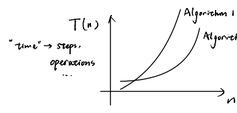
Asymptotic Notation



Upper Bound	$O(f(n))$	$\exists\ c>0,\ n_0\geq 0,$ for all $n\geq n_0.\ T(n)\leq c\cdot f(n)$
Lower Bound	$\Omega(f(n))$	$\exists\ c>0,\ n_0\geq 0,$ for all $n\geq n_0.\ T(n)\geq c\cdot f(n)$
Tight Bound	$\Theta(f(n))$	$\exists\ c_1, c_2>0\ n_0\geq 0,$ for all $n\geq n_0\ c_1 f(n)\leq T(n)\leq c_2 f(n)$

Big-O. proof (By definition)

$\exists\ c, n_0,\ n\geq n_0, c>0,\ T(n)\leq c\cdot f(n) \Rightarrow T(n)=O(f(n))$

e.g. $T(n)=n$. $g(n)=n\log_2 n$ Prove that $T(n)=O(g(n))$

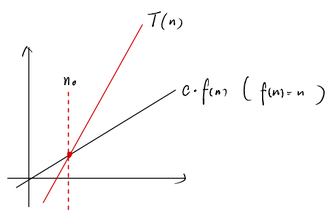
We are going to find c n_0 $(c>0)$ So that when $n\geq n_0$. $T(n)\leq c\cdot g(n)$

i.e. $n\leq c\cdot n\log_2 n$ $(n>n_0)$

$c\log_2 n\geq 1$.

when $c=1$, $n_0=2$, $c\log_2 n\geq 1$

Hence. $T(n)=O(g(n))$



Proof By Contradiction

e.g. $T(n)=n^2$, $g(n)=n$.

We proof that $T(n)\neq O(g(n))$

Suppose that $T(n)=O(g(n))$

So that $\exists\ c>0, n_0\geq 0,\ n>n_0,\ T(n)\leq c\cdot g(n)$

i.e. $n^2\leq c\cdot n$ when $n>n_0$.

$n\leq c$ for $\forall\ n\geq n_0$.

Since n can be arbitrarily large.

the inequality does not satisfied

So $T(n)\neq O(g(n))$

Big-Ω "lower bound"

Big-Θ $T(n)=\Theta(gn)$ iff $T(n)=O(g(n))$ and $T(n)=\Omega(g(n))$

Remark:

① if A. B has same asymptotic running time

We cannot determine which one is better.

e.g. $A(n)=n$. $B(n)=1000n$ ⟶ experiment. implement

② if A is asymptotically slower than B

A is very likely slower than B

Exercise:

(1) $1000n+n\log n=\Theta(n\log n)$

$A=\log\lceil n \quad B=\sqrt{\log n}$

See $m=\lceil n$ then $A=\log m=\Theta(\log m)$

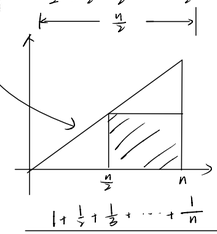
$B=\sqrt{2\log m}=\Theta(\sqrt{\log m})$

$\therefore A=\Theta(B)$

(2) $\sum_{i=1}^{n} i=1+2+3+4+\cdots+n=\Theta(n^2)$

$\sum_{i=1}^{n} i<n\cdot n=n^2$

$\sum_{i=1}^{n} i>\frac{n}{2}+\frac{n}{2}+\frac{n}{2}+\cdots+\frac{n}{2}=\frac{n^2}{4}$



(3) $1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}$

	Lower Bound		Upper Bound
for $n=2^k$	$\frac{1}{2}$	1	1
	$\frac{1}{4}\cdot 2$	$\frac{1}{3}+\frac{1}{4}$	$\frac{1}{2}\cdot 2$
$k=\log_2 n$	$\frac{1}{8}\cdot 4$	$\frac{1}{5}+\frac{1}{6}+\frac{1}{7}+\frac{1}{8}$	$\frac{1}{4}\cdot 4$
	⋮	⋮	⋮
	$2^{k-1}\cdot\frac{1}{2^k}$	$+\frac{1}{2^k-1}$	1

$\therefore \frac{1}{2}\log_2 n<1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}<\log_2 n$

$\therefore \Theta(\log_2 n)$

☆ $2^{2n}=(2^n)^2\neq\Theta(2^n)$

$2^{n+2}=4\cdot 2^n=\Theta(2^n)$

☆ $\Rightarrow \Theta(2^{n+k})=\Theta(2^n)$

$\log_2(n^3)=3\log_2(n)=\Theta(\log n)$

☆ $\Theta(\log(n^c))=\Theta(\log n)$

$\log_{10} n=\frac{\log_2 n}{\log_2 10}=\Theta(\log n)$

☆ $\Theta(\log_k n)=\Theta(\log n)$

$\log_b n\log_b a=\log_b n\log_b a$

☆ $a^{\log_b n}=n^{\log_b a}$ (Since $\log_b a^{(\log_b n)}=\log_b n^{\log_b a}$)

☆ $n\log n=O(\frac{n^2}{\log n})$

proof:

$\frac{n\log n}{\frac{n^2}{\log n}}=\frac{(\log n)^2}{n}$

$\lim_{n\to\infty}\frac{(\log n)^2}{n}=0$ $(\frac{\infty}{\infty})$ ⟶ $\exists\ n_0$, when $n\geq n_0$. $n\log n<\frac{(\log n)^2}{n}$

Growth of functions

constant < "log" < polynomial < exponential.

Remark

any "Polynomial"'s growth is faster than any "log" (e.g. $\log_2 n=O(n^{\frac{1}{1000}})$ ☆)

Introduction

Addition

$x=\overline{x_n x_{n-1}\cdots x_1}$

$y=\overline{y_n y_{n-1}\cdots y_1}$ ⟶ $z=\overline{z_{n+1} z_n\cdots z_1}$

```
c ← 0
for i ← 1 to n
    z_i ← x_i + y_i + c
    if z_i > 10 then z_i ← z_i − 10
                     c ← 1
```

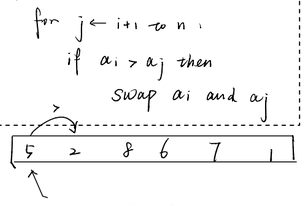
Sorting Problem

Select the minimum

① Selection Sort.

```
for i ← 1 to n−1:
    for j ← i+1 to n:
        if a_i > a_j then
            swap a_i and a_j
```

1. Find minimum in array A[1...n]
   Let A[1] = $A^{(1)}_{min}$
2. Find minimum in array A[2...n]
   Let A[2] = $A^{(2)}_{min}$
   ⋮
n-1. Find minimum in array A[n-1, n]
   Swap A[n-1] and A_{min}



1. (n-1) Comparison
2. (n-2) comparison
⋮
n-1. 1 comparison

→ $1+2+3+\cdots+(n-1)=\frac{n(n-1)}{2}$ times of Comparison

Remarks: A[i] (e.g. A[1]) always refers to the first elements of the array.

i.e. if 5 ⇔ 2. then A[1] = 5 → A[1] = 2

Proof (by induction)

When n=1. obviously correct.

if for n=m-1. the algorithm is correct.

Want to show n=m, the algorithm is correct

First we find A_{min} in array A[1...m]

and Swap A_1, A_{min}

Then We are Sorting array A[2...m] (Can be Sorted By inductive Hypothesis)

Which is sorting array of length m-1

Hence the algorithm is correct when n=m

⇒ the algorithm is correct

Insertion Sort

```
for i ← 2 to n do:
    j ← i − 1
    while j ≥ 1 and A[j] > A[j+1] do
        swap A[j] and A[j+1]
        j ← j − 1
```

Remark: if $A[j]\leq A[j+1]$ at first then directly jump out to next iteration!

Comparison and Swap

- ascending: (n-1) times. (only when $A[j]>A[j+1]$)
- descending: $\frac{n(n-1)}{2}$ times

i=2: (5 1 8 6 3 2)→(1 5 8 6 3 2)→(1 5 8 6 3 2)

i=3: (1 5 8 6 3 2)→(1 5 8 6 3 2)

i=4: (1 5 8 6 3 2)→(1 5 6 8 3 2)→(1 5 6 8 3 2)

i=5: (1 5 6 8 3 2)→(1 5 6 3 8 2)→(1 5 3 6 8 2)→(1 3 5 6 8 2)→(1 3 5 6 8 2)

i=6: (1 3 5 6 8 2)→(1 3 5 6 2 8)→(1 3 5 2 6 8)→(1 3 2 5 6 8)→(1 2 3 5 6 8)→(1 2 3 5 6 8)

Evaluate an Algorithm

① Memory (Space Complexity)

② Time (Time complexity)

Method: 1. empirical (by implementation)

2. analytical

Comparing two algorithms is not (usually) a simple matter!

- Even for same input size n, different inputs can lead to different running times
- Calculating exact # of instructions executed is very difficult/tedious

- Selection Sort: always $\frac{n(n-1)}{2}$ Comparisons.
- Insertion Sort: ranging from (n-1) to $\frac{n(n-1)}{2}$

Best / Worse / Average Case analysis

For "Insertion Sort":

Best Case: Already Sorted

⇒ n-1 times of comparison

Worst Case:

⇒ $\sum_{i=1}^{n-1} i=\frac{n(n-1)}{2}$ times of comparison

☆ Average Case:

We assume that: For all permutation of array,

They are equal likely (i.e. Uniform Distribution)

Then. for each iteration ↗ Best: 1 time comparison

↘ Worse: i-1 times.

On Average $\frac{i-1}{2}$ times (e.g. i=5. then E(time) = $1\times\frac{1}{4}+2\times\frac{1}{4}+3\times\frac{1}{4}+4\times\frac{1}{4}=\frac{5}{2}$?

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