together { lab test 2 → 3 questions
quiz → all topics covered, analysis of algorithm for sure
↳ from tutorials

Algorithms

- well-defined, unambiguous - set of instructions to produce outputs for legitimate inputs in a finite amount of time

- correct and consistent, precise and unambiguous, finiteness

- program (in certain lang) is an instance of an algo

practice - Fibonacci sequence algo w/ time complexity log n

- will cover: searching, graph, combinatorial,
  sorting, string processing
  text matching, genone, pattern recognition ← string processing
  combinatorial ↓ arrangement, pattern, design, assign, schedule, connect, configure.
  cryptography, match - cover ← configure.

stability in sorting: stable sorting algorithms sort repeated elements in the same order that they appear in the input

A	70	descending →	C	80
B	70		A	70
C	80		B	70

- algorithm design strategies
  - → brute force & exhaustive search (traverse list)
  - → divide & conquer (binary tree (kinda))
  - → greedy strategy
  - → decrease & conquer (binary search)
  - → transform & conquer (transform input format)
  - → iterative improvement (matching)

complexities

time
· amount used

space
amount of memory units used

we actually talk about order of growth ← time complexity / efficiency

count number of primitive operations in the algorithm

→ declaration
→ assignment
→ arithmetic operations
→ logic operations

these take a constant time to perform, unrelated to problem size or input value

i) repetition structure : for/while
ii) selection structure : if/else statement. switch-case
iii) recursive

→ best case c_1
→ worst case c_2
→ avg case $p(x<a)c_1 + p(x \geq a)c_2$

$\text{prob}_{s1} \times t_{s1} + \text{prob}_{s2} \times t_{s2} + \text{prob}_{s3} \times t_{s3} \ldots$

· primitive op in each call
· number of calls.

si → situation i
t_{si} → time taken for si

important results :

- sum of geometric : $\dfrac{a(1-r^n)}{1-r}$
- — " — arithmetic : $\dfrac{n}{2}[a+l] = \dfrac{n}{2}[2a+(n-1)d]$

arithmetico-geometric : $\sum_{t=1}^{k} t\,2^{t-1} = 2^k(k-1)+1$

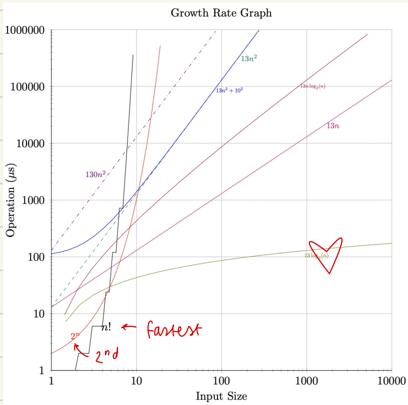
- $\sum k^2 = \dfrac{k(k+1)(2k+1)}{6}$

$\sum k = \dfrac{k(k+1)}{2}$

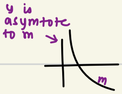
$\sum k^3 = \left(\dfrac{k(k+1)}{2}\right)^2$

order of growth

Algorithm	linear	linearithmic	quadratic 1	quadratic 2	quadratic 3	exponential
Input / Operation(s)	$13n$	$13n\log_2 n$	$13n^2$	$130n^2$	$13n^2+10^2$	2^n
10	0.00013	0.00043	0.0013	0.013	0.0014	0.001024
100	0.0013	0.0086	0.13	1.3	0.1301	4×10^{16} years
10^4	0.13	1.73	22mins	3.61hrs	22mins	
10^6	13	259	150days	1505days	150days	



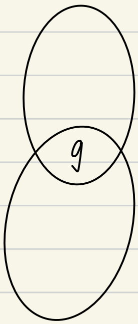
straight line that approaches a given curve continually but does not meet it at any finite distance



Asymptotic Notation

Big-Oh (O), Big-Omega (Ω) & Big-Theta (Θ)

notations used to describe order of growth of a f^n



$f \in \Omega(g)$ set of f^n that grow at higher or same rate as g

$f \in \Theta(g)$ set of f^n that grow at same rate as g

$f \in O(g)$: set of f^n that grow at lower of same rate as g

if $g(n) = n^2$

$f \cdot f(n) = 130n^2$, $11n^2+7n+5$, any quad f^n of n

$\in \Theta(g)$

Big-Oh Notation

$f, g : N \to R^+$; $f(n) \in O(g(n))$ if f is bounded above with some constant multiple of $g(n)$ $\forall$ large n

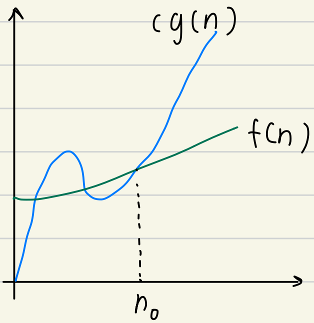
$O(g(n)) = \{f(n) : \exists$ positive constants, c & $n_0 : 0 \leq f(n) \leq cg(n)$ $\forall n \geq n_0\}$

reflexive, transitive

Big-Oh Notation (O)

$f, g : N \to R^+$; $f(n) \in O(g(n))$ if f is bounded above with some constant multiple of $g(n)$ $\forall$ large n

$O(g(n)) = \{f(n) : \exists$ positive constants, c & $n_0 : 0 \leq f(n) \leq cg(n)$ $\forall n \geq n_0\}$



ex 1 $\quad f(n) = 4n+3 \quad g(n) = n$

if $c = 5$, $n_0 = 3$

$f(n) \leq cg(n)$ $\forall n \geq n_0$

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

ex 2 $\quad f(n) = 4n+3 \quad g(n) = n^3$

$n_0 = 3$, $c = 1$

$4n+3 \leq n^3$ $\forall n > 3$

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

$4n+3 \in O(n^3)$

for same def, if $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$, then $f(n) \in O(g(n))$

ex 1 $\quad f(n) = 4n+3 \quad g(n) = n$

$$\frac{4n+3}{n} = 4 + \frac{3}{n}$$

as $n \to \infty$, $\frac{f(n)}{g(n)} \to 4 < \infty$ $\quad \therefore f(n) \in O(g(n))$

ex 2 $f(n) = 4n+3 \quad g(n) = n^3$

$\lim_{n\to\infty} \frac{4n+3}{n^3} = 0 < \infty \quad \therefore f(n) = O(g(n))$

ex 3 $f(n) = 4n+3 \quad g(n) = e^n$

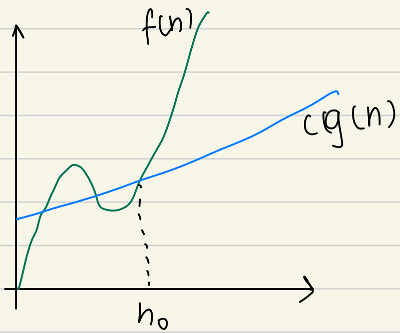
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{4n+3}{e^n} = \lim_{n\to\infty} \frac{4}{ne^n} = 0 < \infty \quad \therefore 4n+3 \in O(e^n)$

reflexive, transitive

Big-Omega Notation (Ω)

$f(n), g(n) : N \to R^+$, $f(n) \in \Omega(g(n))$, if f is bounded below by some constant multiple of $g(n)$ for all large n, ie

$\Omega(g(n)) = \{f(n) : \exists$ positive constants, c and n_0 : $0 \leq c(g(n)) \leq f(n)$ $\forall n \geq n_0\}$



alternately

if $\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$, $f(n) \in \Omega(g(n))$

ex 1 $f(n) = 4n+3$, $g(n) = 5n$

$c = \frac{1}{5}$, $n_0 = 0$

$4n+3 > n \quad (\forall n \geq 0)$

ex 2 $f(n) = n^3 + 2n \quad g(n) = 5n$

$\lim_{n\to\infty} \frac{n^3+2n}{5n} = \lim_{n\to\infty} \frac{n^2+2}{5} = \infty > 0 \quad \therefore 4n+3 \in \Omega(5n)$

transitive, symmetric, reflexive

Big-Theta Notation Θ

$f(n), g(n) : N \to R^+$, bounded both above and below by some constant multiple of $g(n)$ $\forall$ large n

$\Theta(g(n)) = \{ f(n) : \exists$ positive constants, c_1, c_2 and n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ $\forall n \geq n_0 \}$

if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$ where $0 < c < \infty$ then $f(n) \in \Theta(g(n))$

ex1 $f(n) = 2n^2 + 7$ $g(n) = 7n^2 + n$

$\lim_{n \to \infty} \frac{2n^2+7}{7n^2+n} = \lim_{n \to \infty} \frac{4n}{14n} = \frac{2}{7}$ ✓

summary

$\lim_{n\to\infty} f(n)/g(n)$	$f(n) \in O(g(n))$	$f(n) \in \Omega(g(n))$	$f(n) \in \Theta(g(n))$
0	✓		
$0 < c < \infty$	✓	✓	✓
∞		✓	

* → $f(n) = O(g(n)) \Rightarrow g(n)$ is the asymtotic upper bound of $f(n)$
$f(n) = \Omega(g(n)) \Rightarrow$ lower
$f(n) = \Theta(g(n))$ tight

when time complexity of algo A grows faster than algo B's, we saw A is inferior to B steps

1. count no of primitive operations to derive complexity f^n $f(n)$
2. discard constants and multipliers
3. determine dominant term in $f(n)$
4. dom term = big-oh notation for $f(n)$

time complexity of sequential search

```
pt = head;                          → c1
while (pt→ key != a){
    pt= pt→next;                    } c2 (n-1) times
    if (pt== Null) break;
}
```

1. best case : c_1 (a is first) $\Rightarrow \Theta(1)$

2. worst case : $c_2(n-1) + c_1 \Rightarrow \Theta(n)$

3. avg case : all nodes have equal probability of being search key

$$\frac{1}{n}\left(c_1 + (c_1+c_2) + (c_1+2c_2) + (c_1+3c_2) + \dots (c_1+(n-1)c_2)\right)$$

$$\frac{1}{n}\left(nc_1 + c_2 + 2c_2 + \dots + (n-1)c_2\right)$$

in list → $c_1 + \frac{c_2(n-1)(n)}{2n} = c_1 + \frac{c_2}{2}(n-1) = \Theta(n)$

not in list : $c_1 + nc_2 = \Theta(n)$

$$T(n) = \underbrace{P(a \text{ in list})}_{\text{const}}\left(c_1 + \frac{c_2}{n}(n-1)\right) + \overbrace{(1 - P(a \text{ in list}))}^{\text{const}}(c_1 + nc_2)$$

$\therefore$ linear fn : $\Theta(n)$

asymptotic notation in equations

$2n^2 + 3n + 1 = 2n^2 + \Theta(n)$

$T(n) = T(n/2) = \Theta(n)$

$2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2)$

simplification rules

upperbound

1. if $f(n) = O(cg(n))$ for $c > 0$, $f(n) = O(g(n))$
2. if $f(n) = O(g(n))$, $g(n) = O(h(n))$, $f(n) = O(h(n))$
3. $f_1(n) = O(g_1(n))$, $f_2(n) = O(g_2(n))$
   $f_1(n) f_2(n) = O(g_1(n) g_2(n))$

common complexities

Order of Growth	**Class**	**Example**
1	constant	Finding midpoint of an array
$\log n$	logarithmic	Binary search
n	linear	Linear Search
$n \log_2 n$	linearithmic	Merge Sort
n^2	quadratic	Insertion Sort
n^3	cubic	Matrix Inversion (Gauss-Jordan elimination)
2^n	exponential	The Tower of Hanoi Problem
$n!$	factorial	Travelling Salesman Problem

Space Complexity

- number of entities in problem : problem size
- count no. of $\underbrace{\text{basic units}}$
  ↳ things that need constant amount of storage

- array of n int : $\overset{\text{space of 1 int}}{m}n \Rightarrow \Theta(n)$
- matrix m×n int : $cmn \Rightarrow \Theta(n^2)$