

Brute force

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.



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

5

Polynomial running time

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

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

There exist constants $a > 0$ and $b > 0$ such that, for every input of size n , the algorithm performs $\leq a n^b$ primitive computational steps.

← corresponds to $C = 2^b$



Cobham
(1964)



Edmonds
(1965)

6

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 $n^{1+0.02 \ln n}$?



7

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.

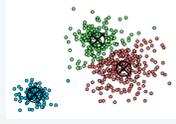
Exceptions. Some exponential-time algorithms are used widely in practice because the worst-case instances don't arise.



simplex algorithm

```
alexio@darwin ~$ cat testo
dark-net.com
dark-linux.net
dark-admin.net
redlinux.net
networking
netrg
pluto
alexio@darwin ~$ cat testo | grep net
dark-net.com
dark-linux.net
dark-admin.net
redlinux.net
networking
netrg
alexio@darwin ~$
```

Linux grep



k-means algorithm

Other types of analyses

Probabilistic. Expected running time of a randomized algorithm.

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



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

Ex. 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, ...

CSCI 355: ALGORITHM DESIGN AND ANALYSIS 2. ALGORITHM ANALYSIS

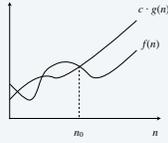
- ▶ computational tractability
- ▶ asymptotic order of growth
- ▶ survey of common running times

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 $n \geq n_0$.

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

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



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

11

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 $f(n) \in O(g(n))$.

Ex. 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 are real-valued functions.

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

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

13

Big O notation: properties

Reflexivity. f is $O(f)$.

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

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

Pf.

- $\exists c_1 > 0$ 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, $0 \leq f_1(n) \cdot f_2(n) \leq \frac{c_1 \cdot c_2}{c} \cdot g_1(n) \cdot g_2(n)$ for all $n \geq \max\{n_1, n_2\}$. ▀

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

← ignore lower-order terms

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 $O(n^3)$.

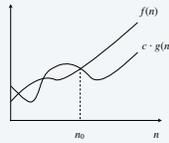
14

Big Omega 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 $n \geq n_0$.

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

- $f(n)$ is both $\Omega(n^2)$ and $\Omega(n)$. ← choose $c = 32, n_0 = 1$
- $f(n)$ is not $\Omega(n^3)$.



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

Vacuous statement. Any compare-based sorting algorithm requires at least $O(n \log n)$ compares in the worst case.

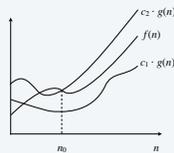
15

Big Theta 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 $n \geq n_0$.

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

- $f(n)$ is $\Theta(n^2)$. ← choose $c_1 = 32, c_2 = 50, n_0 = 1$
- $f(n)$ is neither $\Theta(n)$ nor $\Theta(n^3)$.



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

↗
between $\frac{1}{2} n \log_2 n$
and $n \log_2 n$

18

Asymptotic bounds and limits

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

Pf.

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

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

for all $n \geq n_0$.

- Choose $\epsilon = \frac{1}{2} c > 0$.
- Multiplying by $g(n)$ yields $1/2 c \cdot g(n) \leq f(n) \leq 3/2 c \cdot g(n)$ for all $n \geq n_0$.
- Thus, $f(n)$ is $\Theta(g(n))$ by definition, with $c_1 = 1/2 c$ and $c_2 = 3/2 c$. ■

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

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

20

Asymptotic bounds for some common functions

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

Pf.
$$\lim_{n \rightarrow \infty} \frac{a_0 + a_1 n + \dots + a_d n^d}{n^d} = a_d > 0$$

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

Pf.
$$\frac{\log_a n}{\log_b n} = \frac{1}{\log_b a}$$

no need to specify base (assuming it is a constant)

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

Pf.
$$\lim_{n \rightarrow \infty} \frac{\log_a n}{n^d} = 0$$

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

Pf.
$$\lim_{n \rightarrow \infty} \frac{n^d}{r^n} = 0$$

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

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

21

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 $0 \leq f(m, n) \leq c \cdot g(m, n)$ for all $n \geq n_0$ and $m \geq m_0$.

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

- $f(m, n)$ 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 $O(mn^2)$.

Typical 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.

22

CSCI 355: ALGORITHM DESIGN AND ANALYSIS 2. ALGORITHM ANALYSIS

- ▶ computational tractability
- ▶ asymptotic order of growth
- ▶ survey of common running times

Constant time

Constant time. Running time is $O(1)$.

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.
- ...

↙ bounded by a constant,
which does not depend on input size n

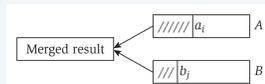
24

Linear time

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 $B = b_1, b_2, \dots, b_n$ into a sorted whole.

$O(n)$ algorithm. Merge in mergesort.



$i \leftarrow 1; j \leftarrow 1.$

WHILE (both lists are nonempty)

IF ($a_i \leq b_j$) 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.

25

Logarithmic time

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.

- Invariant: If x is in the array, then x is in $A[lo \dots hi]$.
- 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 $-1.$

26

Linearithmic time

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.

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
M E R G E S O R T E X A M P L E
E M R G E S O R T E X A M P L E
E M G R E S O R T E X A M P L E
E G M R E S O R T E X A M P L E
E G M R E S O R T E X A M P L E
E G M R E S O R T E X A M P L E
E G M R E S O R T E X A M P L E
E G M R E O R S T E X A M P L E
E E G M O R R S T E X A M P L E
E E G M O R R S E T X A M P L E
E E G M O R R S E T A X M P L E
E E G M O R R S A E T X M P L E
E E G M O R R S A E T X M P L E
E E G M O R R S A E T X M P L E
E E G M O R R S A E T X E L M P
E E G M O R R S A E E L M P T X
A E E E E G L M M O P R R S T X
```

27

Quadratic time

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

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

$O(n^2)$ algorithm. Enumerate all pairs of points (with $i < j$).

```
min ← ∞.
FOR i = 1 TO n
  FOR j = i + 1 TO n
    d ← (xi - xj)2 + (yi - yj)2.
    IF (d < min)
      min ← d.
```

Remark. $\Omega(n^2)$ seems inevitable, but this is just an illusion. [see §5.4]

28

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 $i < j < k$).

```
FOR i = 1 TO n
  FOR j = i + 1 TO n
    FOR k = j + 1 TO n
      IF (ai + aj + ak = 0)
        RETURN (ai, aj, ak).
```

Remark. $\Omega(n^3)$ seems inevitable, but $O(n^2)$ is not hard.

29

Polynomial time

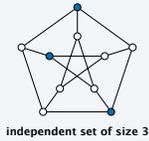
Polynomial time. Running time is $O(n^k)$ for some constant $k > 0$.

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

k is a constant

$O(n^k)$ 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.
- Number of k -element subsets = $\binom{n}{k} = \frac{n(n-1)(n-2) \times \dots \times (n-k+1)}{k(k-1)(k-2) \times \dots \times 1} \leq \frac{n^k}{k!}$
- $O(k^2 n^k / k!) = O(n^k)$.

poly-time for $k=17$, but not practical

30

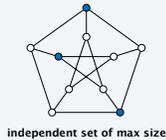
Exponential time

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

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

$O(n^2 2^n)$ algorithm. Enumerate all subsets of n elements.

```
 $S^* \leftarrow \emptyset$ .  
FOREACH subset  $S$  of  $n$  nodes:  
  Check whether  $S$  is an independent set.  
  IF ( $S$  is an independent set and  $|S| > |S^*|$ )  
     $S^* \leftarrow S$ .  
RETURN  $S^*$ .
```



31

Exponential time

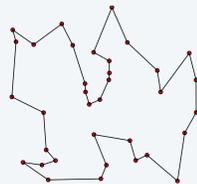
Exponential time. Running time is $O(2^{nk})$ for some constant $k > 0$.

Euclidean TSP. Given n points in the plane, find a tour of minimum length.

$O(n \times n!)$ algorithm. Enumerate all permutations of length n .

```
 $\pi^* \leftarrow \emptyset$ .  
FOREACH permutation  $\pi$  of  $n$  points:  
  Compute length of tour corresponding to  $\pi$ .  
  IF ( $\text{length}(\pi) < \text{length}(\pi^*)$ )  
     $\pi^* \leftarrow \pi$ .  
RETURN  $\pi^*$ .
```

for simplicity, we'll assume Euclidean distances are rounded to nearest integer (to avoid issues with infinite precision)



32