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

Assessments

• Running time activity
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:

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

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

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

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

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**? $T(n) = 2n + 1$

$T(n) = O(?)$

$T(n) = O(n)$

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

$\forall n \geq n_0$

$n \geq n_0$

$c = 2$

$n_0 = 1$
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
    n7 = max (array1.length, or array2.length)
    T(n) = 2n + 0 + Cn + 0 + O + T
```

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

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

$4n + 3 \leq cn$ for $n \geq n_0$

$4n + 3 \leq 4n + 3n \leq cn$ for $n \geq n_0$

$3 \leq 3n \Rightarrow n \geq 1$

$c = 7$, $n_0 = 1$
Searching two arrays for any common number?

Write an algorithm (in pseudocode):

Function Find Common (array1, array2)

For val1 In array1
  If Find (array2, val1) (Zn+1)
    Return True

Return False

What is the total running time?

T(n) = n + n(2n+1) + 1
     = 2n^2 + n + n + 1
Searching two arrays for any common number?

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

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

\[ \frac{2n^2 + 2n + 1}{n} \leq \frac{cn}{n} \]

$2n^2 + 2n + 1 \leq cn^2$

$n \geq n_0$

$2 \cdot 1,000,000 + 2 + \frac{1}{n} \leq c$ (for $n \geq 200$)

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

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

$T(n) = O(n^2)$

$2n^2 + 2n + 1 \leq Cn^2$

$2n^2 + 2n + 1 \geq \frac{1}{2}n^2 + \frac{1}{2}n + 1 \geq Cn^2$ for $n \geq n_0$

$T(n) \leq c \cdot f(n)$, where $n \geq n_0$
Searching a single array for duplicate numbers?

Write an algorithm (in pseudocode):

```
Function FindDuplicate (array)
    array = Merge Sort (array) ≤ 2ln lg n + 2ln
    For i In [1 .. < array length]
        If array[i-1] == array[i]
            Return True
    Return False + 1
```

What is the total running time?

```
≤ 2ln lg n + 2ln + 3n + 1
```

```
≤ 2ln lg n + 2Sn + 1
```
Searching a single array for duplicate numbers?

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

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

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

\[
\Rightarrow 21 + \left( \frac{25}{\log n} \right) + \left( \frac{1}{n \log n} \right) \leq \frac{C}{\log n} \quad \forall \ n \geq n_0
\]

\[
\Rightarrow \frac{25}{\log n} \leq 1 \Rightarrow n \geq 2^{25}
\]

\[
\Rightarrow \frac{25}{\log 2} = 1 \Rightarrow \frac{25}{\log 2} \rightarrow \frac{1}{\log 2} \rightarrow 1 < 1
\]
Searching a single array for duplicate numbers?

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

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

\[
n_0 \geq 2^{2^5}
\]

\[
\frac{1}{n \log n} \leq 1 \quad \Rightarrow \quad 21 + 1 + 1 \leq c \quad \forall \ n \geq 2^{2^5}
\]

\[
c = 221 \quad n_0 = 2^{2^5}
\]
**Exercise**

Big-O Examples

- Claim: \(2^{n+10} = O(2^n)\)

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

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

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

• Claim: $2^{10n} \neq O(2^n)$

$2^{10n-n} \leq c$ for $n \geq n_0$

$2^{9n} \leq c$ for $n \geq n_0$
Big-O Examples

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

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

\[
n^k \leq C(n^{k-1}) \quad \forall n \geq n_0
\]

\[
n^k \leq Cn^{k-1} \quad \forall n \geq n_0
\]

\[n \leq \frac{C}{n^k} \quad \forall n \geq n_0\]

\[\therefore \text{Claim is true}\]
Examples

- Claim: $21n \left( \log_2(n) + 1 \right) = \Theta(n \log_2 n)$
Other Notations

- **Big-O (≤)**: $T(n) = O(f(n))$ if $T(n) \leq c f(n)$, where $n \geq n_0$
- **Big-Omega (≥)**: $T(n) = \Omega(f(n))$ 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 ($\leq$) : $T(n) = O(f(n))$ if $T(n) \leq c f(n)$, where $n \geq n_0$
• little-o ($<$)

• Big-Omega ($\geq$) : $T(n) = \Omega(f(n))$ if $T(n) \geq c f(n)$, where $n \geq n_0$
• Little-omega ($>$)
Examples

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

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

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

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

Note: $c_1, c_2, n_0$ cannot depend on $n$
\[ T(n) = \Theta(f(n)) \]

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: \( c_1, c_2, n_0 \text{ cannot depend on } n \)

**Examples**

- \( 3^{\log_2 n} - O \)
  - Claim: \( 21n (\log_2(n) + 1) = \Theta(n \log_2 n) \)

\[ c_1 = 22 \]
\[ n_0 = 2 \]
Examples

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

$$c_1 n \lg n \leq 21n \lg n + 21 \quad \forall \ n \geq n_0$$

$c_1 n \lg n \leq 21n \lg n \quad \subseteq \quad 21n \lg n + 21n$

$c_1 n \lg n \leq 21n \lg n$

$c_1 n \lg n \leq 21n \lg n$

$c_1 = 21 \ , \ c_2 = 42 \ , \ n_0 = 2$
$O(f(n))$: $T(n) \leq c_2 f(n)$

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

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

What are good values for:
- $c$
- $n_0$
Insertion Sort vs Merge Sort

Computer A: Insertion Sort
- 10,000 MIPS
- $2n^2$
- 5.5 hours
- 23 days
- 10 million numbers

Computer B: Merge Sort
- 10 MIPS
- $50n \cdot \lg n$
- 20 minutes
- 4 hours
- 100 million numbers
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”