Algorithm Design and Implementation

Principle of Algorithms II

Algorithm Analysis

Guoqiang Li

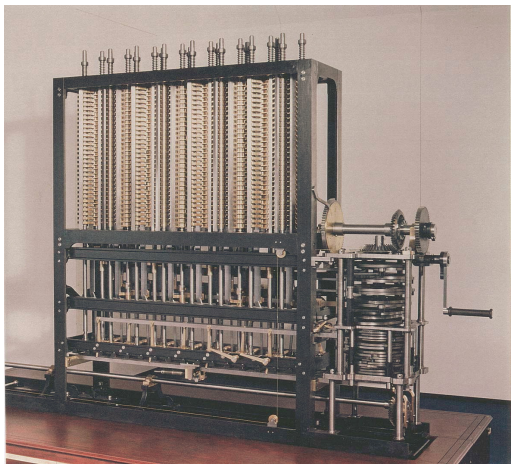
School of Software, Shanghai Jiao Tong University

Computational Tractability

A strikingly modern thought

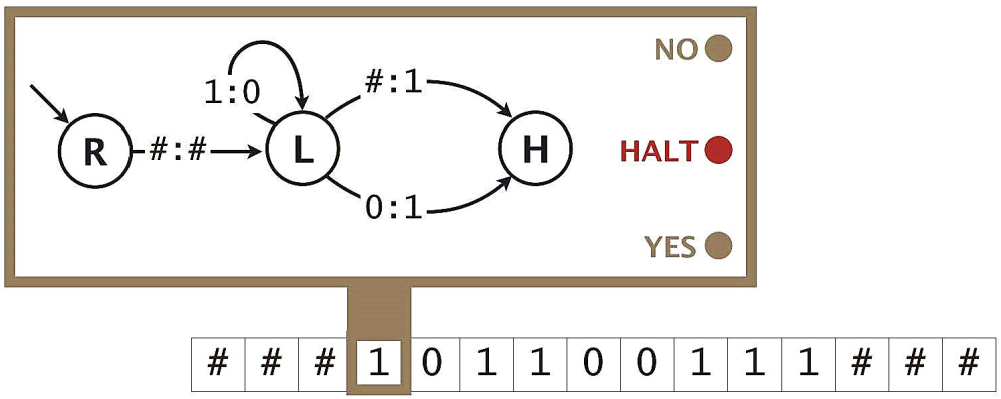
“ As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise—By what course of calculation can these results be arrived at by the machine in the shortest time? ”

- Charles Babbage (1864)



Models of computation: Turing machines

Deterministic Turing machine. Simple and idealistic model.



Running time. Number of steps.
Memory. Number of tape cells utilized.

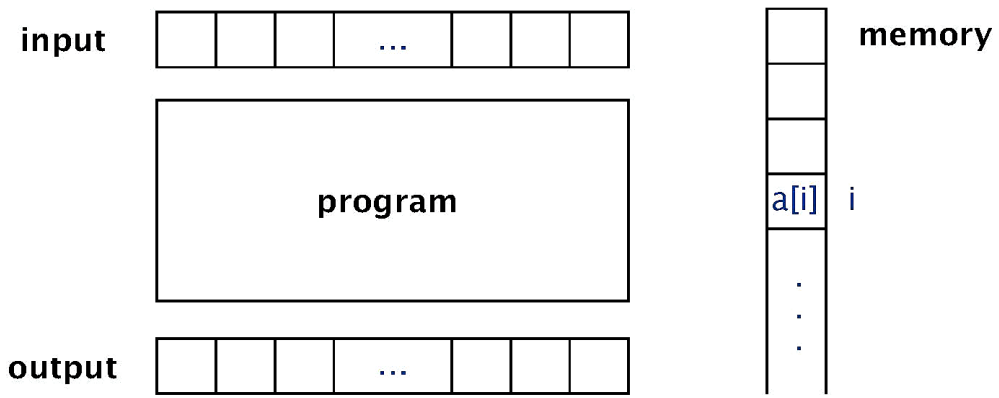
Remark. No random access of memory.

- Single tape TM requires $\geq n^2$ steps to detect n-bit palindromes.
- Easy to detect palindromes in $\leq cn$ steps on a real computer.

Models of computation: word RAM

Word RAM.

- Each memory location and input/output cell stores a w-bit integer.
- Primitive operations: arithmetic/logic operations, read/write memory, array indexing, following a pointer, conditional branch, . . .



Running time. Number of primitive operations.
Memory. Number of memory cells utilized.

Remark. At times, need more refined model (e.g., multiplying n-bit integers).

Brute force

For many nontrivial problems, there is a natural brute-force search algorithm that checks every possible solution.

- Typically takes 2^n steps (or worse) for inputs of size n.
- Unacceptable in practice.

Example

Stable matching problem: test all $n!$ perfect matchings for stability.

Polynomial running time

Desirable scaling property. When the input size doubles, the algorithm should slow down by at most some constant factor C.

Definition

An algorithm is poly-time if the above scaling property holds.

There exist constants $c > 0$ and $d > 0$ such that, for every input of size n, the running time of the algorithm is bounded above by $c\, n^d$ primitive computational steps.

Polynomial running time

We say that an algorithm is efficient if it has a polynomial running time.

Theory. Definition is (relatively) insensitive to model of computation.

Practice. It really works!

- The poly-time algorithms that people develop have both small constants and small exponents.
- Breaking through the exponential barrier of brute force typically exposes some crucial structure of the problem.

Exceptions. Some poly-time algorithms in the wild have galactic constants and/or huge exponents.

Q. Which would you prefer: $20n^{120}$ or $n^{1+0.02 \ln n}$?

Worst-case analysis

Worst case. Running time guarantee for any input of size n.

- Generally captures efficiency in practice.
- Draconian view, but hard to find effective alternative.

Exceptions. Some exponential-time algorithms are used widely in practice because the worst-case instances don't arise.

- simplex algorithm
- DPLL algorithm
- k-means algorithm

Other types of analyses

Probabilistic analysis Expected running time of a randomized algorithm.

Example

The expected number of compares to quicksort n elements is about $2n \ln n$.

Amortized analysis Worst-case running time for any sequence of n operations.

Example

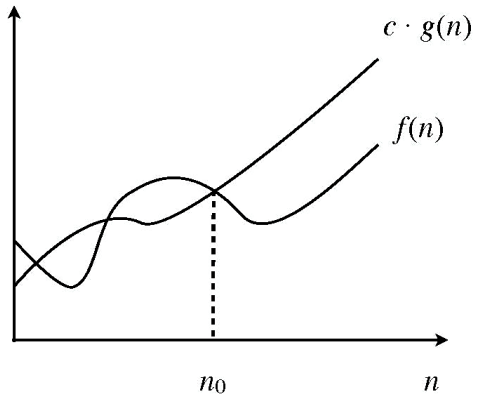
Starting from an empty stack, any sequence of n push and pop operations takes $O(n)$ primitive computational steps using a resizing array.

Also. Average-case analysis, smoothed analysis, competitive analysis, ...

Asymptotic Order of Growth

Big O notation

Upper bounds. $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.



Example

Let $f(n) = 32n^2 + 17n + 1$.

- $f(n)$ is $O(n^2)$.
- $f(n)$ is neither $O(n)$ nor $O(n \log n)$.

Typical usage. Insertion sort makes $O(n^2)$ compares to sort n elements.

Quiz

Let $f(n) = 3n^2 + 17n \log_2 n + 1000$. Which of the following are true?

A $f(n)$ is $O(n^2)$.

B $f(n)$ is $O(n^3)$.

C Both A and B.

D Neither A nor B.

Big O notational abuses

One-way "equality". $O(g(n))$ is a set of functions, but computer scientists often write $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$.

Example

Consider $g_1(n) = 5n^3$ and $g_2(n) = 3n^2$.

- We have $g_1(n) = O(n^3)$ and $g_2(n) = O(n^3)$.
- But, do not conclude $g_1(n) = g_2(n)$.

Domain and codomain. f and g and real-valued functions.

- The domain is typically the natural numbers: $\mathbb{N} \to \mathbb{R}$.
  - input size, recurrence relations, etc.
- Sometimes we extend to the reals: $\mathbb{R}_{\geq 0} \to \mathbb{R}$.
  - plotting, limits, calculus, etc.
- Or restrict to a subset.

Bottom line. OK to abuse notation in this way; not OK to misuse it.

Big O notation: properties

Reflexivity. f is $O(f)$.

Constants. If f is $O(g)$ and $c > 0$, then $c\,f$ is $O(g)$.

Products. If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 f_2$ is $O(g_1 g_2)$.

Proof.

- $\exists c_1 > 0$ and $n_1 \geq 0$ such that $0 \leq f_1(n) \leq c_1 \cdot g_1(n)$ for all $n \geq n_1$.
- $\exists c_2 > 0$ and $n_2 \geq 0$ such that $0 \leq f_2(n) \leq c_2 \cdot g_2(n)$ for all $n \geq n_2$.
- Then, $0 \leq f_1(n) \cdot f_2(n) \leq c_1 \cdot c_2 \cdot g_1(n) \cdot g_2(n)$ for all $n \geq \mathsf{max}\{n_1, n_2\}$.

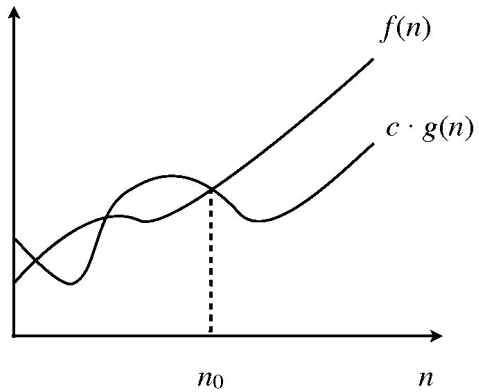
Sums. If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 + f_2$ is $O(\mathsf{max}\{g_1, g_2\})$.

Transitivity. If f is $O(g)$ and g is $O(h)$, then f is $O(h)$.

Ex. $f(n) = 5n^3 + 3n^2 + n + 1234$ is $O(n^3)$.

Big Ω notation

Lower bounds. $f(n)$ is $\Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $f(n) \geq c \cdot g(n) \geq 0$ for all $n \geq n_0$.



Example

Let $f(n) = 32n^2 + 17n + 1$.

- $f(n)$ is both $\Omega(n^2)$ and $\Omega(n)$.
- $f(n)$ is not $\Omega(n^3)$.

Typical usage. Any compare-based sorting algorithm requires $\Omega(n \log n)$ compares in the worst case.

Quiz

Which is an equivalent definition of big Omega notation?

A $f(n)$ is $\Omega(g(n))$ iff $g(n)$ is $O(f(n))$.

B $f(n)$ is $\Omega(g(n))$ iff there exist constants $c > 0$ such that

$$f(n) \geq c \cdot g(n) \geq 0$$

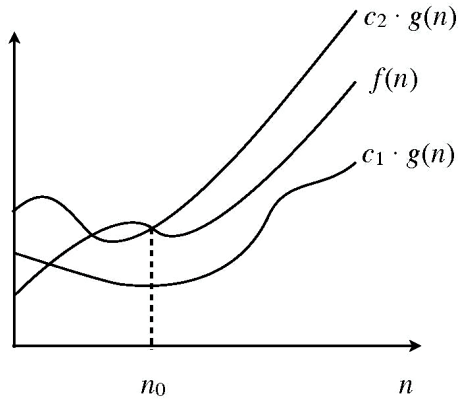
for infinitely many n.

C Both A and B.

D Neither A nor B.

Big Θ notation

Tight bounds. $f(n)$ is $\Theta(g(n))$ if there exist constants $c_1 > 0$, $c_2 > 0$, and $n_0 \geq 0$ such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$.



Example

Let $f(n) = 32n^2 + 17n + 1$.

- $f(n)$ is $\Theta(n^2)$.
- $f(n)$ is neither $\Theta(n^3)$ nor $\Omega(n)$.

Typical usage. Mergesort makes $\Theta(n \log n)$ compares to sort n elements.

(between $1/2 n \log_2 n$ and $n \log_2 n$)

Quiz

Which is an equivalent definition of big Theta notation?

A $f(n)$ is $\Theta(g(n))$ iff $f(n)$ is both $O(g(n))$ and $\Omega(g(n))$.

B $f(n)$ is $\Theta(g(n))$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$ for some constant $0 < c < +\infty$.

C Both A and B.

D Neither A nor B.

Asymptotic bounds and limits

Proposition

If $\lim_{n\to} \frac{f(n)}{g(n)} = c$ *for some constant* $0 < c < \infty$ *then* $f(n)$ *is* $\Theta(g(n))$.

Proof.

By definition of the limit, for any $\varepsilon > 0$, there exists n_0 such that

$$c - \varepsilon \leq \frac{f(n)}{g(n)} \leq c + \varepsilon$$

for all $n \geq n_0$.

Choose $\varepsilon = 1/2c > 0$.

Multiplying by $g(n)$ yields $1/2c \cdot g(n) \leq f(n) \leq 3/2c \cdot g(n)$ for all $n \geq n_0$.

Thus, $f(n)$ is $\Theta(g(n))$ by definition, with $c_1 = 1/2c$ and $c_2 = 3/2c$.

Asymptotic bounds and limits

Proposition

If $\lim_{n\to} \frac{f(n)}{g(n)} = 0$, *then* $f(n)$ *is* $O(g(n))$ *but not* $\Omega(g(n))$.

Proposition

If $\lim_{n\to} \frac{f(n)}{g(n)} = \infty$, *then* $f(n)$ *is* $\Omega(g(n))$ *but not* $O(g(n))$.

Asymptotic bounds for some common functions

Polynomials. Let $f(n) = a_0 + a_1 n + \ldots + a_d n^d$ with $a_d > 0$. Then, $f(n)$ is $\Theta(n^d)$.

$$\lim_{n \to \infty} \frac{a_0 + a_1 n + \ldots + a_d n^d}{n^d} = a_d > 0$$

Logarithms. $\log_a n$ is $\Theta(\log_b n)$ for every $a > 1$ and every $b > 1$.

$$\frac{\log_a n}{\log_b n} = \log_a b$$

Asymptotic bounds for some common functions

Logarithms and polynomials. $\log_a n$ is $O(n^d)$ for every $a > 1$ and every $d > 0$.

$$\lim_{n \to \infty} \frac{\log_a n}{n^d} = 0$$

Exponentials and polynomials. n^d is $O(r^n)$ for every $r > 1$ and every $d > 0$.

$$\lim_{n \to \infty} \frac{n^d}{r^n} = 0$$

Asymptotic bounds for some common functions

Factorials. $n!$ is $2^{\Theta(n \log n)}$.

Stirling's formula:

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Big O notation with multiple variables

Upper bounds. $f(m, n)$ is $O(g(m, n))$ if there exist constants $c > 0$, $m_0 \geq 0$, and$n_0 \geq 0$ such that $f(m, n) \leq c \cdot g(m, n)$ for all $n \geq n_0$ and $m \geq m_0$.

Example

$f(m, n) = 32mn^2 + 17mn + 32n^3$.

- $f(m, n)$ is both $O(mn^2 + n^3)$ and $O(mn^3)$.
- $f(m, n)$ is neither $O(n^3)$ nor $O(mn^2)$.

Typical usage. Breadth-first search takes $O(m + n)$ time to find a shortest path from s to t in a digraph with n nodes and m edges.

Implementing Gale–Shapley

Efficient implementation

Goal. Implement Gale–Shapley to run in $O(n^2)$ time.

```
Gale-Shapley(preference lists for hospitals and students)

Initialize M to empty matching;
while some hospital h is unmatched and hasn't proposed to every
 student do
    s ← first student on h's list to whom h has not yet proposed;
    if s is unmatched then
        Add h–s to matching M;
    end
    else
        if s prefers h to current partner h' then
            Replace h'–s with h–s in matching M;
        end
        else
            s rejects h;
        end
    end
end
Return stable matching M
```

Efficient implementation

Goal. Implement Gale–Shapley to run in $O(n^2)$ time.

Representing hospitals and students. Index hospitals and students $1, \ldots, n$.

Representing the matching.

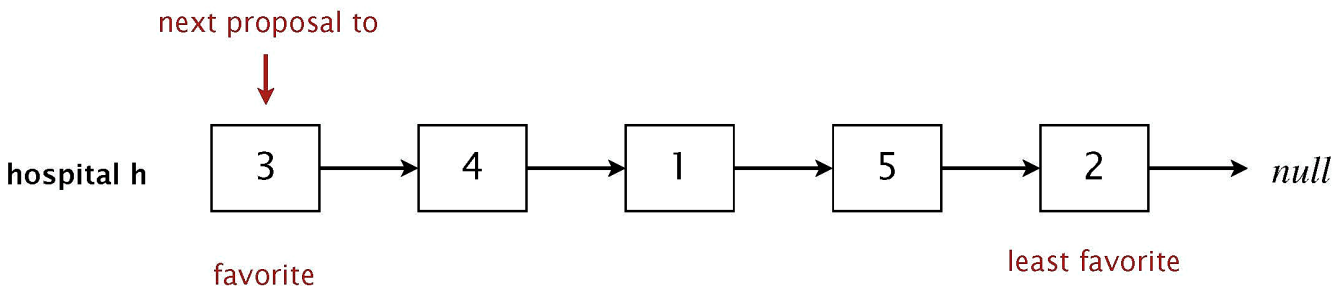
- Maintain two arrays $student[h]$ and $hospital[s]$.
  - if h matched to s, then $student[h] = s$ and $hospital[s] = h$
  - use value 0 to designate that hospital or student is unmatched
- Can add/remove a pair from matching in $O(1)$ time.

- Maintain set of unmatched hospitals in a queue (or stack).
- Can find an unmatched hospital in $O(1)$ time.

Data representation: making a proposal

Hospital makes a proposal.

- Key operation: find hospital's next favorite student.
- For each hospital: maintain a list of students, ordered by preference.
- For each hospital: maintain a pointer to student for next proposal.



Bottom line. Making a proposal takes $O(1)$ time.

Data representation: accepting/rejecting a proposal

Student accepts/rejects a proposal.

- Does student s prefer hospital h to hospital h'?
- For each student, create inverse of preference list of hospitals.

pref[]

1st	2nd	3rd	4th	5th	6th	7th	8th
8	3	7	1	4	5	6	2

rank[]

1	2	3	4	5	6	7	8
4th	8th	2nd	5th	6th	7th	3rd	1st

student prefers hospital 4 to 6 since $rank[4] < rank[6]$.

Bottom line. After $\Theta(n^2)$ preprocessing time (to create the n ranking arrays), it takes $O(1)$ time to accept/reject a proposal.

Stable matching: summary

Theorem

Can implement Gale–Shapley to run in $O(n^2)$ time.

Proof.

- $\Theta(n^2)$ preprocessing time to create the n ranking arrays.
- There are $O(n^2)$ proposals; processing each proposal takes $O(1)$ time.

Theorem

In the worst case, any algorithm to find a stable matching must query the hospital's preference list $\Omega(n^2)$ times.

Survey of Common Running Times

Constant time

Constant time. Running time is $O(1)$, bounded by a constant, which does not depend on input size n.

Examples.

Conditional branch.

Arithmetic/logic operation.

Declare/initialize a variable.

Follow a link in a linked list.

Access element i in an array.

Compare/exchange two elements in an array.

...

Linear time

Linear time. Running time is $O(n)$.

Merge two sorted lists. Combine two sorted linked lists $A = a_1, a_2, \ldots, a_n$ and $B = b_1, b_2, \ldots, b_n$ into a sorted whole.

Target sum. Given a sorted array of n distinct integers and an integer T, find two that sum to exactly T?

Logarithmic time

Logarithmic time. Running time is $O(\log n)$.

Binary search. Given a sorted array A of n distinct integers and an integer x, find index of x in array.

Search in sorted rotated array. Given a rotated sorted array of n distinct integers and an element x, determine if x is in the array.

Linearithmic time

Linearithmic time. Running time is $O(n \log n)$.

Sorting. Given an array of n elements, rearrange them in ascending order.

Largest empty interval. Given n timestamps $x_1, \ldots, x_n$ on which copies of a file arrive at a server, what is largest interval when no copies of file arrive?

Quadratic time

Quadratic time. Running time is $O(n^2)$.

Closest pair of points. Given a list of n points in the plane $(x_1, y_1), \ldots, (x_n, y_n)$ find the pair that is closest to each other.

Q: Can we do better?

Cubic time

Cubic time. Running time is $O(n^3)$.

3-Sum. Given an array of n distinct integers, find three that sum to 0.

$O(n^3)$ algorithm. Enumerate all triples (with $i < j < k$).

Q: How about an $O(n^2)$ algorithm?

Polynomial time

Polynomial time. Running time is $O(n^k)$ for some constant $k > 0$.

Independent set of size k. Given a graph, find k nodes such that no two are joined by an edge.

$O(n^k)$ algorithm. Enumerate all subsets of k nodes.

- Check whether S is an independent set of size k takes $O(k^2)$ time.
- Number of k-element subsets is

$$\binom{n}{k} \leq \frac{n^k}{k!}$$

- $O(k^2 n^k / k!) = O(n^k)$.

Exponential time

Exponential time. Running time is $O(2^{n^k})$ for some constant $k > 0$.

Independent set. Given a graph, find independent set of max cardinality.