log n}{\log 3}$ $\quad \log_4 n = \frac{\log n}{\log 4}$ $\quad f = \Theta(g)$

constants

ii) $f(n) = 4n\log n$ $\quad g(n) = 3n^2\log n$

$f = O(g)$

iii) polynomials dominate logs

$f = \Omega(g)$

iv) $f = \Theta(n)$ $\quad g = \Theta(n)$

$f = \Theta(g)$

2 Bit Counter

Consider an n-bit counter that counts from 0 to $2^n - 1$. As it moves from x to $x+1$, it tracks how many bits were flipped from x.

When $n = 5$, the counter has the following values:

Step	Value	# Bit-Flips
0	00000	–
1	00001	1
2	00010	2
3	00011	1
4	00100	3
$\vdots$	$\vdots$	
31	11111	1

find sum of this as function of n

For example, the last two bits flip when the counter goes from 1 to 2. Using $\Theta(\cdot)$ notation, find the growth of the *total* number of bit flips (the sum of all the numbers in the "# Bit-Flips" column) as a function of n.

times i^{th} bit from left flips : 2^i

$$\sum_{i=1}^{n} (2^i) = 2^1 + 2^2 + \ldots + 2^{n-1} + 2^n = \Theta(2^n)$$

3 Asymptotic Bound Practice

Prove that for any $\epsilon > 0$ we have $\log x \in O(x^\epsilon)$.

want to show: $\lim_{x\to\infty} \frac{\log x}{x^\epsilon} = 0$, aka x^ϵ grows asymptotically faster than $\log x$

$$\lim_{x\to\infty} \frac{\log x}{x^\epsilon} = \lim_{x\to\infty} \frac{\frac{d}{dx}\log x}{\frac{d}{dx}x^\epsilon}$$ using l'Hôpital's rule for ∞/∞

$$= \lim_{x\to\infty} \frac{1/x}{\epsilon x^{\epsilon-1}}$$

$$= \lim_{x\to\infty} \frac{1}{\epsilon x^{\epsilon-1}} = 0$$

Divide and Conquer

Break problem down into independent parts → Solve each part separately → Piece together results

"top down"

ex: binary search

- do work that influences your split
- answer at the end

"bottom up"

ex: merge sort

- split into subproblems first
- work is done as you combine smaller problems

4 Hadamard matrices

The Hadamard matrices $H_0, H_1, H_2, \ldots$ are defined as follows:

- H_0 is the 1×1 matrix $[1]$
- For $k > 0, H_k$ is the $2^k \times 2^k$ matrix

$$H_k = \left[\begin{array}{c|c} H_{k-1} & H_{k-1} \\ \hline H_{k-1} & -H_{k-1} \end{array}\right]$$

(a) Write down the Hadamard matrices H_0, H_1, and H_2.

$H_0 = 1$

$$H_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

(b) Compute the matrix-vector product $H_2 \cdot v$ where H_2 is the Hadamard matrix you found above, and

$$v = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

Note that since H_2 is a 4×4 matrix, and the vector has length 4, the result will be a vector of length 4.

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 4 \end{bmatrix}$$

(c) Now, we will compute another quantity. Take v_1 and v_2 to be the top and bottom halves of v respectively. Therefore, we have that

$$v_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, v_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

Compute $u_1 = H_1(v_1 + v_2)$ and $u_2 = H_1(v_1 - v_2)$ to get two vectors of length 2. Stack u_1 above u_2 to get a vector u of length 4. What do you notice about u?

$$u_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$u_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 2 \\ -2 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 4 \end{bmatrix} = H_2 v$$

(d) Suppose that

$$v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

is a column vector of length $n = 2^k$. v_1 and v_2 are the top and bottom half of the vector, respectively. Therefore, they are each vectors of length $\frac{n}{2} = 2^{k-1}$. Write the matrix-vector product $H_k v$ in terms of H_{k-1}, v_1, and v_2 (note that H_{k-1} is a matrix of dimension $\frac{n}{2} \times \frac{n}{2}$, or $2^{k-1} \times 2^{k-1}$). Since H_k is a $n \times n$ matrix, and v is a vector of length n, the result will be a vector of length n.

$$H_k v = \begin{bmatrix} H_{k-1}(v_1 + v_2) \\ H_{k-1}(v_1 - v_2) \end{bmatrix}$$

(e) Use your results from (c) to come up with a divide-and-conquer algorithm to calculate the matrix-vector product $H_k v$, and show that it can be calculated using $O(n \log n)$ operations. Assume that all the numbers involved are small enough that basic arithmetic operations like addition and multiplication take unit time. You do not need to prove correctness.

2 subproblems, each half the size

$O(n)$ time to find vectors $v_1 + v_2$ and $v_1 - v_2$

$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

$T(n) = aT(\lceil n/b \rceil) + O(n^d)$

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

master's theorem: $O(n \log n)$

5 Extra Divide and Conquer Practice: Sorted Array

Given a sorted array A of n (possibly negative) distinct integers, you want to find out whether there is an index i for which $A[i] = i$. Devise a divide-and-conquer algorithm that runs in $O(\log n)$ time.

Binary search

if A empty:
    return False

if $A[\frac{n}{2}] = i$:
    return True
if $A[\frac{n}{2}] > \frac{n}{2}$
    search left half
if $A[\frac{n}{2}] < \frac{n}{2}$
    search right half

3 Asymptotic Bound Practice

Prove that for any $\epsilon > 0$ we have $\log x \in O(x^\epsilon)$.

4 Hadamard matrices

The Hadamard matrices $H_0, H_1, H_2, \ldots$ are defined as follows:

- H_0 is the 1×1 matrix $[1]$
- For $k > 0, H_k$ is the $2^k \times 2^k$ matrix

$$H_k = \left[\begin{array}{c|c} H_{k-1} & H_{k-1} \\ \hline H_{k-1} & -H_{k-1} \end{array}\right]$$

(a) Write down the Hadamard matrices H_0, H_1, and H_2.

(b) Compute the matrix-vector product $H_2 \cdot v$ where H_2 is the Hadamard matrix you found above, and

$$v = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

Note that since H_2 is a 4×4 matrix, and the vector has length 4, the result will be a vector of length 4.

(c) Now, we will compute another quantity. Take v_1 and v_2 to be the top and bottom halves of v respectively. Therefore, we have that

$$v_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, v_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

Compute $u_1 = H_1(v_1 + v_2)$ and $u_2 = H_1(v_1 - v_2)$ to get two vectors of length 2. Stack u_1 above u_2 to get a vector u of length 4. What do you notice about u?

(d) Suppose that

$$v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

is a column vector of length $n = 2^k$. v_1 and v_2 are the top and bottom half of the vector, respectively. Therefore, they are each vectors of length $\frac{n}{2} = 2^{k-1}$. Write the matrix-vector product $H_k v$ in terms of H_{k-1}, v_1, and v_2 (note that H_{k-1} is a matrix of dimension $\frac{n}{2} \times \frac{n}{2}$, or $2^{k-1} \times 2^{k-1}$). Since H_k is a $n \times n$ matrix, and v is a vector of length n, the result will be a vector of length n.

(e) Use your results from (c) to come up with a divide-and-conquer algorithm to calculate the matrix-vector product $H_k v$, and show that it can be calculated using $O(n \log n)$ operations. Assume that all the numbers involved are small enough that basic arithmetic operations like addition and multiplication take unit time. You do not need to prove correctness.

5 Extra Divide and Conquer Practice: Sorted Array

Given a sorted array A of n (possibly negative) distinct integers, you want to find out whether there is an index i for which $A[i] = i$. Devise a divide-and-conquer algorithm that runs in $O(\log n)$ time.