Asymptotic Notation (Big O)

https://cs.pomona.edu/classes/cs140
Outline

Topics and Learning Objectives
- Discuss total running time
- Discuss asymptotic running time
- Learn about asymptotic notation

Exercises
- Running time
Extra Resources

• Chapter 3: asymptotic notation
Comparing Algorithms and Data Structures

We like to compare algorithms and data structures
• Speed
• Memory usage

We don’t always need to care about little details

We ignore some details anyway
• Data locality
• Differences among operations
Constants

\(0.01n^2\)

\(100n\log_2(n)\)
Big-O Example Code (ODS 1.3.3)

```python
# function_one has a total running time of 2n log n + 2n - 250
a = function_one(input_one)

# function_two has a total running time of 3n log n + 6n + 48
b = function_two(input_two)

• The total running time of the code above is:

\[ 2n \log n + 2n - 250 + 1 + 3n \log n + 6n + 48 + 1 \]

\[ 5n \log n + 8n - 200 \]
Big-O Example Math (ODS 1.3.3)

\[ 5n \log n + 8n - 200 \]

• We don’t care about most of these details
• We want to be able to quickly glance at the running time of an algorithm and know how it compares to others
• So we say the following

\[ 5n \log n + 8n - 200 = O(n \log n) \]
Big-O (Asymptotic Running Time)

If and only if (iff) we can find values for \( c, n_0 > 0 \), such that

\[
T(n) \leq c \cdot f(n), \text{ where } n \geq n_0
\]

Note: \( c, n_0 \) cannot depend on \( n \)
Searching an array for a given number?

Write an algorithm (in pseudocode):

```
Function FindNum(array, num)
    For val In array
        If val == num
            Return True
        End If
    End For
    Return False
```

What is the total running time?

$T(n) = 2n + 1$
Searching an array for a given number?

What is the asymptotic running time? $T(n) = 2n + 1$

$T(n) = O(\,?)$

$T(n) = O(n)$

$T(n) \leq c \cdot n$

$2n + 1 \leq c \cdot n$

$\forall n \geq n_0$

$3n \leq c \cdot n$

$c = 3$

$n_0 = 1$
Search two separate arrays (sequentially) for a given number?

Write an algorithm (in pseudocode):

```plaintext
Function FindNumIn2 (array1, array2, num):
    if FindNum (array1, num) OR FindNum (array2, num)
        return 1
    n = max (array1.length, array2.length)
    T(n) = 2n + O + C2n + O + O + \Theta
```

What is the total running time?
Search two separate arrays (sequentially) for a given number?

What is the asymptotic running time? \( T(n) = 4n + 3 = O(n) \)

\[
T(n) = O(n)
\]

\[
4n + 3 \leq Cn \quad \forall \quad n \geq n_0
\]

\[
4n + 3 \leq 4n + 3n \leq Cn \quad \forall \quad n \geq n_0
\]

\[
3 \leq 3n \Rightarrow n \geq 1
\]

\[
c = 7, \quad n_0 = 1
\]
Searching two arrays for any common number?

Write an algorithm (in pseudocode):

```plaintext
Function Find Common (array1, array2)
  For val1 In array1
    If FindLater (array2, val1) (2n+1)
      Return True
  Return False
```

What is the total running time?

\[ T(n) = n + n \times (2n+1) + 1 \]

\[ = 2n^2 + n + n + 1 \]
Searching two arrays for any common number?

What is the asymptotic running time? $T(n) = 2n^2 + 2n + 1$

$$T(n) \neq O(n)$$

$$\frac{2n^2 + 2n + 1}{n} \leq \frac{c n}{n}$$

$$2n + 2 + \frac{1}{n} \leq c$$

$\forall n \geq n_0$

$1,000,000 \geq 200$

$n_0 = 200$
Searching two arrays for any common number?

What is the asymptotic running time? \( T(n) = 2n^2 + 2n + 1 \)

\[
T(n) = O(n^2)
\]

\[
2n^2 + 2n + 1 \leq c \cdot n^2
\]

\[
2n^2 + 2n + 1 \leq 2n^2 + zn^2 + 1n^2 \leq c \cdot n^2
\]

For all \( n \geq n_0 \)

\[
1 \leq n \quad n \geq 1
\]

\[
C = S_1, \quad n_0 = 1
\]
Searching a single array for duplicate numbers?

Write an algorithm (in pseudocode):  

```plaintext
Function FindDuplicate (array)
    array = Merge Sort (array) \leq \frac{Zn \log n + Zn}{ln}
    For i In [1..<array.length] \leq \frac{ln}{ln}
        If array[i-1] == array[i] \leq \frac{3n}{ln}
            Return True
    Return False + 1
```

What is the total running time?

- \(2 \ln \log n + 2n + 4n + 1\)
- \(2 \ln \log n + 2Sn + 1\)
Searching a single array for duplicate numbers?

What is the asymptotic running time? \( T(n) = 21n \lg n + 25n + 1 \)

\[
T(n) = O(n \lg n)
\]

\[
\frac{21n \lg n + 25n + 1}{n \lg n} \leq Cn \lg n \quad \forall \ n \geq n_0
\]

\[
\rightarrow 21 + \frac{25}{\lg n} + \frac{1}{n \lg n} \leq C \quad \forall \ n \geq n_0
\]

\[
\frac{25}{\lg n} \leq 0 \rightarrow \frac{25}{\lg 2} \rightarrow \frac{25}{25 \lg 2} \rightarrow \frac{1}{\lg 2} \rightarrow \frac{1}{2} < 1
\]
Searching a single array for duplicate numbers?

What is the asymptotic running time? \( T(n) = 21n \log n + 25n + 1 \)

\[ 21 + 1 + \frac{1}{n \log n} \leq c \quad \forall \ n \geq n_0 \]

\[ n_0 \geq 2^{25} \]

\[ c = 221, \quad n_0 = 2^{25} \]
Exercise
Big-O Examples

• Claim: \( 2^{n+10} = O(2^n) \)

\[
2^{n+10} \leq c \cdot 2^n + n \geq n_0
\]

\[
2^n \cdot 2^{10} \leq c \cdot 2^n + n \geq n_0
\]

\[ c = 2^{10}, \quad n_0 = 1 \]
Big-O Examples

• Claim: \(2^{10n} \neq O(2^n)\)

\[
\frac{2^{10n}}{2^n} \leq c^{2^n} \quad \forall \ n \geq n_0
\]

\[
2^{10n-n} \leq c
\]

\[
2^{an} \leq c \quad \forall \ n \geq n_0
\]
Big-O Examples

• Claim: for every $k \geq 1$, $n^k$ is not $O(n^{k-1})$

$$\forall k \geq 1 \quad n^k \neq O(n^{k-1})$$

Note: $c, n_0$ cannot depend on $n$
Examples

• Claim: $21n (\log_2(n) + 1) = \Theta(n\log_2 n)$
Other Notations

• **Big-O** ($\leq$): $T(n) = O(f(n))$ if $T(n) \leq c f(n)$, where $n \geq n_0$

• **Big-Omega** ($\geq$): $T(n) = \Omega(f(n))$ if $T(n) \geq c f(n)$, where $n \geq n_0$

• **Theta** ($=$): $T(n) = \Theta(f(n))$ if $T(n) = O(f(n))$ and $T(n) = \Omega(f(n))$

$$c_1 f(n) \leq T(n) \leq c_2 f(n), \text{ where } n \geq n_0$$
Other Notations

- **Big-O (≤)**: $T(n) = O(f(n))$ if $T(n) \leq c f(n)$, where $n \geq n_0$

- **Little-o (<)**

- **Big-Omega (≥)**: $T(n) = \Omega(f(n))$ if $T(n) \geq c f(n)$, where $n \geq n_0$

- **Little-omega (>)**
Examples

- **Claim:** $21n (\log_2(n) + 1) = \Theta(n \log_2 n)$

$$21n \log n + 21n \leq c_2 n \log n \quad \forall n \geq n_0$$

$$21n \log n + 21n \leq 21n \log n + 21n \log n \leq c_n \log n$$

$$21n \log n \leq 21n \log n$$

$$21n \leq 21 + 1 \log n \leq \log n \quad n \geq 2$$

$T(n) = \Theta(f(n))$

If and only if we can find values for $c, n_0 > 0$, such that

$$c_1 f(n) \leq T(n) \leq c_2 f(n), \text{ where } n \geq n_0$$

Note: $c_1, c_2, n_0$ cannot depend on $n$
Examples

Claim: $21n (\log_2(n) + 1) = \Theta(n \log_2 n)$

$c_2 \geq 22$

$n_0 = 22$
\[ T(n) = \Theta(f(n)) \]

If and only if we can find values for \( c, n_0 > 0 \), such that
\[ c_1 f(n) \leq T(n) \leq c_2 f(n), \quad \text{where } n \geq n_0 \]

Note: \( c_1, c_2, n_0 \) cannot depend on \( n \)

**Examples**

- Claim: \( 21n \log_2(n) + 1 = \Theta(n \log_2 n) \)

\[
\begin{align*}
    c_1 n \log n & \leq 21n \log n + 21 \quad \forall n \geq n_0 \\
    c_1 n \log n & \leq 21n \log n \\
    c_1 n \log n & \leq 21n \log n \\
    c_1 n \log n & \leq 21n \log n \\
\end{align*}
\]

\[
\begin{align*}
    c_1 = 21, & \quad \text{and } c_2 = 42, \quad n_0 = 2
\end{align*}
\]
$O(f(n))$: $T(n) \leq c_2 f(n)$

$\Theta(f(n))$: $c_1 f(n) \leq T(n) \leq c_2 f(n)$

$\Omega(f(n))$: $T(n) \geq c_1 f(n)$
What is \( f(n) \)?

What are good values for:
- \( c \)
- \( n_0 \)
Insertion Sort vs Merge Sort

Computer A: Insertion Sort
- Time: 5.5 hours
- 10 million numbers
- Performance: $O(n^2)$
- MIPS: 10,000
- Complexity: $2n^2$

Computer B: Merge Sort
- Time: 20 minutes
- 100 million numbers
- Performance: $O(n \log n)$
- MIPS: 10
- Complexity: $50n \log n$
Simplifying the Comparison

• Why can we remove leading coefficients?
• Why can we remove lower order terms?

• They are both insignificant when compared with the growth of the function.
• They both get factored into the constant “c”