Analysis of Algorithms

T. M. Murali

August 26, 2009
What is Algorithm Analysis?

- Measure resource requirements: how do the amount of time and space that an algorithm uses scale with increasing input size?
- How do we put this notion on a concrete footing?
- What does it mean for one function to grow faster or slower than another?
What is Algorithm Analysis?

- Measure resource requirements: how do the amount of time and space that an algorithm uses scale with increasing input size?
- How do we put this notion on a concrete footing?
- What does it mean for one function to grow faster or slower than another?
- Goal: Develop algorithms that provably run quickly and use low amounts of space.
Worst-case Running Time

- We will measure **worst-case** running time of an algorithm.
  - Avoid depending on test cases or sample runs.
- Bound the largest possible running time the algorithm over all inputs of size $n$, as a function of $n$. 
Worst-case Running Time

- We will measure worst-case running time of an algorithm.
  - Avoid depending on test cases or sample runs.
- Bound the largest possible running time the algorithm over all inputs of size $n$, as a function of $n$.
- Why worst-case? Why not average-case or on random inputs?
Worst-case Running Time

- We will measure worst-case running time of an algorithm.
  - Avoid depending on test cases or sample runs.
- Bound the largest possible running time the algorithm over all inputs of size $n$, as a function of $n$.
- Why worst-case? Why not average-case or on random inputs?
- *Input size* = number of elements in the input.
Worst-case Running Time

- We will measure worst-case running time of an algorithm.
  - Avoid depending on test cases or sample runs.
- Bound the largest possible running time the algorithm over all inputs of size $n$, as a function of $n$.
- Why worst-case? Why not average-case or on random inputs?
- Input size = number of elements in the input. Values in the input do not matter.
- Assume all elementary operations take unit time: assignment, arithmetic on a fixed-size number, comparisons, array lookup, following a pointer, etc.
  - Make analysis independent of hardware and software.
Polynomial Time

▶ Brute force algorithm: Check every possible solution.
Polynomial Time

- Brute force algorithm: Check every possible solution.
- What is a brute force algorithm for sorting: given $n$ numbers, permute them so that they appear in increasing order?
Polynomial Time

- Brute force algorithm: Check every possible solution.
- What is a brute force algorithm for sorting: given \( n \) numbers, permute them so that they appear in increasing order?
  - Try all possible \( n! \) permutations of the numbers.
  - For each permutation, check if it is sorted.
Polynomial Time

- Brute force algorithm: Check every possible solution.
- What is a brute force algorithm for sorting: given $n$ numbers, permute them so that they appear in increasing order?
  - Try all possible $n!$ permutations of the numbers.
  - For each permutation, check if it is sorted.
  - Running time is $nn!$. Unacceptable in practice!
Polynomial Time

- Brute force algorithm: Check every possible solution.
- What is a brute force algorithm for sorting: given $n$ numbers, permute them so that they appear in increasing order?
  - Try all possible $n!$ permutations of the numbers.
  - For each permutation, check if it is sorted.
  - Running time is $n!$. Unacceptable in practice!
- Desirable scaling property: when the input size doubles, the algorithm should only slow down by some constant factor $k$. 
Polynomial Time

- Brute force algorithm: Check every possible solution.
- What is a brute force algorithm for sorting: given $n$ numbers, permute them so that they appear in increasing order?
  - Try all possible $n!$ permutations of the numbers.
  - For each permutation, check if it is sorted.
  - Running time is $nn!$. Unacceptable in practice!

- Desirable scaling property: when the input size doubles, the algorithm should only slow down by some constant factor $k$.

- An algorithm has a *polynomial* running time if there exist constants $c > 0$ and $d > 0$ such that on every input of size $n$, the running time of the algorithm is bounded by $cn^d$ steps.
Polynomial Time

- Brute force algorithm: Check every possible solution.
- What is a brute force algorithm for sorting: given \( n \) numbers, permute them so that they appear in increasing order?
  - Try all possible \( n! \) permutations of the numbers.
  - For each permutation, check if it is sorted.
  - Running time is \( nn! \). Unacceptable in practice!
- Desirable scaling property: when the input size doubles, the algorithm should only slow down by some constant factor \( k \).

- An algorithm has a *polynomial* running time if there exist constants \( c > 0 \) and \( d > 0 \) such that on every input of size \( n \), the running time of the algorithm is bounded by \( cn^d \) steps.

**Definition**

An algorithm is *efficient* if it has a polynomial running time.
Upper and Lower Bounds

- Express “$4n^2 + 100$ does not grow faster than $n^2$.”
- Express “$n^2/4$ grows faster than $n + 1,000,000$.”
Upper and Lower Bounds

- Express “$4n^2 + 100$ does not grow faster than $n^2$.”
- Express “$n^2/4$ grows faster than $n + 1,000,000$.”

Definition

Asymptotic upper bound: A function $f(n)$ is $O(g(n))$ if we have $f(n) \leq g(n)$.
Upper and Lower Bounds

- Express “$4n^2 + 100$ does not grow faster than $n^2$.”
- Express “$n^2/4$ grows faster than $n + 1,000,000$.”

**Definition**

*Asymptotic upper bound*: A function $f(n)$ is $O(g(n))$ if there exists constant $c > 0$ such that we have $f(n) \leq cg(n)$. 
Upper and Lower Bounds

- Express “$4n^2 + 100$ does not grow faster than $n^2$.”
- Express “$n^2/4$ grows faster than $n + 1,000,000$.”

**Definition**

*Asymptotic upper bound:* A function $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $f(n) \leq cg(n)$.
Upper and Lower Bounds

- Express “$4n^2 + 100$ does not grow faster than $n^2$.”
- Express “$n^2/4$ grows faster than $n + 1,000,000$.”

**Definition**

*Asymptotic upper bound:* A function $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $f(n) \leq cg(n)$.

**Definition**

*Asymptotic lower bound:* A function $f(n)$ is $\Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $f(n) \geq cg(n)$. 
Upper and Lower Bounds

- Express "4n^2 + 100 does not grow faster than n^2."
- Express "n^2/4 grows faster than n + 1,000,000."

**Definition**

*Asymptotic upper bound:* A function $f(n)$ is $O(g(n))$ if there exist constant $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $f(n) \leq cg(n)$.

**Definition**

*Asymptotic lower bound:* A function $f(n)$ is $\Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $f(n) \geq cg(n)$.

**Definition**

*Asymptotic tight bound:* A function $f(n)$ is $\Theta(g(n))$ if $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$. 
Upper and Lower Bounds

- Express “$4n^2 + 100$ does not grow faster than $n^2$.”
- Express “$n^2/4$ grows faster than $n + 1,000,000$.”

**Definition**

*Asymptotic upper bound:* A function $f(n)$ is $O(g(n))$ if there exist constant $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $f(n) \leq cg(n)$.

**Definition**

*Asymptotic lower bound:* A function $f(n)$ is $\Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $f(n) \geq cg(n)$.

**Definition**

*Asymptotic tight bound:* A function $f(n)$ is $\Theta(g(n))$ if $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$.

- In these definitions, $c$ is a constant independent of $n$. 
## Upper and Lower Bounds

- Express "4n^2 + 100 does not grow faster than n^2."
- Express "n^2/4 grows faster than n + 1,000,000."

**Definition**

*Asymptotic upper bound*: A function $f(n)$ is $O(g(n))$ if there exist constant $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $f(n) \leq cg(n)$.

**Definition**

*Asymptotic lower bound*: A function $f(n)$ is $\Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $f(n) \geq cg(n)$.

**Definition**

*Asymptotic tight bound*: A function $f(n)$ is $\Theta(g(n))$ if $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$.

- In these definitions, $c$ is a constant independent of $n$.
- Abuse of notation: say $g(n) = O(f(n))$, $g(n) = \Omega(f(n))$, $g(n) = \Theta(f(n))$. 
Examples

- $f(n) = pn^2 + qn + r$ is
Examples

- $f(n) = pn^2 + qn + r$ is $\theta(n^2)$. Can ignore lower order terms.
Examples

- $f(n) = pn^2 + qn + r$ is $\theta(n^2)$. Can ignore lower order terms.
- Is $f(n) = pn^2 + qn + r = O(n^3)$?
Examples

- $f(n) = pn^2 + qn + r$ is $\theta(n^2)$. Can ignore lower order terms.
- Is $f(n) = pn^2 + qn + r = O(n^3)$?
- $f(n) = \sum_{0 \leq i \leq d} a_i n^i =$
Examples

- $f(n) = pn^2 + qn + r$ is $\theta(n^2)$. Can ignore lower order terms.
- Is $f(n) = pn^2 + qn + r = O(n^3)$?
- $f(n) = \sum_{0 \leq i \leq d} a_i n^i = O(n^d)$, if $d > 0$ is an integer constant and $a_d > 0$.
  - $O(n^d)$ is the definition of polynomial time.
Examples

- \( f(n) = pn^2 + qn + r \) is \( \theta(n^2) \). Can ignore lower order terms.
- Is \( f(n) = pn^2 + qn + r = O(n^3) \)?
- \( f(n) = \sum_{0 \leq i \leq d} a_i n^i = O(n^d) \), if \( d > 0 \) is an integer constant and \( a_d > 0 \).
  - \( O(n^d) \) is the definition of polynomial time.
- Is an algorithm with running time \( O(n^{1.59}) \) a polynomial-time algorithm?
Examples

- \( f(n) = pn^2 + qn + r \) is \( \theta(n^2) \). Can ignore lower order terms.
- Is \( f(n) = pn^2 + qn + r = O(n^3) \)?
- \( f(n) = \sum_{0 \leq i \leq d} a_i n^i = O(n^d) \), if \( d > 0 \) is an integer constant and \( a_d > 0 \).
  - \( O(n^d) \) is the definition of polynomial time.
- Is an algorithm with running time \( O(n^{1.59}) \) a polynomial-time algorithm?
- \( O(\log_a n) = O(\log_b n) \) for any pair of constants \( a, b > 1 \).
- For every \( x > 0 \), \( \log n = O(n^x) \).
Examples

- $f(n) = pn^2 + qn + r$ is $\theta(n^2)$. Can ignore lower order terms.
- Is $f(n) = pn^2 + qn + r = O(n^3)$?
- $f(n) = \sum_{0 \leq i \leq d} a_i n^i = O(n^d)$, if $d > 0$ is an integer constant and $a_d > 0$.
  - $O(n^d)$ is the definition of polynomial time.
- Is an algorithm with running time $O(n^{1.59})$ a polynomial-time algorithm?
- $O(\log_a n) = O(\log_b n)$ for any pair of constants $a, b > 1$.
- For every $x > 0$, $\log n = O(n^x)$.
- For every $r > 1$ and every $d > 0$, $n^d = O(r^n)$. 
Properties of Asymptotic Growth Rates

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 \( f = \Theta(h) \).
Properties of Asymptotic Growth Rates

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 \( f = \Theta(h) \).

Additivitvity

- If \( f = O(h) \) and \( g = O(h) \), then \( f + g = O(h) \).
- Similar statements hold for lower and tight bounds.
Properties of Asymptotic Growth Rates

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 \( f = O(h) \) and \( g = O(h) \), then \( f + g = O(h) \).
- Similar statements hold for lower and tight bounds.
- If \( k \) is a constant and there are \( k \) functions \( f_i = O(h), 1 \leq i \leq k \).
Properties of Asymptotic Growth Rates

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 \( f = O(h) \) and \( g = O(h) \), then \( f + g = O(h) \).
- Similar statements hold for lower and tight bounds.
- If \( k \) is a constant and there are \( k \) functions \( f_i = O(h), 1 \leq i \leq k \), then \( f_1 + f_2 + \ldots + f_k = O(h) \).
Properties of Asymptotic Growth Rates

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 \( f = O(h) \) and \( g = O(h) \), then \( f + g = O(h) \).
- Similar statements hold for lower and tight bounds.
- If \( k \) is a constant and there are \( k \) functions \( f_i = O(h), 1 \leq i \leq k \), then \( f_1 + f_2 + \ldots + f_k = O(h) \).
- If \( f = O(g) \), then \( f + g = \)
Properties of Asymptotic Growth Rates

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 \( f = O(h) \) and \( g = O(h) \), then \( f + g = O(h) \).
- Similar statements hold for lower and tight bounds.
- If \( k \) is a constant and there are \( k \) functions \( f_i = O(h), 1 \leq i \leq k \), then \( f_1 + f_2 + \ldots + f_k = O(h) \).
- If \( f = O(g) \), then \( f + g = \Theta(g) \).
**Linear Time**

- Running time is at most a constant factor times the size of the input.
Linear Time

- Running time is at most a constant factor times the size of the input.
- Finding the minimum, merging two sorted lists.
Linear Time

- Running time is at most a constant factor times the size of the input.
- Finding the minimum, merging two sorted lists.
- Sub-linear time.
Linear Time

- Running time is at most a constant factor times the size of the input.
- Finding the minimum, merging two sorted lists.
- Sub-linear time. Binary search in a sorted array of $n$ numbers takes $O(\log n)$ time.
\[ O(n \log n) \] Time

- Any algorithm where the costliest step is sorting.
Quadratic Time

- Enumerate all pairs of elements.
Quadratic Time

- Enumerate all pairs of elements.
- Given a set of $n$ points in the plane, find the pair that are the closest.
Quadratic Time

- Enumerate all pairs of elements.
- Given a set of $n$ points in the plane, find the pair that are the closest. Surprising fact: will solve this problem in $O(n \log n)$ time later in the semester.
Does a graph have an independent set of size $k$, where $k$ is a constant, i.e. there are $k$ nodes such that no two are joined by an edge?
Does a graph have an independent set of size $k$, where $k$ is a constant, i.e. there are $k$ nodes such that no two are joined by an edge?

Algorithm: For each subset $S$ of $k$ nodes, check if $S$ is an independent set. If the answer is yes, report it.
Does a graph have an independent set of size \( k \), where \( k \) is a constant, i.e. there are \( k \) nodes such that no two are joined by an edge?

Algorithm: For each subset \( S \) of \( k \) nodes, check if \( S \) is an independent set. If the answer is yes, report it.

Running time is \( O(n^k) \) time.
Does a graph have an independent set of size \( k \), where \( k \) is a constant, i.e. there are \( k \) nodes such that no two are joined by an edge?

Algorithm: For each subset \( S \) of \( k \) nodes, check if \( S \) is an independent set. If the answer is yes, report it.

Running time is \( O(k^2 \binom{n}{k}) = O(n^k) \).
Beyond Polynomial Time

- What is the largest size of an independent set in a graph with \( n \) nodes?
Beyond Polynomial Time

- What is the largest size of an independent set in a graph with $n$ nodes?
- Algorithm: For each $1 \leq i \leq n$, check if the graph has an independent size of size $i$. Output largest independent set found.
Beyond Polynomial Time

▶ What is the largest size of an independent set in a graph with $n$ nodes?
▶ Algorithm: For each $1 \leq i \leq n$, check if the graph has an independent size of size $i$. Output largest independent set found.
▶ What is the running time?
Beyond Polynomial Time

What is the largest size of an independent set in a graph with $n$ nodes?

Algorithm: For each $1 \leq i \leq n$, check if the graph has an independent size of size $i$. Output largest independent set found.

What is the running time? $O(n^2 2^n)$. 