Analysis of Algorithms

T. M. Murali

January 25, 2021
What is Algorithm Analysis?

- Measure resource requirements: how does the amount of time and space 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 does the amount of time and space 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.
- Bound the largest possible running time the algorithm over all inputs of size $n$, as a function of $n$. 

\[ T(n) = \text{worst-case running time of algorithm over all inputs of size } n \]

\[ T(n) = \Omega(f(n)) \]

\[ T(n) = O(g(n)) \]
Worst-case Running Time

- We will measure worst-case running time of an algorithm.
- Bound the largest possible running time the algorithm over all inputs of size $n$, as a function of $n$.
- *Input size* = number of elements in the input.
Worst-case Running Time

- We will measure worst-case running time of an algorithm.
- Bound the largest possible running time the algorithm over all inputs of size $n$, as a function of $n$.
- Input size = number of elements in the input. Values in the input do not matter, except for specific algorithms.
- Assume all elementary operations take unit time: assignment, arithmetic on a fixed-size number, comparisons, array lookup, following a pointer, etc.
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 \( n \times n! \). 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 $n \times n!$. Unacceptable in practice!

- Desirable scaling property: when the input size doubles, the algorithm should only slow down by some constant factor $c$. ▶ Poll
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 $n \times n!$. Unacceptable in practice!

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

- 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 $n \times n!$. Unacceptable in practice!

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

- 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.
Comparing Mathematical Functions

- Assume all (mathematical) functions take only positive arguments and values.
- Different algorithms for the same problem may have different (worst-case) running times.
- Example of sorting:
Comparing Mathematical Functions

- Assume all (mathematical) functions take only positive arguments and values.
- Different algorithms for the same problem may have different (worst-case) running times.
- Example of sorting: bubble sort, insertion sort, quick sort, merge sort, etc.

Example:
- Bubble sort and insertion sort take roughly $n^2$ comparisons while quick sort (only on average) and merge sort take roughly $n \log_2 n$ comparisons.

"Roughly" hides potentially large constants, e.g., running time of merge sort may in reality be $10^n n \log_2 n$.

How can make statements such as the following, in order to compare the running times of different algorithms?

- $100 n \log_2 n \leq n^2$
- $10000 n \leq n^2$
- $5 n^2 - 4 n \geq 1000 n \log_n$
Comparing Mathematical Functions

- Assume all (mathematical) functions take only positive arguments and values.
- Different algorithms for the same problem may have different (worst-case) running times.
- Example of sorting: bubble sort, insertion sort, quick sort, merge sort, etc.
- Bubble sort and insertion sort take roughly $n^2$ comparisons while quick sort (only on average) and merge sort take roughly $n \log_2 n$ comparisons.
  - “Roughly” hides potentially large constants, e.g., running time of merge sort may in reality be $10n \log_2 n$. 
Comparing Mathematical Functions

- Assume all (mathematical) functions take only positive arguments and values.
- Different algorithms for the same problem may have different (worst-case) running times.
- Example of sorting: bubble sort, insertion sort, quick sort, merge sort, etc.
- Bubble sort and insertion sort take roughly $n^2$ comparisons while quick sort (only on average) and merge sort take roughly $n \log_2 n$ comparisons.
  - “Roughly” hides potentially large constants, e.g., running time of merge sort may in reality be $10n \log_2 n$.
- How can make statements such as the following, in order to compare the running times of different algorithms?
  - $100n \log_2 n \leq n^2$
  - $10000n \leq n^2$
  - $5n^2 - 4n \geq 1000n \log n$
"$10000n \leq n^2$"
“$10000n \leq n^2$”

10000$n$ vs. $O(n^2)$
Upper Bound

Definition

Asymptotic upper bound: A function $f(n)$ is $O(g(n))$ if
for all $n$, $f(n) \leq c g(n)$.

$10000n$ is $O(n^2)$,
**Upper Bound**

**Definition**

**Asymptotic upper bound**: A function \( f(n) \) is \( O(g(n)) \) if there exists a constant \( c > 0 \) such that for all \( n \), \( f(n) \leq c g(n) \).

\[ 10000n \text{ is } O(n^2), \]

\[ c = 1, \quad n_0 = 10000 \]
Upper Bound

**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 $n \geq n_0$, $f(n) \leq c g(n)$.

10,000$n$ is $O(n^2)$,
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 $n \geq n_0$, $f(n) \leq c \cdot g(n)$.

10000n is $O(n^2)$, $c = 1$, $n_0 = 10000$
$100n \log_2 n$ and $n^2$

$100n \log_2 n$ is $O(n^2)$,
$100n \log_2 n$ and $n^2$

$100n \log_2 n$ is $O(n^2)$, $c = 1$, $n_0 = 1500$
100n \log_2 n and \ n^2

100n \log_2 n \ is \ O(n^2), \ c = 100, \ n_0 = 1
Lower Bound

Definition

Asymptotic lower bound: A function $f(n)$ is $\Omega(g(n))$ if for all $n \geq n_0$, we have $f(n) \geq c \cdot g(n)$.
### Lower Bound

**Definition**

**Asymptotic lower bound:** A function \( f(n) \) is \( \Omega(g(n)) \) if there exists constant \( c > 0 \) such that for all \( n \geq n_0 \), we have \( f(n) \geq cg(n) \).
**Lower Bound**

**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 \( f(n) \geq cg(n) \).
Lower Bound

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 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 \( f(n) \geq cg(n) \).
Lower Bound

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 $f(n) \geq cg(n)$.

$n \log_2 n/10$ is $\Omega(n)$, $c = 1$, $n_0 = 1024$
Meaning of “Lower Bound” in Different Contexts

- Mathematical functions:
  
  \[ n \log n / 10 = \Omega(n) \]
  
  This statement is purely about these two functions. Not in the context of any algorithm or problem.

- Algorithms:
  - The lower bound on the running time of bubble sort is \( \Omega(n^2) \).
  
  There is some input of \( n \) numbers that will cause bubble sort to take at least \( \Omega(n^2) \) time, e.g., input the numbers in decreasing order.

  - But there may be other, faster algorithms.

- Problems:
  - The problem of sorting \( n \) numbers has a lower bound of \( \Omega(n \log n) \).
  
  For any comparison-based sorting algorithm, there is at least one input for which that algorithm will take \( \Omega(n \log n) \) steps.

  - The stable matching problem has a lower bound of \( \Omega(n^2) \).
  
  For any algorithm, there is at least one input for which the algorithm will take \( \Omega(n^2) \) steps, even if all the preference matrices are already stored in memory (Ng and Hirschberg, SIAM J. Comput., 1990).
Meaning of “Lower Bound” in Different Contexts

- Mathematical functions: $n$ is a lower bound for $n \log n/10$, i.e., $n \log n/10 = \Omega(n)$. 
Meaning of “Lower Bound” in Different Contexts

- Mathematical functions: $n$ is a lower bound for $n \log n/10$, i.e., $n \log n/10 = \Omega(n)$. This statement is purely about these two functions. Not in the context of any algorithm or problem.
Meaning of “Lower Bound” in Different Contexts

- Mathematical functions: $n$ is a lower bound for $n \log n / 10$, i.e., $n \log n / 10 = \Omega(n)$. This statement is purely about these two functions. Not in the context of any algorithm or problem.

- Algorithms:
  - The lower bound on the running time of *bubble sort* is $\Omega(n^2)$. 

  ▶ There is some input of $n$ numbers that will cause bubble sort to take at least $\Omega(n^2)$ time, e.g., input the numbers in decreasing order. 

  ▶ There may be other, faster algorithms. 

- Problems:
  - The problem of sorting $n$ numbers has a lower bound of $\Omega(n \log n)$. For any comparison-based sorting algorithm, there is at least one input for which that algorithm will take $\Omega(n \log n)$ steps. 

  ▶ The stable matching problem has a lower bound of $\Omega(n^2)$. For any algorithm, there is at least one input for which the algorithm will take $\Omega(n^2)$ steps, even if all the preference matrices are already stored in memory (Ng and Hirschberg, *SIAM J. Comput.* 1990).
Meaning of “Lower Bound” in Different Contexts

- Mathematical functions: \( n \) is a lower bound for \( n \log n/10 \), i.e., \( n \log n/10 = \Omega(n) \). This statement is purely about these two functions. Not in the context of any algorithm or problem.

- Algorithms:
  - The lower bound on the running time of \textit{bubble sort} is \( \Omega(n^2) \). There is some input of \( n \) numbers that will cause bubble sort to take at least \( \Omega(n^2) \) time, e.g.,
Meaning of “Lower Bound” in Different Contexts

- Mathematical functions: \( n \) is a lower bound for \( n \log n/10 \), i.e., \( n \log n/10 = \Omega(n) \). This statement is purely about these two functions. Not in the context of any algorithm or problem.

- Algorithms:
  - The lower bound on the running time of bubble sort is \( \Omega(n^2) \). There is some input of \( n \) numbers that will cause bubble sort to take at least \( \Omega(n^2) \) time, e.g., input the numbers in decreasing order.
  - But there may be other, faster algorithms.
Meaning of “Lower Bound” in Different Contexts

- Mathematical functions: $n$ is a lower bound for $n \log n/10$, i.e., $n \log n/10 = \Omega(n)$. This statement is purely about these two functions. Not in the context of any algorithm or problem.

- Algorithms:
  - The lower bound on the running time of bubble sort is $\Omega(n^2)$. There is some input of $n$ numbers that will cause bubble sort to take at least $\Omega(n^2)$ time, e.g., input the numbers in decreasing order.
  - But there may be other, faster algorithms.

- Problems:
  - The problem of sorting $n$ numbers has a lower bound of $\Omega(n \log n)$.
Meaning of “Lower Bound” in Different Contexts

- Mathematical functions: $n$ is a lower bound for $n \log n/10$, i.e., $n \log n/10 = \Omega(n)$. This statement is purely about these two functions. Not in the context of any algorithm or problem.

- Algorithms:
  - The lower bound on the running time of bubble sort is $\Omega(n^2)$. There is some input of $n$ numbers that will cause bubble sort to take at least $\Omega(n^2)$ time, e.g., input the numbers in decreasing order.
  - But there may be other, faster algorithms.

- Problems:
  - The problem of sorting $n$ numbers has a lower bound of $\Omega(n \log n)$. For any comparison-based sorting algorithm, there is at least one input for which that algorithm will take $\Omega(n \log n)$ steps.
Meaning of “Lower Bound” in Different Contexts

- Mathematical functions: $n$ is a lower bound for $n \log n / 10$, i.e.,
  \[ n \log n / 10 = \Omega(n). \] This statement is purely about these two functions. Not in the context of any algorithm or problem.

- Algorithms:
  - The lower bound on the running time of bubble sort is $\Omega(n^2)$. There is some input of $n$ numbers that will cause bubble sort to take at least $\Omega(n^2)$ time, e.g., input the numbers in decreasing order.
  - But there may be other, faster algorithms.

- Problems:
  - The problem of sorting $n$ numbers has a lower bound of $\Omega(n \log n)$. For any comparison-based sorting algorithm, there is at least one input for which that algorithm will take $\Omega(n \log n)$ steps.
  - The stable matching problem has a lower bound of $\Omega(n^2)$. 
Meaning of “Lower Bound” in Different Contexts

- Mathematical functions: $n$ is a lower bound for $n \log n/10$, i.e., $n \log n/10 = \Omega(n)$. This statement is purely about these two functions. Not in the context of any algorithm or problem.

- Algorithms:
  - The lower bound on the running time of bubble sort is $\Omega(n^2)$. There is some input of $n$ numbers that will cause bubble sort to take at least $\Omega(n^2)$ time, e.g., input the numbers in decreasing order.
  - But there may be other, faster algorithms.

- Problems:
  - The problem of sorting $n$ numbers has a lower bound of $\Omega(n \log n)$. For any comparison-based sorting algorithm, there is at least one input for which that algorithm will take $\Omega(n \log n)$ steps.
  - The stable matching problem has a lower bound of $\Omega(n^2)$. For any algorithm, there is at least one input for which the algorithm will take $\Omega(n^2)$ steps, even if all the preference matrices are already stored in memory (Ng and Hirschberg, *SIAM J. Comput.*, 1990).
Tight Bound

Definition

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

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 all these definitions, $c$ and $n_0$ are constants independent of $n$.
- Abuse of notation: say $g(n) = O(f(n))$, $g(n) = \Omega(f(n))$, $g(n) = \Theta(f(n))$. 
Properties of Asymptotic Growth Rates

Dropping argument $n$ on this slide for visual clarity.

**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 $f + g = \Theta(g)$. 

T. M. Murali January 25, 2021 Analysis of Algorithms
Properties of Asymptotic Growth Rates

Dropping argument $n$ on this slide for visual clarity.

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.
Dropping argument $n$ on this slide for visual clarity.

**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

Dropping argument $n$ on this slide for visual clarity.

**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

$$f_1 + f_2 + \ldots + f_k =$$
Dropping argument $n$ on this slide for visual clarity.

**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

Dropping argument $n$ on this slide for visual clarity.

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

Dropping argument \( n \) on this slide for visual clarity.

**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 \( f + g = \Theta(g) \).
# Examples

<table>
<thead>
<tr>
<th>( f(n) )</th>
<th>( g(n) )</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>( pn^2 + qn + r )</td>
<td>( O(n^3) )</td>
<td>( n^2 \leq n^3 ), if ( n \geq 1 )</td>
</tr>
<tr>
<td>( pn^2 + qn + r )</td>
<td>( O(n^d) )</td>
<td>( d ) is an integer constant and ( a_d &gt; 0 )</td>
</tr>
<tr>
<td>( \sum_{0 \leq i \leq d} a_i n^i )</td>
<td>( O(n^d) )</td>
<td>( d ) is the definition of polynomial time.</td>
</tr>
<tr>
<td>( O(n^{1.59}) )</td>
<td>( O(n^{1.59}) )</td>
<td>For every constant ( r &gt; 1 ) and every constant ( d &gt; 0 ), ( n^d = O(r^n) ), e.g., ( n^3 = O(1.1^n) ).</td>
</tr>
<tr>
<td>( \log_a n )</td>
<td>( O(n) )</td>
<td>For every constant ( x &gt; 0 ), ( \log n = O(n^x) ), e.g., ( \log n = O(n^{0.00001}) ).</td>
</tr>
</tbody>
</table>
### Examples

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$g(n)$</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$\Theta(n^2)$</td>
<td></td>
</tr>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$\Theta(n^2)$</td>
<td></td>
</tr>
<tr>
<td>$\sum_{0\leq i\leq d} a_i n^i$</td>
<td>$O(n^{1.59})$</td>
<td></td>
</tr>
<tr>
<td>$O(n^{1.59})$</td>
<td>$O(n^{1.59})$</td>
<td></td>
</tr>
<tr>
<td>$\log_a n$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Examples

<table>
<thead>
<tr>
<th>( f(n) )</th>
<th>( g(n) )</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>( pn^2 + qn + r )</td>
<td>( \Theta(n^2) )</td>
<td></td>
</tr>
<tr>
<td>( pn^2 + qn + r )</td>
<td>( O(n^3) )?</td>
<td></td>
</tr>
<tr>
<td>( \sum_{0 \leq i \leq d} a_i n^i )</td>
<td>( O(n^{1.59}) )</td>
<td></td>
</tr>
<tr>
<td>( \log_a n )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For every constant \( x > 0 \), \( \log n = O(n^x) \), e.g., \( \log n = n^{0.00001} \).

For every constant \( r > 1 \) and every constant \( d > 0 \), \( n^d = O(r^n) \), e.g., \( n^3 = O(1.1^n) \).
## Examples

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$g(n)$</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$\Theta(n^2)$</td>
<td>$n^2 \leq n^3$, if $n \geq 1$</td>
</tr>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$O(n^3)$?</td>
<td></td>
</tr>
<tr>
<td>$\sum_{0 \leq i \leq d} a_i n^i$</td>
<td>$O(n^{1.59})$</td>
<td></td>
</tr>
<tr>
<td>$O(n^{1.59})$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\log_a n$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Examples

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$g(n)$</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$\Theta(n^2)$</td>
<td>$n^2 \leq n^3$, if $n \geq 1$</td>
</tr>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$O(n^3)$?</td>
<td>$n^2 \leq n^3$, if $n \geq 1$</td>
</tr>
<tr>
<td>$\sum_{0 \leq i \leq d} a_in^i$</td>
<td>$\Theta(n^d)$</td>
<td>if $d &gt; 0$ is an integer constant and $a_d &gt; 0$</td>
</tr>
<tr>
<td>$O(n^{1.59})$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\log_a n$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $O(n^d)$ is the definition of *polynomial time*.
## Examples

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$g(n)$</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$\Theta(n^2)$</td>
<td>$n^2 \leq n^3$, if $n \geq 1$</td>
</tr>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$O(n^3)$?</td>
<td>if $d &gt; 0$ is an integer constant and $a_d &gt; 0$</td>
</tr>
<tr>
<td>$\sum_{0 \leq i \leq d} a_i n^i$</td>
<td>$\Theta(n^d)$</td>
<td></td>
</tr>
<tr>
<td>$O(n^{1.59})$</td>
<td>Polynomial time?</td>
<td></td>
</tr>
<tr>
<td>$\log_a n$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $O(n^d)$ is the definition of *polynomial time*. 

---

T. M. Murali  
January 25, 2021  
Analysis of Algorithms
# Examples

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$g(n)$</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$\Theta(n^2)$</td>
<td>$n^2 \leq n^3$, if $n \geq 1$</td>
</tr>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$O(n^3)$?</td>
<td>if $d &gt; 0$ is an integer constant and $a_d &gt; 0$</td>
</tr>
<tr>
<td>$\sum_{0 \leq i \leq d} a_in^i$</td>
<td>$\Theta(n^d)$</td>
<td></td>
</tr>
<tr>
<td>$O(n^{1.59})$</td>
<td>Polynomial time? Poll</td>
<td>Yes, since $n^{1.59}$ is $O(n^2)$</td>
</tr>
<tr>
<td>$\log_a n$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $O(n^d)$ is the definition of *polynomial time*. 
## Examples

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$g(n)$</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$\Theta(n^2)$</td>
<td>$n^2 \leq n^3$, if $n \geq 1$</td>
</tr>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$O(n^3)$?</td>
<td>if $d &gt; 0$ is an integer constant and $a_d &gt; 0$</td>
</tr>
<tr>
<td>$\sum_{0 \leq i \leq d} a_i n^i$</td>
<td>$\Theta(n^d)$</td>
<td></td>
</tr>
<tr>
<td>$O(n^{1.59})$</td>
<td>Polynomial time?</td>
<td>Yes, since $n^{1.59}$ is $O(n^2)$</td>
</tr>
<tr>
<td>$\log_a n$</td>
<td>$O(\log_b n)$</td>
<td></td>
</tr>
</tbody>
</table>

- $O(n^d)$ is the definition of *polynomial time*. 
## Examples

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$g(n)$</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$\Theta(n^2)$</td>
<td>$n^2 \leq n^3$, if $n \geq 1$</td>
</tr>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$O(n^3)$?</td>
<td>if $d &gt; 0$ is an integer constant and $a_d &gt; 0$</td>
</tr>
<tr>
<td>$\sum_{0 \leq i \leq d} a_i n^i$</td>
<td>$\Theta(n^d)$</td>
<td>Yes, since $n^{1.59}$ is $O(n^2)$</td>
</tr>
<tr>
<td>$O(n^{1.59})$</td>
<td>Polynomial time?</td>
<td>Yes, for any pair of constants $a, b &gt; 1$</td>
</tr>
<tr>
<td>$\log_a n$</td>
<td>$O(\log_b n)$</td>
<td>$O(n^d)$ is the definition of <em>polynomial time</em>.</td>
</tr>
</tbody>
</table>

- $O(n^d)$ is the definition of *polynomial time*. 

---

T. M. Murali  January 25, 2021  Analysis of Algorithms
## Examples

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$g(n)$</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$\Theta(n^2)$</td>
<td>$n^2 \leq n^3$, if $n \geq 1$</td>
</tr>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$O(n^3)$?</td>
<td>if $d &gt; 0$ is an integer constant and $a_d &gt; 0$</td>
</tr>
<tr>
<td>$\sum_{0 \leq i \leq d} a_i n^i$</td>
<td>$\Theta(n^d)$</td>
<td></td>
</tr>
<tr>
<td>$O(n^{1.59})$</td>
<td>Polynomial time?</td>
<td>Yes, since $n^{1.59}$ is $O(n^2)$</td>
</tr>
<tr>
<td>$\log_a n$</td>
<td>$O(\log_b n)$</td>
<td>Yes, for any pair of constants $a, b &gt; 1$</td>
</tr>
</tbody>
</table>

- $O(n^d)$ is the definition of *polynomial time*.
- For every constant $x > 0$, $\log n = O(n^x)$, e.g., $\log n = n^{0.00001}$. 

T. M. Murali January 25, 2021 Analysis of Algorithms
## Examples

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$g(n)$</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$\Theta(n^2)$</td>
<td>$n^2 \leq n^3$, if $n \geq 1$</td>
</tr>
<tr>
<td>$pn^2 + qn + r$</td>
<td>$O(n^3)$?</td>
<td>if $d &gt; 0$ is an integer constant and $a_d &gt; 0$</td>
</tr>
<tr>
<td>$\sum_{0 \leq i \leq d} a_i n^i$</td>
<td>$\Theta(n^d)$</td>
<td>Yes, since $n^{1.59}$ is $O(n^2)$</td>
</tr>
<tr>
<td>$O(n^{1.59})$</td>
<td>Polynomial time?</td>
<td>Yes, for any pair of constants $a, b &gt; 1$</td>
</tr>
<tr>
<td>$\log_a n$</td>
<td>$O(\log_b n)$</td>
<td></td>
</tr>
</tbody>
</table>

- $O(n^d)$ is the definition of *polynomial time*.
- For every constant $x > 0$, $\log n = O(n^x)$, e.g., $\log n = n^{0.00001}$.
- For every constant $r > 1$ and every constant $d > 0$, $n^d = O(r^n)$, e.g., $n^3 = O(1.1^n)$. 
Different functions of $n$

- $n$
- $n \log n$
- $n^2$
- $n^3$
- $2^n$
More functions of $n$

- $n$
- $\log_2 n$
- $\log_3 n$
- $n^{0.5}$
Running time is at most a constant factor times the size of the input.
Running time is at most a constant factor times the size of the input.
Finding the minimum, merging two sorted lists.
• Running time is at most a constant factor times the size of the input.
• Finding the minimum, merging two sorted lists.
• Computing the median (or kth smallest) element in an unsorted list.
Running time is at most a constant factor times the size of the input.
Finding the minimum, merging two sorted lists.
Computing the median (or $k$th smallest) element in an *unsorted* list. “Median-of-medians” algorithm.
Sub-linear time.

### Linear Time

- $n$
- $n \log n$
- $n^2$
- $n^3$
- $2^n$
• Running time is at most a constant factor times the size of the input.
• Finding the minimum, merging two sorted lists.
• Computing the median (or $k$th smallest) element in an unsorted list. “Median-of-medians” algorithm.
• Sub-linear time. Binary search in a sorted array of $n$ numbers takes $O(\log n)$ time.
Any algorithm where the costliest step is sorting.
Enumerate all pairs of elements.
Enumerate all pairs of elements.

Given a set of $n$ points in the plane, find the pair that are the closest.
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.
COVID-19 proximity graph: each node is a person shopping in Kroger, an edge connects two people who came within six feet of each other.

- Some subgraphs can have high potential for virus transmission.

\[ O(n^k) \text{ Time} \]
COVID-19 proximity graph: each node is a person shopping in Kroger, an edge connects two people who came within six feet of each other.

Some subgraphs can have high potential for virus transmission.

Does a graph have a clique of size $k$, where $k$ is a constant, i.e. there are $k$ nodes such that every pair is connected by an edge?
COVID-19 proximity graph: each node is a person shopping in Kroger, an edge connects two people who came within six feet of each other.

Some subgraphs can have high potential for virus transmission.

Does a graph have a \textit{clique} of size \(k\), where \(k\) is a constant, i.e. there are \(k\) nodes such that every pair is connected by an edge?
COVID-19 proximity graph: each node is a person shopping in Kroger, an edge connects two people who came within six feet of each other.

Some subgraphs can have high potential for virus transmission.

Does a graph have a *clique* of size $k$, where $k$ is a constant, i.e. there are $k$ nodes such that every pair is connected by an edge?
COVID-19 proximity graph: each node is a person shopping in Kroger, an edge connects two people who came within six feet of each other.

Some subgraphs can have high potential for virus transmission.

Does a graph have a **clique** of size $k$, where $k$ is a constant, i.e., there are $k$ nodes such that every pair is connected by an edge? How do we find such a clique?
COVID-19 proximity graph: each node is a person shopping in Kroger, an edge connects two people who came within six feet of each other.

Some subgraphs can have high potential for virus transmission.

Does a graph have a clique of size \( k \), where \( k \) is a constant, i.e. there are \( k \) nodes such that every pair is connected by an edge? How do we find such a clique?

Algorithm: For each subset \( S \) of \( k \) nodes, check if \( S \) is a clique. If the answer is yes, report it.
COVID-19 proximity graph: each node is a person shopping in Kroger, an edge connects two people who came within six feet of each other.

Some subgraphs can have high potential for virus transmission.

Does a graph have a clique of size $k$, where $k$ is a constant, i.e. there are $k$ nodes such that every pair is connected by an edge? How do we find such a clique?

Algorithm: For each subset $S$ of $k$ nodes, check if $S$ is a clique. If the answer is yes, report it.

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

$O(n^k)$ Time

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n \log n$</th>
<th>$n^2$</th>
<th>$n^3$</th>
<th>$2^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>400</td>
<td>800</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>1,500</td>
<td>1,000</td>
<td>2,400</td>
<td>256</td>
</tr>
<tr>
<td>6</td>
<td>5,000</td>
<td>5,000</td>
<td>12,000</td>
<td>6,553</td>
</tr>
<tr>
<td>8</td>
<td>15,000</td>
<td>15,000</td>
<td>36,000</td>
<td>262,144</td>
</tr>
<tr>
<td>10</td>
<td>45,000</td>
<td>45,000</td>
<td>100,000</td>
<td>1,099,511</td>
</tr>
<tr>
<td>12</td>
<td>135,000</td>
<td>135,000</td>
<td>270,000</td>
<td>4,194,304</td>
</tr>
</tbody>
</table>
What is the largest size of a clique in a graph with $n$ nodes?

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

What is the running time? $O(n^2)$.
What is the largest size of a clique in a graph with $n$ nodes?

Algorithm: For each $1 \leq i \leq n$, check if the graph has a clique of size $i$. Output largest clique found.
What is the largest size of a clique in a graph with \( n \) nodes?

Algorithm: For each \( 1 \leq i \leq n \), check if the graph has a clique of size \( i \). Output largest clique found.

What is the running time?

\[ O\left( n^2 \right) \]
What is the largest size of a clique in a graph with \( n \) nodes?

Algorithm: For each \( 1 \leq i \leq n \), check if the graph has a clique of size \( i \). Output largest clique found.

What is the running time? \( O(n^2 2^n) \).
Results of Poll on Teaching Style

Thank you for the responses!

1. Class speed: Just right (73%)
2. iPad+doodling was helpful: Yes (94%)
Results of Poll on Teaching Style

Thank you for the responses!

1. Class speed: Just right (73%)
2. iPad+doodling was helpful: Yes (94%)
   - But improve your handwriting!
   - iPad screen is small and a little blurry in video recording.

Polls
- Zoom help me think (98%)
- There should be more Zoom polls (71%)

Other suggestions:
- Use Discord for TA office hours (I am open, if there are many more requests)
- Please upload videos (Yes, I do later in the day)
- What type of notes should I take?
  - Post slides with drawings (Will video recording suffice as a substitute?)
- Like, love, super engaging, favorite class, enjoying, fun.
Results of Poll on Teaching Style

Thank you for the responses!

1. Class speed: Just right (73%)

2. iPad+doodling was helpful: Yes (94%)
   - But improve your handwriting!
   - iPad screen is small and a little blurry in video recording.

3. Polls
   - Zoom help me think (98%)
   - There should be more Zoom polls (71%)
   - Some suggestions to give less time

4. Web-based forum for questions and answers: Piazza (70%)
Results of Poll on Teaching Style

Thank you for the responses!

1. Class speed: Just right (73%)
2. iPad+doodling was helpful: Yes (94%)
   - But improve your handwriting!
   - iPad screen is small and a little blurry in video recording.
3. Polls
   - Zoom help me think (98%)
   - There should be more Zoom polls (71%)
   - Some suggestions to give less time
4. Web-based forum for questions and answers: Piazza (70%)
5. Other suggestions:
   - Use Discord for TA office hours (I am open, if there are many more requests)
   - Please upload videos (Yes, I do later in the day)
   - What type of notes should I take?
   - Post slides with drawings (Will video recording suffice as a substitute?)
Results of Poll on Teaching Style

Thank you for the responses!

1. Class speed: Just right (73%)
2. iPad+doodling was helpful: Yes (94%)
   ▶ But improve your handwriting!
   ▶ iPad screen is small and a little blurry in video recording.
3. Polls
   ▶ Zoom help me think (98%)
   ▶ There should be more Zoom polls (71%)
   ▶ Some suggestions to give less time
4. Web-based forum for questions and answers: Piazza (70%)
5. Other suggestions:
   ▶ Use Discord for TA office hours (I am open, if there are many more requests)
   ▶ Please upload videos (Yes, I do later in the day)
   ▶ What type of notes should I take?
   ▶ Post slides with drawings (Will video recording suffice as a substitute?)
   ▶ Like, love, super engaging, favorite class, enjoying, fun.
Results of Poll on PQs and Graph Searches

1. Priority queues: Refresher (57%), Summary (31%)
2. Breadth-first search: Refresher (47%), Summary (39%)
3. Depth-first search: Refresher (49%), Summary (39%)