I) Introduction

- practical reason: avoid performance bugs

eg: FFT: from $O(n^2) \rightarrow O(n \log n)$

enabled new technology | DVD, JPEG, MRI..

N-body simulation

$$O(n^2) \rightarrow O(n \log n)$$

simulate gravitational interactions among n bodies

Scientific method

- observe
- hypothesize
- predict
- verify
- validate

$O(n^2)$ $\qquad\qquad$ $O(n \log_2 n)$

$\cancel{n} \cdot n$ $\qquad\qquad$ $\cancel{n} \cdot \underbrace{\log_2 n}$

10^6 $\qquad\qquad$ $\log_2 10^6$

$2^{19} < 10^6 < 2^{20}$

$$\boxed{19 < \log_2 n < \underline{\underline{20}}}$$

$10^6 \;/\; 20 = 10^5/2$

$= 50000$

2) Observations

given 8 distinct integers, how many triplets sum to exactly 0?

$$\boxed{T(n) = a\,n^b}$$

$1.3 / 16\,000^2 \simeq$

$1.3 / (16^2 \cdot 10^6)$ $\qquad$ $\frac{5}{10^3}$

$\frac{1.3}{16^2} \cdot \frac{1}{10^6} = 0.005 \cdot \frac{1}{10^6}$

$$\lg(T(N)) = b \cdot \lg N + c$$

$$\Rightarrow T(N) = a \cdot N^b, \quad \text{where } \boxed{a = 2^c}$$

$$\log_2 T(N) = b \cdot \log_2 N + c$$

$$2^{b \cdot \log_2 N + c} = T(N)$$

$$2^{b \cdot \log_2 N} \cdot 2^c = T(N)$$

$$\underbrace{\left(2^{\log_2 N}\right)^b \cdot 2^c} = T(N)$$

$$N^b \cdot a = T(N)$$

$$\Rightarrow \boxed{T(N) = aN^b}$$

III) Mathematical Models

Total running time: sum of cost × frequency for all operations

* String concatenation in Java is not constant time but $O(c \cdot N) \to O(N)$

Cost model: use some basic operation as a proxy for running time

- drop non-dominant terms

$f(N) \sim g(N) \rightarrow \boxed{\lim_{N \to \infty} \frac{f(N)}{g(N)} = 1}$

- drop constants

$\binom{N}{3} = \frac{N(N-1)(N-2)}{3!} \sim \frac{1}{6} N^3$

---

a) $1 + 2 + .. + N \rightarrow \sum_{i=1}^{N} i = \frac{n(n+1)}{2}$

$\sim \boxed{\frac{1}{2} N^2} \rightarrow \boxed{N^2}$

b) $1 + 1/2 + 1/3 + .. + 1/N \rightarrow \sum_{i=1}^{N} \frac{1}{i} \sim \boxed{\ln N}$

$\ln x = \log_e x$

$\boxed{e = 2.71828}$

c) 3-sum triple loop $\sum_{i=1}^{N} \sum_{j=i}^{N} \sum_{k=j}^{N} 1 \sim$

$\boxed{\frac{1}{6} N^3}$

eg for (i from 0 to n)

for (j from i+1 to n)

for (k from 0 to n, step $\boxed{k = k * 2}$)

$\lg N$

$\therefore$

$$\left[1 + 2 + \ldots + (N-1)\right] \lg N$$

$$= \boxed{\frac{(n-1)n}{2} \lg N} \times \text{access operations} = 3$$

$$\Rightarrow \boxed{O\left(\frac{3}{2} N^2 \cdot \lg N\right)} \sim$$

IV) Order-of-Growth Classifications

- small set of functions

  $1, \log N, N, N \cdot \log N, N^2, N^3, 2^N, N!$

- $N \log N$ = linearithmic

(*) Binary search: mathematical analysis

- Binary search uses at most $1 + \lg N$ compares to search in a sorted array of N size
- $T(N)$ = # compare to binary search in a sorted subarray of size $\leq N$
- Binary search recurrence

$$T(N) \leq T(N/2) + 1 \quad \text{for } N > 1$$

splitting in half $\qquad T(1) = 1$

$$\begin{aligned} T(N) &\leq T(N/2) + 1 \\ &\leq T(N/4) + 1 + 1 \\ &\leq T(N/8) + 1 + 1 + 1 \\ &\;\;\vdots \\ &\leq T(N/N) + 1 + 1 + 1 \ldots + 1 \\ &= \underline{\underline{1 + \lg N}} \end{aligned}$$

$\Rightarrow$ we can develop a $N^2 \log N$ algorithm for the 3-sum problem

- sort N distinct Numbers
- for each pair of numbers a[i], a[j]

we do a binary search for $\boxed{-(a[i]+a[j])}$

=> if we find it we print out the values

Theory of Algorithms

- best case : lower bound on cost
- worst case : upper bound on cost

notation	provides	used to
Big Theta Θ	asymptotic order of growth	classify algorithms
Big Oh O	$\Theta(N^2)$ and smaller	develop upper bounds
Big omega Ω	$\Theta(N^2)$ and larger	develop lower bounds

$\boxed{\text{Eg } 1}$ 1-SUM = "Is there a 0 in the array"

- upper bound $O(N)$ $\leq$
- lower bound = proof that no algorithm can do better $\Omega(N)$ $\geq$
- optimal $\Theta(N)$ tight bound runtime

$\boxed{\text{Eg. } 2}$ 3-SUM

- upper: $O(N^3)$ → we found a better one $O(N^2 \log N)$
- lower bound → $\Omega(N)$ we have to examine all N entries (at least)
- optimal – we don't know

$\boxed{\text{Tilde notation}}$ ~ provide approximate model

↳ optimal, similar to Θ but containing the coefficient of the leading term.

Memory

bit $<{}^{0}_{1}$, byte = 8 bits

MB $= 2^{20}$

GB $= 2^{30}$

32-bit machine - 4 byte

64-bit machine - 8 byte

pointers

types	bytes
boolean	1
byte	1
char	2
int	4
float	4
long	8
double	8

char[ ]

2N + 24 bytes

24 → overhead

- object overhead : 16 bytes
- reference : 8 bytes
- padding : multiple of 8 bytes

Date class	Object
int day → 4 bytes
int month → 4 bytes
int year → 4 bytes

16 bytes overhead

4 bytes of padding (multiple of 8)

→ 32 bytes

- array : 24 bytes + memory for each entry N x size of entry

$\boxed{\text{Tilde}}$ only drop non-dominant terms.