Disc Math - §10 - Analysis of algorithms

→ Idea: estimate the time efficiency in an algorithm we estimate the number of operations it performs.

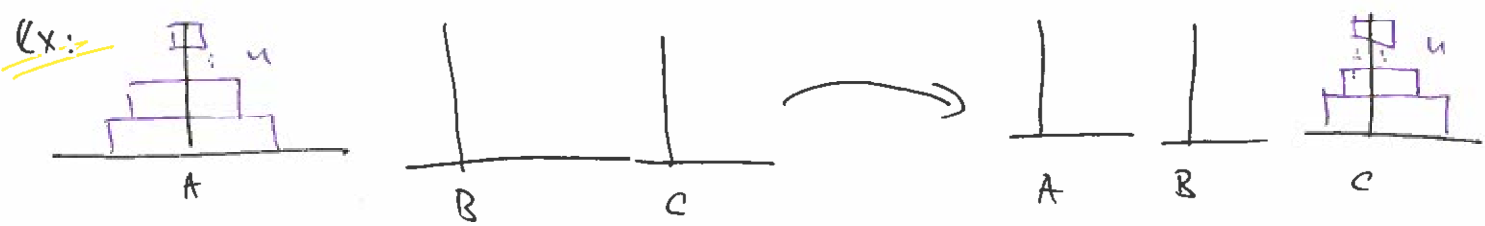
Ex: Consider an algorithm which finds max value in an array:

```
ARRAY_MAX (s)
    largest = s[1]
    i = 2
    while i ≤ s.last
        if s[i] > largest
        largest = s[i]
        i = i+1
    return largest.
```

Let the input array be of size n. A reasonable definition of the execution time is the number of iterations of the while loop. → $n-1$. (We usually do not count steps steps which are simply "housekeeping" work as $i=2$ or $i=i+1$ above.)

preview: Analysing a recursive algorithm requires solving an ~~equivalence~~ recurrence relation.

Ex:



$C(1) = 1$, $C(n) = C(n-1) + 1 + C(n-1) = 2C(n-1) + 1$.

$$C(n) = 2^{n-1}C(1) + \sum_{i=2}^{n} 2^{n-i}(1) = \cancel{2^{n-1}} + \sum_{i=1}^{n} 2^{n-i} - \cancel{2^{n-1}} =$$

$$= 2^n \frac{1/2 - 1/2^{n+1}}{1 - 1/2} = 2^n - 1$$ ← This is an optimal algorithm.

Ex: Binary search → find an item that is sorted in increasing order

Disc Math - § - Analysis of algorithms

search first the midpoint; if fail test for smaller or larger; say larger.

next search here etc.

Let $C(n)$ be the max number of comparisions needed to search a list of length n.

$$C(1) = 1 \quad , \quad C(n) = 1 + C\left(\frac{n}{2}\right) \, , \, n \geq 2 \, , \, n = 2^m$$

This is a divide-and-conquer recurrence relation; the solution is

$$C(n) = 1 + \log n.$$

Ex: Euclidean algorithm, find $\gcd(a,b) = ?$, $a > b$.

$E(a)$ - number of divisions to compute $\gcd(a,b)$ in the worst case

Sol: Observe that if $i > j$ and i is divided by j with remainder r, then $r < i/2$: There are two cases

1) $j \leq i/2$, then $r < j \leq i/2$
2) $j > i/2$, then $i = 1 * j + r$, $r = (i-j) < i/2$

Since in two steps the remainder becomes the dividend, successive dividends are at least halved every two steps. Thus

$$E(a) \leq 2\log(a)$$

since a can be halved at most $\log(a)$ times.

Ex. $a = 1024$, $E(a) \leq 20$. (rem: The upper bound is rather loose.)

Asymptotic bounds.

Ex: $t(n) = 60n^2 + 5n + 1$ Then $n = 100$, $t(100) = 600501$ But if $p(n) = 60n^2$ then $p(100) = 600000$. So the lower order terms contribute relatively little.

Ex: Let $t(n) = 60n^2 + 5n + 1$ be measured in seconds. Then

$$T(n) = n^2 + \frac{n}{12} + \frac{1}{60}$$ is measured in minutes.

rem: Thus if we measure the time needed to execute the algorithm on input of size n, we seek out the dominant term and ignore constant coefficients.

Ex: $t(n) = 60n^2 + 5n + 1$ is of order $n^2 \to t(n) = \Theta(n^2)$; $t(n)$ is big theta of n^2.

Def: Let f and g be nonnegative functions on $\mathbb{N}$. We write

$$\boxed{f(n) = O(g(n))}$$

and say that $f(n)$ is of order at most $g(n)$ if $\exists C_1 > 0$, $\exists N_1$ s.t.

$$f(n) \le C_1 g(n) \quad \forall n \ge N_1.$$

We write $$\boxed{f(n) = \Omega(g(n))}$$

and say that $f(n)$ is of order at least $g(n)$ if $\exists C_2 > 0$, $\exists N_2$ s.t.

$$f(n) \ge C_2(g) \quad \forall n \ge N_2.$$

We write $$\boxed{f(n) = \Theta(g(n))}$$

and say that $f(n)$ is of order $g(n)$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

rem: $f(n) = O(g(n))$ if except for a constant factor and a finite number of exceptions, f is bounded above by g. We also say that g is an asymptotic upper bound for f.

Similarly if $f(n) = \Omega(g(n))$, g is an asymptotic lower bound for f. Also if $f(n) = \Theta(g(n))$, f is bounded above and below by g. We also say that g is an asymptotic tight bound for f.

Disc Math - § - Analysis of Algorithms

rem: If $f = O(g)$ and $f \neq \Theta(g)$ then $f = o(g)$ ("small oh").

ex: $n = o(n^2)$, $\log n = o(n)$, $n = O(2^n)$ in fact $n = o(2^n)$.
$\log_a(n) = \Theta(\log_b n)$.

Th: Let $p(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$, $a_k \neq 0$. Then

$$p(n) = \Theta(n^k).$$

More ex:
i) $1 + 2 + \dots + n = \Theta(n^2)$
ii) $1^k + 2^k + \dots + n^k = \Theta(n^{k+1})$
iii) $\frac{1}{1} + \frac{1}{2} + \dots + \frac{1}{n} = \sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$.

rem: There is an arithmetic of equivalence classes $\Theta(f)$, e.g.
$\Theta(x) + \Theta(x^2) = \Theta(x^2)$, $\Theta(x^2) + \Theta(x^2) = \Theta(x^2)$ etc.

Def: If an algorithm requires $t(n)$ units of time to terminate in the worst case for an input of size n and

$$t(n) = O(g(n))$$

we say that the worst-case time required by the algorithm is $O(g(n))$. If $t(n) = \Theta(g(n))$ then we say that the worst-case time required by the algorithm is $\Theta(g(n))$ or that the computational complexity of the algorithm of order $\Theta(g(n))$.

→ best-case time
→ average-case time.

ex: Tower of Hanoi is $\Theta(2^n)$ (best, worst, average).

ex: Euclidean algorithm is $O(\log n)$. (worst-case).

ex: Gaussian elimination is $O(n^3)$ (worst-case).

ex: Say executing a step takes 0.0001 s.

Algor.	$n = 100$
$\Theta(n)$	0.01 s
$\Theta(n^2)$	1 s
$\Theta(2^n)$	4×10^{16} centuries

intractable

Disc Math - § - Analysis of algorithms

rem: The solution for a divide-and-conquer recurrence relation
$S(n) = cS(\frac{n}{2}) + g(n)$, $n \geq 2$, $n = 2^m$ is given by
$S(n) = c^{\log n} S(1) + \sum_{i=1}^{\log n} c^{(\log n)-i} g(2^i)$.

More generally we can consider splitting into b subproblems:
$S(n) = aS(\frac{n}{b}) + g(n)$, $n \geq 2$, $n = b^m$.

Th: Master Theorem. Consider the recurrence relation

$$S(1) \geq 0 \; ; \quad S(n) = aS\left(\frac{n}{b}\right) + n^c, \; n \geq 2$$

where $n = b^m$; $a, b \in \mathbb{Z}$, $a \geq 1$, $b > 1$; $c \in \mathbb{R}^+$. Then

1) $a < b^c$ $\quad S(n) = \Theta(n^c)$
2) $a = b^c$ $\quad S(n) = \Theta(n^c \log n)$
3) $a > b^c$ $\quad S(n) = \Theta(n^{\log_b a})$.

Pr: Book. ☒

Ex: $S(n) = 4S\left(\frac{n}{5}\right) + n^3$, $n \geq 2$, $n = 5^m$
$a = 4$, $b = 5$, $c = 3$; $4 < 5^3 \longrightarrow$ Case 1 $\longrightarrow S(n) = \Theta(n^3)$. ☒

Order of magnitude of Algorithms

$\Theta(1)$	$\Theta(\log\log n)$	$\Theta(\log n)$	$\Theta(n^c), 0<c<1$	$\Theta(n)$	$\Theta(n^2)$
constant	LogLog	Log	Sublinear	Linear	Quadratic
$\Theta(n^k), k \geq 1$	$\Theta(c^n)$	$\Theta(n!)$			
polynomial	exponential	factorial			

rem: A problem that has a worst-case polynomial-time algorithm is feasible (tractable); does not have PTime - intractable.

Problems could be unsolvable → Decision problem for predicate logic; Halting problem: given an arbitrary program and set of inputs, will the algorithm eventually halt?

def: (rough) A problem is NP if it can be solved in nondeterministic polynomial time.

Here are examples of NP (complete) problems: graph coloring, Hamiltonian cycle; travelling salesman problem.

rem: $P \subseteq NP \subseteq EXP$. In fact $NP \subset EXP$, i.e. $NP \neq EXP$.

Th: If any NP-complete problem is in P, then $NP = P$. ☒

Stephen Cook: "$P = NP$ would transform mathematics by allowing a computer to find a formal proof of any theorem which has a proof of reasonable length, since formal proofs can be easily recognized in polynomial time. Example problems may well include all of the CMI prize problems."

HW § 3.3 p. 212

7, 8, 13, 14.

HW § 5.5 p 421

5, 6, 9, 17, 19, 21, 25