Objectives

Algorithm analysis

Data structures

Review

How do we define efficiency in algorithms?

- In what "case" do we analyze algorithms?

What term do we care about in algorithms? Why?

What is the definition of "big O"?

What is the symbol for the *asymptotic lower bound*?

What is the symbol for the *asymptotic tight bound*?

Review: Worst-Case Polynomial-Time

Def. An algorithm is **efficient** if its running time is *polynomial*

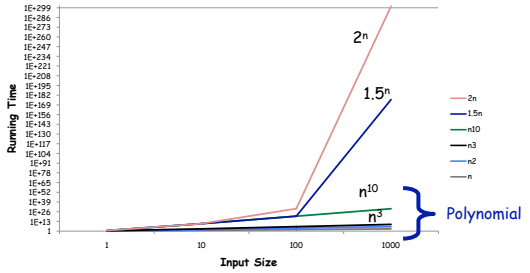
Justification: It really works in practice!

- In practice, poly-time algorithms that people develop almost always have low constants and low exponents
- Although $6.02 \times 10^{23} \times N^{20}$ is technically poly-time, it would be useless in practice

Exceptions.

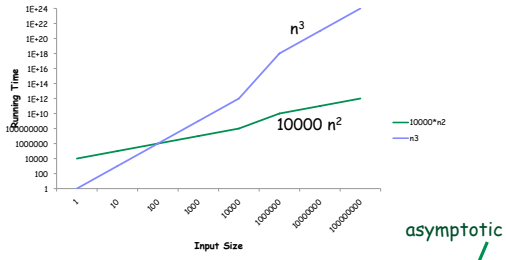
- Some poly-time algorithms do have high constants and/or exponents, and are useless in practice
- Some exponential-time (or worse) algorithms are widely used because the worst-case instances seem to be rare

Review: Running Times



- Huge difference from polynomial to not polynomial
- Differences in runtime matter more as input size increases

Review: Running Times



asymptotic

As input size increases, n^3 dominates large constant * n^2
➔Care about running time as input size *approaches infinity*
➔Only care about highest-order term

Review: Asymptotic Order of Growth Upper Bounds

T(n) is the worst case running time of an algorithm

We say that T(n) is **O(f(n))**

- "order f(n)"

if there exist constants c > 0 and $n_0 \geq 0$ such that for all $n \geq n_0$

- i.e., sufficiently large n, c cannot depend on n

we have **$T(n) \leq c \cdot f(n)$**

- i.e., T(n) is bounded above by a constant multiple of f(n)

➢T is ***asymptotically upperbounded*** by f

Example: Upper Bound

$T(n) = pn^2 + qn + r$

- p, q, r are positive constants

For all $n \geq 1$,

$T(n) = pn^2 + qn + r \leq pn^2 + qn^2 + rn^2 = (p+q+r)\, n^2$

➔$T(n) \leq cn^2$, where $c = p+q+r$

➔$T(n) = O(n^2)$

Also correct to say that $T(n) = O(n^3)$

Review: Asymptotic Order of Growth: Lower Bounds

Complementary to upper bound.

T(n) is **$\Omega(f(n))$**

if there exist constants $\varepsilon > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$

- i.e., sufficiently large n, ε cannot depend on n

we have **$T(n) \geq \varepsilon \cdot f(n)$**

- i.e., T(n) is bounded below by a constant multiple of f(n)

➢T is ***asymptotically lowerbounded*** by f

Example: Lower Bound

$T(n) = pn^2 + qn + r$

- p, q, r are positive constants

Idea: Need to *deflate* terms rather than inflate

For all $n \geq 0$,

$T(n) = pn^2 + qn + r \geq pn^2$

➔$T(n) \geq cn^2$, where $\varepsilon = p$

➔$T(n) = \Omega(n^2)$

Also correct to say that $T(n) = \Omega(n)$

Asymptotic Order of Growth

Tight bounds. T(n) is **$\Theta(f(n))$** if T(n) is both O(f(n)) and Ω(f(n))

- The "right" bound

Properties

Transitivity

- If f = O(g) and g = O(h) then f = O(h)
- If f = Ω(g) and g = Ω(h) then f = Ω(h)
- If f = Θ(g) and g = Θ(h) then f = Θ(h)

Proofs in book

Additivity

- If f = O(h) and g = O(h) then f + g = O(h)
- If f = Ω(h) and g = Ω(h) then f + g = Ω(h)
- If f = Θ(h) and g = O(h) then f + g = Θ(h)

Transitivity

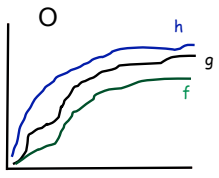
If g asymptotically upperbounds f
And h asymptotically upperbounds g
Then h assymptotically upperbounds h

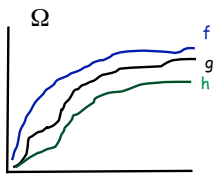
If f = O(g) and g = O(h) then f = O(h)

If f = Ω(g) and g = Ω(h) then f = Ω(h)

If f = Θ(g) and g = Θ(h) then f = Θ(h)

Proofs in book





Additivity

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

Sketch proof for O

Proofs in book

- $f \le c \cdot h$ (by defn of O)
- $g \le d \cdot h$
- $f + g \le c \cdot h + d \cdot h = (c + d)\, h = c' \cdot h$

ASYMPTOTIC BOUNDS FOR CLASSES OF ALGORITHMS

Asymptotic Bounds for Polynomials

$a_0 + a_1 n + \ldots + a_d n^d$ is $\Theta(n^d)$ if $a_d > 0$

- Asymptotic runtime determined by higher-order term

Other examples of polynomial times:

- $O(n^{1/2})$
- $O(n^{1.58})$
- $O(n \log n) \le O(n^2)$

Asymptotic Bounds for Logarithms

Logarithms. $\log_b n = x$, where $b^x = n$

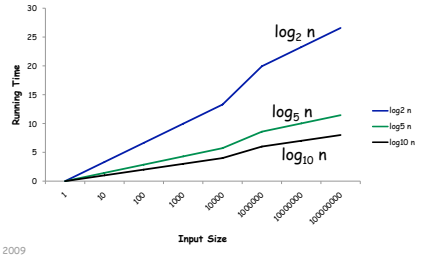
- x is number of digits to represent n in base-b representation

What does this mean for the running time of an algorithm that is O(log n)?

Asymptotic Bounds for Logarithms

Logarithms. $\log_b n = x$, where $b^x = n$

- x is number of digits to represent n in base-b representation



Asymptotic Bounds for Logarithms

Logarithms: $\log_b n = x$, where $b^x = n$

- x is number of digits to represent n in base-b representation

➔Slowly growing functions

$O(\log_a n) = O(\log_b n)$ for any constants a, b > 0

- Don't need to specify the base

For every $x > 0$, $\log n = O(n^x)$

- Log grows slower than every polynomial

Asymptotic Bounds for Exponentials

Exponentials: functions of the form $f(n) = r^n$ for constant base *r*

- Faster growth rates as *n* increases

For every $r > 1$ and every $d > 0$, $n^d = O(r^n)$

➢Every exponential grows faster than every polynomial

Summary of Asymptotic Bounds

In terms of growth rates ….

Summary of Asymptotic Bounds

In terms of growth rates ….

Logarithms < Polynomials < Exponentials

A SURVEY OF COMMON RUNNING TIMES

Linear Time: O(n)

Running time is at most a ***constant*** factor times the size of the input

- Example: process the input in one pass, doing constant amount of work
- Online algorithms
- Data stream algorithms

Linear Time: O(n)

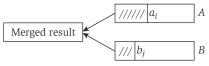
Computing the maximum: Compute maximum of n numbers $a_1, \ldots, a_n$

```
max = a1
for i = 2 to n
   if (ai > max)
      max = ai
```

Constant work for each input (does not depend on n)

Linear Time: O(n)

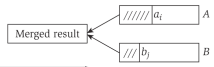
Example. Merge. Combine two sorted lists $A = a_1, a_2, \ldots, a_n$ with $B = b_1, b_2, \ldots, b_n$ into sorted whole



Linear Time: O(n)

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

Claim. Merging two lists of size n takes O(n) time



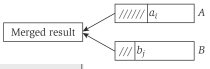

```
i = 1, j = 1
while (both lists are nonempty)
   if (a_i ≤ b_j)
        append a_i to output list and increment i
   else (a_i > b_j)
        append b_j to output list and increment j

append remainder of nonempty list to output list
```


Linear Time: O(n)

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

Claim. Merging two lists of size n takes O(n) time

Proof. After each comparison, the length of output list increases by 1




```
i = 1, j = 1
while (both lists are nonempty)
   if (a_i ≤ b_j)
        append a_i to output list and increment i
   else (a_i ≤ b_j)
        append b_j to output list and increment j

append remainder of nonempty list to output list
```


O(n log n) Time

Also referred to as linearithmic time

Arises in divide-and-conquer algorithms

- Splitting input into equal pieces, solve recursively, combine solutions in linear time

What well-known set of algorithms has an O(n logn) running time?

O(n log n) Time Example

Sorting. Mergesort and heapsort are sorting algorithms that perform O(n log n) comparisons

- Break input into equal-sized pieces
  - Running time of this step?
- Sorts each half recursively
- Merges sorted halves into a sorted list
  - Running time of this step?

O(n log n) Time Example

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

O(n log n) solution

- Sort time-stamps
- Scan sorted list in order, identifying the maximum gap between successive time-stamps

Quadratic Time: $O(n^2)$

Examples?

Quadratic Time: $O(n^2)$

Examples:

- Enumerate all pairs of elements
- Nested loops (*n* iterations)

Quadratic Time: $O(n^2)$

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

$O(n^2)$ solution. Try all pairs of points

```
min = (x1 - x2)² + (y1 - y2)²
for i = 1 to n {
   for j = i+1 to n {
      d = (xi - xj)² + (yi - yj)²
      if (d < min)
         min = d
   }
}
```

don't need to take square roots

$\Omega(n^2)$ seems inevitable, but Chapter 5 has an $O(n \log n)$ solution

Cubic Time: $O(n^3)$

Examples?

Cubic Time: $O(n^3)$

Enumerate all triples of elements

Set disjointness. Given *n* sets $S_1, \ldots, S_n$ each of which is a subset of 1, 2, …, *n*, is there some pair of these which are disjoint?

Cubic Time: $O(n^3)$

Enumerate all triples of elements

Set disjointness. Given *n* sets $S_1, \ldots, S_n$ each of which is a subset of 1, 2, …, *n*, is there some pair of these which are disjoint?

$O(n^3)$ solution. For each pair 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
```

Polynomial Time: $O(n^k)$ Time

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

- *k* is a constant

Polynomial Time: $O(n^k)$ Time

If to get all pairs, the algorithm is $O(n^2)$, what is an example of an $O(n^k)$ algorithm?

Polynomial Time: $O(n^k)$ Time

If to get all pairs, the algorithm is $O(n^2)$, what is an example of an $O(n^k)$ algorithm?

- All subsets of size *k*

Polynomial Time: $O(n^k)$ Time

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

- *k* is a constant

Polynomial Time: $O(n^k)$ Time

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

- *k* is a constant

```
foreach subset S of k nodes
   check whether S in an independent set
   if (S is an independent set)
      report S is an independent set
```

$O(n^k)$ solution.

- Enumerate all subsets of k nodes
- Check whether S is an independent set = $O(k^2)$.
- Number of k element subsets = $\binom{n}{k} = \frac{n(n-1)(n-2)\cdots(n-k+1)}{k(k-1)(k-2)\cdots(2)(1)} \leq \frac{n^k}{k!}$
- $O(k^2 n^k / k!) = O(n^k)$.

poly-time for k=17, but not practical

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)
      S* = S
```

O(log n) Time

Sublinear time

Know any algorithms that take O(log n) time?

O(log n) Time

Example: Binary search

Often requires some pre-processing or data structure that allows cheaper "querying" than *n* time