COSC 290 Discrete Structures

Lecture 22: Algorithm Analysis

Prof. Michael Hay
Wednesday, Oct. 25, 2017
Colgate University

Plan for today

1. Properties of Big-Oh
2. Algorithm Analysis
3. Analysis exercises

Properties of Big-Oh

Recall: Big Oh Notation

Let $f : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ and $g : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$.

We say $f(n) = O(g(n))$ when there exists some $n_0 \in \mathbb{R}^{\geq 0}$ and some $c \in \mathbb{R}^{>0}$ such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$.

$$\exists n_0 \in \mathbb{R}^{\geq 0} : \exists c \in \mathbb{R}^{>0} : \forall n \in \mathbb{R}^{\geq 0} : (n \geq n_0) \implies (f(n) \leq c \cdot g(n))$$

A graphical view

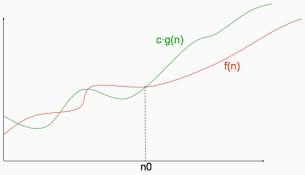

Figure 1: $f(n) = O(g(n))$. For $n \geq n_0$, $f(n)$ grows no faster than $c \cdot g(n)$.

Useful properties

Let ϵ be arbitrary constant.

- (Log slower than polynomial) If $f(n) = \log n$, $f(n) = O(n^\epsilon)$.
- (Polynomial bounded by degree) If $f(n)$ is a degree k polynomial, $f(n) = O(n^k)$.
- (Polynomial slower than exponential)If $f(n) = n^k$, $f(n) = O((1+\epsilon)^n)$.
- (Transitivity) If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$.
- If $f(n) = O(h_1(n))$ and $g(n) = O(h_2(n))$, then
    - (Addition) $f(n) + g(n) = O(h_1(n) + h_2(n))$, and
    - (Multiplication) $f(n) \cdot g(n) = O(h_1(n) \cdot h_2(n))$.
- (Equivalence of addition/max) $f(n) = O(g(n) + h(n)) \iff f(n) = O(\max(g(n), h(n)))$

Exercise

Recall definition of big-Oh: we say $f(n) = O(g(n))$ when there exists some $n_0 \in \mathbb{R}^{\geq 0}$ and some $c \in \mathbb{R}^{>0}$ such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$.

$$\exists n_0 \in \mathbb{R}^{\geq 0} : \exists c \in \mathbb{R}^{>0} : \forall n \in \mathbb{R}^{\geq 0} : (n \geq n_0) \implies (f(n) \leq c \cdot g(n))$$

Exercise: Use this to prove that if $f(n) = O(h_1(n))$ and $g(n) = O(h_2(n))$, then $f(n) \cdot g(n) = O(h_1(n) \cdot h_2(n))$.

Work in small groups. Raise your hand when you have an answer.

Big Omega

We say $f(n) = \Omega(g(n))$ when there exists some $n_0 \in \mathbb{R}^{\geq 0}$ and some $c \in \mathbb{R}^{>0}$ such that for all $n \geq n_0$, $f(n) \geq c \cdot g(n)$.

$$\exists n_0 \in \mathbb{R}^{\geq 0} : \exists c \in \mathbb{R}^{>0} : \forall n \in \mathbb{R}^{\geq 0} : (n \geq n_0) \implies (f(n) \geq c \cdot g(n))$$

Whereas Big Oh is is an (asymptotic) upper bound, big Omega is an (asymptotic) lower bound.

Big Theta and the rest

1. $f(n) = O(g(n))$ says f grows no faster than $g(n)$
2. $f(n) = \Omega(g(n))$ says f grows no slower than $g(n)$
3. $f(n) = \Theta(g(n))$ says f grows at the same rate as $g(n)$
   - In other words, $f(n) = \Omega(g(n))$ and $f(n) = O(g(n))$
4. $f(n) = o(g(n))$ says f grows (strictly) slower faster than $g(n)$
   - In other words, $f(n) = O(g(n))$ but $f(n) \neq \Omega(g(n))$
5. $f(n) = \omega(g(n))$ says f (strictly) faster than $g(n)$
   - In other words, $f(n) = \Omega(g(n))$ but $f(n) \neq O(g(n))$

You are expected to become familiar with this notation, especially first three!

Algorithm Analysis

Analyzing runtime

- Let $f(n)$ be exact runtime on input of size n.
- $f(n)$ is number of "primitive steps" required
- Typically, analyze runtime for *worst-case* inputs. Alternatives: best-case, average-case.
- Aim to identify asymptotic upper bounds $f(n) = O(g(n))$ and asymptotic lower bounds $f(n) = \Omega(g(n))$.

Bubble Sort

```
1: function BubbleSort(A)
2:     for i := 0 to n − 1 do
3:         for j := 0 to n − 1 − (i + 1) do
4:             if A[j] > A[j + 1] then
5:                 Swap A[j] and A[j + 1]
```

Suppose line 4 has cost t_4 and line 5 has cost t_5 (when executed).

If $f(n)$ is the worst-case runtime on array of size n, then let's show $f(n) = O(n^2)$ and $f(n) = \Omega(n^2)$. In other words, $f(n) = \Theta(n^2)$.

Shown on board

Analysis

$$
\begin{aligned}
f(n) &= \sum_{i=0}^{n-1} \sum_{j=0}^{(n-1-(i+1))} c(i,j) && t_4 \le c(i,j) \le t_4 + t_5 \\
&\le (t_4+t_5) \sum_{i=0}^{n-1} \sum_{j=0}^{(n-1-(i+1))} 1 && \text{assume a swap every time} \\
&= (t_4+t_5) \sum_{i=0}^{n-1} ((n-1-(i+1)) - 0 + 1) && \text{because } \sum_{k=a}^{b} 1 = b - a + 1 \\
&= (t_4+t_5) \sum_{i=0}^{n-1} (n-1) - i && \text{simplify} \\
&= (t_4+t_5) \sum_{k=0}^{n-1} k && \text{change of variable, } k = n-1-i \\
&= (t_4+t_5) \frac{(n-1)n}{2} = O(n^2)
\end{aligned}
$$

A similar analysis can show it's $\Omega(n^2)$, and thus $\Theta(n^2)$.

Useful identities

$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ (Example 5.4)

$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ (Exercise 5.1)

$\sum_{i=0}^{n} (n-i) = \sum_{i=0}^{n} i$ (Example 2.15, p.213-214)

And others from Ch. 2.2, Ch. 5.2, 6.3.

Analysis exercises

Poll: Bubble Sort

```
function BubbleSortVariant(A)
    for i := 0 to n − 1 do
        madeSwap := False
        for j := 0 to n − 1 − (i + 1) do
            if A[j] > A[j + 1] then
                Swap A[j] and A[j + 1]
                madeSwap := True
        if madeSwap = False then
            return A
```

Let $f(n)$ denote run-time on a best-case input. Which of the following are true?

1. $f(n) = O(n)$
2. $f(n) = \Omega(n)$
3. $f(n) = O(n^2)$
4. $f(n) = \Omega(n^2)$
5. More than one / None of the above

Poll: worst-case analysis

Input: Arrays A and B, both of size n
Output: True if $\exists i, j : A[i] = B[j]$, False otherwise.

$i = 0$, $found = False$
while $i < n$ and $found = False$ **do**
  $j = 0$
  while $j < n$ **do**
    if $A[i] = B[j]$ **then**
      $found = True$
    $j = j + 1$
  $i = i + 1$
return $found$

Let $f(n)$ denote runtime on worst-case input. Which of the following are true?

1. $f(n) = O(n)$
2. $f(n) = \Omega(n)$
3. $f(n) = O(n^2)$
4. $f(n) = \Omega(n^2)$
5. More than one / None of the above

Poll: best-case analysis

Input: Arrays A and B, both of size n
Output: True if $\exists i, j : A[i] = B[j]$, False otherwise.

$i = 0$, $found = False$
while $i < n$ and $found = False$ **do**
  $j = 0$
  while $j < n$ **do**
    if $A[i] = B[j]$ **then**
      $found = True$
    $j = j + 1$
  $i = i + 1$
return $found$

Let $f(n)$ denote runtime on best-case input. Which of the following are true?

1. $f(n) = O(1)$
2. $f(n) = O(n)$
3. $f(n) = \Omega(n)$
4. $f(n) = \Omega(n^2)$
5. More than one / None of the above

Poll: sort and compare simulation

Input: Arrays A and B, both of size n
Output: Number of elements of A that appear in B

A := sort(A)
B := sort(B)
$i := 0$, $j := 0$, $count := 0$
while $i < n$ and $j < n$ **do**
  print $A[i]$, $B[j]$
  if $A[i] < B[j]$ **then**
    $i := i + 1$
  else if $A[i] > B[j]$ **then**
    $j := j + 1$
  else
    $count := count + 1$
    $j := j + 1$, $j := j + 1$
return $count$

On a piece of paper, simulate execution on $A = [1, 6, 2]$ and $B = [4, 2, 6]$. Which of the following pairs is *not* printed?

1. 1 2
2. 1 4
3. 2 2
4. 6 4
5. 6 6

Poll: sort and compare runtime analysis

Input: Arrays A and B, both of size n
Output: Number of elements of A that appear in B

A := sort(A)
B := sort(B)
$i := 0$, $j := 0$, $count := 0$
while $i < n$ and $j < n$ **do**
  print $A[i]$, $B[j]$
  if $A[i] < B[j]$ **then**
    $i := i + 1$
  else if $A[i] > B[j]$ **then**
    $j := j + 1$
  else
    $count := count + 1$
    $j := j + 1$, $j := j + 1$
return $count$

Assuming that the cost of sorting is $h(n)$, what is the worst-case runtime of this algorithm?

1. $\Theta(n + h(n))$
2. $\Theta(n + 2 \cdot h(n))$
3. $\Theta(n^2 + h(n))$
4. $\Theta(n^2 + 2 \cdot h(n))$
5. More than one / None of above