Algorithms Design

Chap02-Basics of Algorithm Analysis

College of Computer Science

Nankai University

Tianjin, P.R.China

Outline

2.1 Computational Tractability

2.2 Asymptotic Order of Growth

2.3 Survey of Common Running Times

2.1 Computational Tractability

Brute force.

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

- Typically takes 2^N time or worse for inputs of size N.
- Unacceptable in practice.

E.g. Stable Matching Problem: test all n! perfect matchings for stability.

2.1 Computational Tractability

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

There exists constants $c > 0$ and $d > 0$ such that on every input of size N, its running time is bounded by cN^d steps.

Def.(2-1) An algorithm is poly-time if the above scaling property holds.

2.1 Computational Tractability

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

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.

2.1 Computational Tractability

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. E.g. Unit grep, simplex algorithm.

2.1 Computational Tractability

Other types of analyses

Probabilistic. Expected running time of a randomized algorithm.

- E.g. The expected number of compares to quicksort n elements is about $2nlog(n)$

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

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

2.1 Computational Tractability

Why does it make sense?

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	4 sec
$n = 30$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	18 min	10^{25} years
$n = 50$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	11 min	36 years	very long
$n = 100$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	$<$ 1 sec	$<$ 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	$<$ 1 sec	$<$ 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	$<$ 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

2.2 Asymptotic Order of Growth

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 $n \geq n_0$.

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

- $f(n)$ is $O(n^2)$. When $c = 50, n_0 = 1$
- $f(n)$ *is neither* $O(n)$ nor $O(n\log n)$.

Practical usage

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

Quiz 2-1

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.

2.2 Asymptotic Order of Growth

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

$f(n)$ is order of $g(n)$.

- f is asymptotically upper bounded by g.

E.g. $g_1(n) = 5n^3$ 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 $g_1(n) = g_2(n)$.

Note $O()$ expresses only an upper bound, not the exact growth rate of the function.

2.2 Asymptotic Order of Growth

Big O notation: properties

- Reflexivity: f is $O(f)$.
- Constants: If f is $O(g)$ and $c > 0$, then $c \cdot f$ is $O(g)$.
- Products: If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 \cdot f_2$ is $O(g_1 g_2)$.
- Sums: If f_1 is $O(g_1)$ and f2 is $O(g_2)$, then $f_1 + f_2$ is $O(\max\{g_1, g_2\})$.
- Transitivity. If f is $O(g)$ and g is $O(h)$ then f is $O(h)$.

Quiz 2-2

$\boldsymbol{f(n) = 5n^3 + 3n^2 + n + 1234.}$
Which one is correct?

A. $f(n) = O(n^3)$

B. $f(n) = O(n^2)$

C. $f(n) = O(n)$

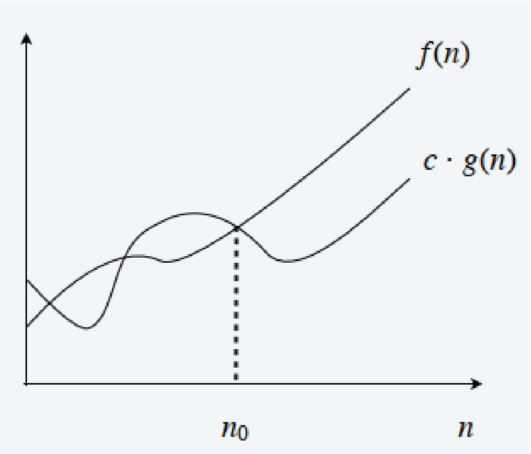
D. $f(n) = O(1)$

2.2 Asymptotic Order of Growth

Lower bounds. $f(n)$ is $\Omega(g(n))$ if there exist constants c > o and $n_0 \geq 0$

such that $f(n) \geq c \cdot g(n) \geq 0$ for all $n \geq n_0$.

- f is asymptotically lower bounded by g.



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

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

When $c = 32, n_0 = 1$

Practical Usage

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

Quiz 2-3

Which is an equivalent definition of asymptotically lower bounds?

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

B. $f(n)$ is $\Omega(g(n))$ iff there exists a constant $c > 0$ 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.

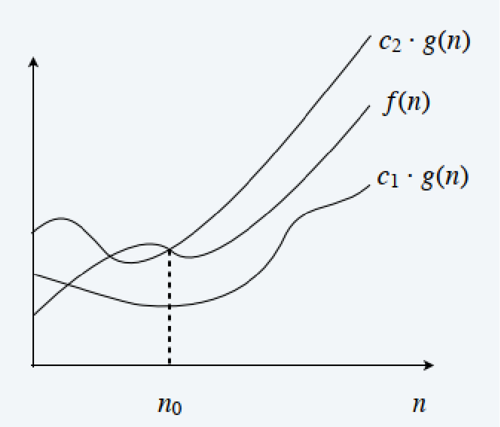
2.2 Asymptotic Order of Growth

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 $n \geq n_0$.

- $g(n)$ is asymptotically tight bound for $f(n)$.

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

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



Practical Usage

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

Quiz 2-3

Which is an equivalent definition of tight bound?

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 $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = c$ for some constant $0 < c < \infty$.

C. Both A and B.

D. Neither A nor B.

2.2 Asymptotic Order of Growth

Asymptotic bounds and limits

- If $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = c$ for some constant $0 < c < \infty$, then $f(n)$ is $\Theta(g(n))$.
- If $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = 0$,then $f(n)$ is $\mathrm{O}(g(n))$.
- If $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \infty$,then $f(n)$ is $\Omega(g(n))$.

2.2 Asymptotic Order of Growth

Transitivity

- If $f = O(g)$ and $g = O(h)$ then $f = O(h)$.
- If $f = \Omega(g)$ and $g = \Omega(h)$ then $f = \Omega(h)$.
- If $f = \Theta(g)$ and $g = \Theta(h)$ then $f = \Theta(h)$.

Additivity

- If $f = O(g)$ and $g = O(h)$ then $f + g = O(h)$.
- If $f = \Omega(g)$ and $g = \Omega(h)$ then $f + g = \Omega(h)$.
- If $f = \Theta(g)$ and $g = \Theta(h)$ then $f + g = \Theta(h)$.

2.2 Asymptotic Order of Growth

Asymptotic bounds for some common functions

Polynomials

- Let $f(n) = a_0 + a_1 n + \cdots + a_d n^d$ with $a_d > 0$. Then, $f(n)$ is $O(n^d)$.

Logarithms

- $log_a n$ is $\Theta(log_b n)$ for every $a > 1$ and every $b > 1$.
- $log_a n$ is $O(n^x)$ for every $a > 1$ and every $x > 0$.

2.2 Asymptotic Order of Growth

Exponentials

- r^n is $\Omega(n^d)$ for every $r > 1$ and every $d > 0$.

Factorials

- $n!$ is $2^{\Theta(nlogn)}$.
- Pf. Stirling's formula: $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

2.2 Asymptotic Order of Growth

Upper bounds with multiple variables

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

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

- $f(m, n)$ is both $O(mn^2 + n^3)$ and $O(mn^3)$.
- $f(m, n)$ is $O(n^3)$ if a precondition to the problem implies $m \leq n$.
- $f(m, n)$ is neither $O(n^3)$ nor $O(mn^2)$.

Practical usage

- In the worst case, 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.

2.3 Survey of Common Running Times

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.
- …

2.3 Survey of Common Running Times

Logarithmic time.

- Running time is O(log n).

Search in a sorted array.

- Given a sorted array A of n distinct integers and an integer x, find index of x in array.
- $O(\log n)$ algorithm
  - Binary search

After k iterations of WHILE loop,

$(hi - lo + 1) \leq n / 2^k \Rightarrow k \leq 1 + \log_2 n$

$lo \leftarrow 1;\ hi \leftarrow n.$

WHILE $(lo \leq hi)$

$mid \leftarrow \lfloor (lo + hi) / 2 \rfloor.$

IF $(x < A[mid])$ $hi \leftarrow mid - 1.$

ELSE IF $(x > A[mid])$ $lo \leftarrow mid + 1.$

ELSE RETURN $mid.$

RETURN $-1.$

2.3 Survey of Common Running Times

Linear time.

- Running time is O(n).

Merge two sorted lists.

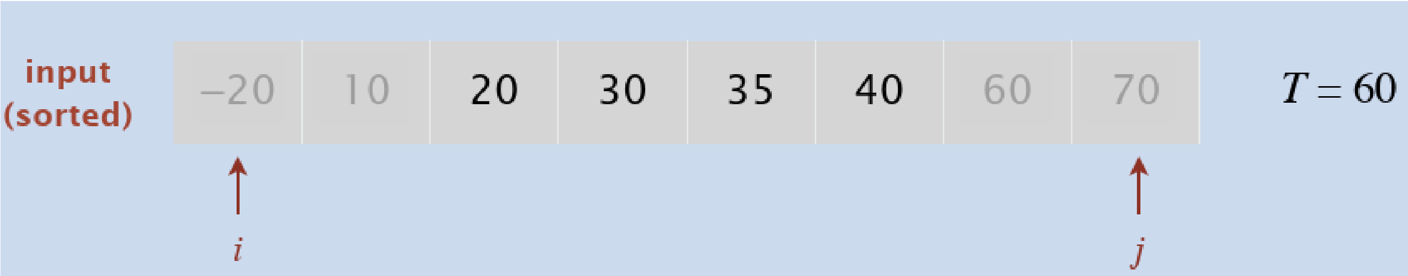
- Combine two sorted linked lists $A = [a_1, a_2, \dots, a_n]$ and $B = [b_1, b_2, \cdots, b_n]$ into a sorted list.

```
i ← 1; j ← 1.
WHILE (both lists are nonempty)
   IF (a_i ≤ b_j) append a_i to output list and increment i.
   ELSE           append b_j to output list and increment j.
Append remaining elements from nonempty list to output list.
```

2.3 Survey of Common Running Times

TARGET-SUM

- Given a sorted array of n distinct integers and an integer T, find two that sum to exactly T ?
- $O(n^2)$ algorithm.
    - Try all pairs.
- $O(n)$ algorithm.
    - Exploit sorted order.



2.3 Survey of Common Running Times

Linearithmic time.

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

Sorting

- Given an array of n elements, rearrange them in ascending order.
- $O(n \log n)$ algorithm
  - Mergesort.

Largest-Empty-Interval

- Given n timestamps $x_1, \cdots, x^n$ on which copies of a file arrive at a server, what is largest interval when no copies of file arrive?
- O(n log n) algorithm

2.3 Survey of Common Running Times

Quadratic time.

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

Closest pair of points.

- Given a list of n points in the plane $\{(x_1, y_1), \cdots, (x_n, y_n)\}$, find the pair that is closest to each other.
- $O(n^2)$ algorithm

```
min ← ∞.
FOR i = 1 TO n
    FOR j = i + 1 TO n
        d ← (x_i – x_j)^2 + (y_i – y_j)^2.
        IF (d < min)
            min ← d.
```

2.3 Survey of Common Running Times

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
- $O(n^2)$ algorithm

FOR $i = 1$ TO n
  FOR $j = i + 1$ TO n
    FOR $k = j + 1$ TO n
    IF $(a_i + a_j + a_k = 0)$
      RETURN (a_i, a_j, a_k).

2.3 Survey of Common Running Times

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

$O(n^3)$ algorithm. Try all triples.

$O(n^2)$ algorithm.

- Sort the array a.
- For each integer a_i: solve TARGET-SUM on the array containing all elements except a_i with the target sum $T = -a_i$.

2.3 Survey of Common Running Times

Set Disjointness. Given n sets $S_1, \cdots, S_n$ each of which is a subset of $\{1,2,\cdots,n\}$, is there some pair of these which are disjoint?

$O(n^3)$ solution. For each pairs of sets, determine if they are disjoint.

```
foreach set Si {
   foreach other set Sj {
      foreach element p of Si {
         determine whether p also belongs to Sj
      }
      if (no element of Si belongs to Sj)
         report that Si and Sj are disjoint
   }
}
```

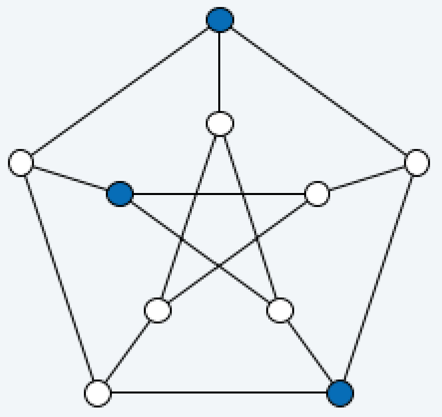
2.3 Survey of Common Running Times

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.



independent set of size 3

2.3 Survey of Common Running Times

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

FOREACH subset S of k nodes:
  Check whether S is an independent set.
  IF (S is an independent set)
    RETURN S.

- Check whether S is an independent set of size k takes $O(k^2)$ time.
- \# k-element subsets $= C_n^k \leq \dfrac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots 1} \leq \dfrac{n^k}{k!}$
- $O\left(\dfrac{k^2 n^k}{k!}\right) = O(n^k)$

2.3 Survey of Common Running Times

Exponential Time

Independent set. Given a graph, what is maximum size of an independent set?

$O(n^2 2^n)$ solution. Enumerate all subsets.

```
S* ← ϕ
foreach subset S of nodes {
   check whether S in an independent set
   if (S is largest independent set seen so far)
      update S* ← S
   }
}
```