Data Structures and Algorithms

CS245-2017S-02

Algorithm Analysis

David Galles

Department of Computer Science
University of San Francisco

02-0: Algorithm Analysis

When is algorithm A better than algorithm B?

02-1: Algorithm Analysis

When is algorithm A better than algorithm B?

- Algorithm A runs faster

02-2: Algorithm Analysis

When is algorithm A better than algorithm B?

- Algorithm A runs faster
- Algorithm A requires less space to run

02-3: Algorithm Analysis

When is algorithm A better than algorithm B?

- Algorithm A runs faster
- Algorithm A requires less space to run

Space / Time Trade-off

- Can often create an algorithm that runs faster, by using more space

For now, we will concentrate on time efficiency

Best Case vs. Worst Case

How long does the following function take to run:

```
boolean find(int A[], int element) {
  for (i=0; i<A.length; i++) {
     if (A[i] == elem)
       return true;
  }
  return false;
}
```

02-5: Best Case vs. Worst Case

How long does the following function take to run:

```
boolean find(int A[], int elem) {
  for (i=0; i<A.length; i++) {
     if (A[i] == elem)
        return true;
  }
  return false;
}
```

It depends on if – and where – the element is in the list

02-6: Best Case vs. Worst Case

- Best Case – What is the fastest that the algorithm can run
- Worst Case – What is the slowest that the algorithm can run
- Average Case – How long, on average, does the algorithm take to run

Worst Case performance is almost always important.
Usually, Best Case performance is unimportant (why?)

Usually, Average Case = Worst Case (but not always!)

02-7: Measuring Time Efficiency

How long does an algorithm take to run?

02-8: Measuring Time Efficiency

How long does an algorithm take to run?

- Implement on a computer, time using a stopwatch.

02-9: Measuring Time Efficiency

How long does an algorithm take to run?

- Implement on a computer, time using a stopwatch. Problems:
  - Not just testing algorithm – testing implementation of algorithm
  - Implementation details (cache performance, other programs running in the background, etc) can affect results
  - Hard to compare algorithms that are not tested under *exactly the same conditions*

02-10: Measuring Time Efficiency

How long does an algorithm take to run?

- Implement on a computer, time using a stopwatch. Problems:
  - Not just testing algorithm – testing implementation of algorithm
  - Implementation details (cache performance, other programs running in the background, etc) can affect results
  - Hard to compare algorithms that are not tested under *exactly the same conditions*
- Better Method: Build a mathematical model of the running time, use model to compare algorithms

02-11: Competing Algorithms

- Linear Search

```
for (i=low; i <= high; i++)
  if (A[i] == elem) return true;
return false;
```

- Binary Search

```
int BinarySearch(int low, int high, elem) {
  if (low > high) return false;
  mid = (high + low) / 2;
  if (A[mid] == elem) return true;
  if (A[mid] < elem)
    return BinarySearch(mid+1, high, elem);
  else
    return BinarySearch(low, mid-1, elem);
}
```

02-12: Linear vs Binary

- Linear Search

```
for (i=low; i <= high; i++)
  if (A[i] == elem) return true;
return false;
```

Time Required, for a problem of size n (worst case):

02-13: Linear vs Binary

- Linear Search

```
for (i=low; i <= high; i++)
  if (A[i] == elem) return true;
return false;
```

Time Required, for a problem of size n (worst case):

$c_1 * n$ for some constant c_1

02-14: Linear vs Binary

- Binary Search

```
int BinarySearch(int low, int high, elem) {
  if (low > high) return false;
  mid = (high + low) / 2;
  if (A[mid] == elem) return true;
  if (A[mid] < elem)
    return BinarySearch(mid+1, high, elem);
  else
    return BinarySearch(low, mid-1, elem);
}
```

Time Required, for a problem of size n (worst case):

02-15: Linear vs Binary

- Binary Search

```
int BinarySearch(int low, int high, elem) {
  if (low > high) return false;
  mid = (high + low) / 2;
  if (A[mid] == elem) return true;
  if (A[mid] < elem)
    return BinarySearch(mid+1, high, elem);
  else
    return BinarySearch(low, mid-1, elem);
}
```

Time Required, for a problem of size n (worst case): $c_2 * lg(n)$ for some constant c_2

02-16: Do Constants Matter?

- Linear Search requires time $c_1 * n$, for some c_1
- Binary Search requires time $c_2 * lg(n)$, for some c_2

What if there is a *very* high overhead cost for function calls?

What if c_2 is *1000 times larger* than c_1?

02-17: Constants *Do Not* Matter!

Length of list	Time Required for Linear Search	Time Required for Binary Search
10	0.001 seconds	0.3 seconds
100	0.01 seconds	0.66 seconds
1000	0.1 seconds	1.0 seconds
10000	1 second	1.3 seconds
100000	10 seconds	1.7 seconds
1000000	2 minutes	2.0 seconds
10000000	17 minutes	2.3 seconds
10^{10}	11 days	3.3 seconds
10^{15}	30 centuries	5.0 seconds
10^{20}	300 million years	6.6 seconds

02-18: Growth Rate

We care about the *Growth Rate* of a function – how much more we can do if we add more processing power

Faster Computers $\neq$ Solving Problems Faster
Faster Computers $=$ Solving Larger Problems

- Modeling more variables
- Handling bigger databases
- Pushing more polygons

02-19: Growth Rate Examples

	Size of problem that can be solved					
time	$10n$	$5n$	$n \lg n$	n^2	n^3	2^n
1 s	1000	2000	1003	100	21	13
2 s	2000	4000	1843	141	27	14
20 s	20000	40000	14470	447	58	17
1 m	60000	120000	39311	774	84	19
1 hr	3600000	7200000	1736782	18973	331	25

02-20: Constants and Running Times

- When calculating a formula for the running time of an algorithm:
  - Constants aren't as important as the growth rate of the function
  - Lower order terms don't have much of an impact on the growth rate
    - $x^3 + x$ vs x^3
- We'd like a formal method for describing what is important when analyzing running time, and what is not.

02-21: Big-Oh Notation

$O(f(n))$ is the set of all functions that are bound from above by $f(n)$

$T(n) \in O(f(n))$ if

$$\exists c, n_0 \text{ such that } T(n) \leq c * f(n) \text{ when } n > n_0$$

02-22: Big-Oh Examples

$$
\begin{aligned}
n &\in O(n) \text{ ?} \\
10n &\in O(n) \text{ ?} \\
n &\in O(10n) \text{ ?} \\
n &\in O(n^2) \text{ ?} \\
n^2 &\in O(n) \text{ ?} \\
10n^2 &\in O(n^2) \text{ ?} \\
n \lg n &\in O(n^2) \text{ ?} \\
\ln n &\in O(2n) \text{ ?} \\
\lg n &\in O(n) \text{ ?} \\
3n + 4 &\in O(n) \text{ ?} \\
5n^2 + 10n - 2 &\in O(n^3)\text{? } O(n^2) \text{ ? } O(n) \text{ ?}
\end{aligned}
$$

Big-Oh Examples

$$\begin{array}{rcl}
n & \in & O(n) \\
10n & \in & O(n) \\
n & \in & O(10n) \\
n & \in & O(n^2) \\
n^2 & \notin & O(n) \\
10n^2 & \in & O(n^2) \\
n \lg n & \in & O(n^2) \\
\ln n & \in & O(2n) \\
\lg n & \in & O(n) \\
3n + 4 & \in & O(n) \\
5n^2 + 10n - 2 & \in & O(n^3), \in O(n^2), \notin O(n)\ ?
\end{array}$$

02-24: Big-Oh Examples II

$$
\begin{aligned}
\sqrt{n} &\in O(n) \text{ ?} \\
\lg n &\in O(2^n) \text{ ?} \\
\lg n &\in O(n) \text{ ?} \\
n \lg n &\in O(n) \text{ ?} \\
n \lg n &\in O(n^2) \text{ ?} \\
\sqrt{n} &\in O(\lg n) \text{ ?} \\
\lg n &\in O(\sqrt{n}) \text{ ?} \\
n \lg n &\in O(n^{\frac{3}{2}}) \text{ ?} \\
n^3 + n \lg n + n\sqrt{n} &\in O(n \lg n) \text{ ?} \\
n^3 + n \lg n + n\sqrt{n} &\in O(n^3) \text{ ?} \\
n^3 + n \lg n + n\sqrt{n} &\in O(n^4) \text{ ?}
\end{aligned}
$$

02-25: Big-Oh Examples II

$$
\begin{aligned}
\sqrt{n} &\in O(n) \\
\lg n &\in O(2^n) \\
\lg n &\in O(n) \\
n \lg n &\notin O(n) \\
n \lg n &\in O(n^2) \\
\sqrt{n} &\notin O(\lg n) \\
\lg n &\in O(\sqrt{n}) \\
n \lg n &\in O(n^{\frac{3}{2}}) \\
n^3 + n \lg n + n\sqrt{n} &\notin O(n \lg n) \\
n^3 + n \lg n + n\sqrt{n} &\in O(n^3) \\
n^3 + n \lg n + n\sqrt{n} &\in O(n^4)
\end{aligned}
$$

02-26: Big-Oh Examples III

$$f(n) = \begin{cases} n & \text{for } n \text{ odd} \\ n^3 & \text{for } n \text{ even} \end{cases}$$

$$g(n) = n^2$$

$f(n) \in O(g(n))$?

$g(n) \in O(f(n))$?

$n \in O(f(n))$?

$f(n) \in O(n^3)$?

02-27: Big-Oh Examples III

$$f(n) = \begin{cases} n & \text{for } n \text{ odd} \\ n^3 & \text{for } n \text{ even} \end{cases}$$

$$g(n) = n^2$$

$$\begin{aligned} f(n) &\not\in O(g(n)) \\ g(n) &\not\in O(f(n)) \\ n &\in O(f(n)) \\ f(n) &\in O(n^3) \end{aligned}$$

02-28: Big-Ω Notation

$\Omega(f(n))$ is the set of all functions that are bound from *below* by $f(n)$

$T(n) \in \Omega(f(n))$ if

$\exists c, n_0$ such that $T(n) \geq c * f(n)$ when $n > n_0$

02-29: Big-Ω Notation

$\Omega(f(n))$ is the set of all functions that are bound from *below* by $f(n)$

$T(n) \in \Omega(f(n))$ if

$\exists c, n_0$ such that $T(n) \geq c * f(n)$ when $n > n_0$

$$f(n) \in O(g(n)) \Rightarrow g(n) \in \Omega(f(n))$$

02-30: Big-Θ Notation

$\Theta(f(n))$ is the set of all functions that are bound *both* above *and* below by $f(n)$. Θ is a *tight bound*

$T(n) \in \Theta(f(n))$ if

$$T(n) \in O(f(n)) \text{ and } T(n) \in \Omega(f(n))$$

02-31: Big-Oh Rules

1. If $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$, then $f(n) \in O(h(n))$
2. If $f(n) \in O(kg(n)$ for any constant $k > 0$, then $f(n) \in O(g(n))$
3. If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$, then $f_1(n) + f_2(n) \in O(\max(g_1(n), g_2(n)))$
4. If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$, then $f_1(n) * f_2(n) \in O(g_1(n) * g_2(n))$

(Also work for Ω, and hence Θ)

02-32: Big-Oh Guidelines

- Don't include constants/low order terms in Big-Oh
- Simple statements: $\Theta(1)$
- Loops: Θ(inside) * # of iterations
  - Nested loops work the same way
- Consecutive statements: Longest Statement
- Conditional (if) statements:
  O(Test + longest branch)

02-33: Calculating Big-Oh

```
for (i=1; i<n; i++)
   sum++;
```

02-34: Calculating Big-Oh

```
for (i=1; i<n; i++)          Executed n times
  sum++;                      O(1)
```

Running time: $O(n), \Omega(n), \Theta(n)$

02-35: Calculating Big-Oh

```
for (i=1; i<n; i=i+2)
  sum++;
```

02-36: Calculating Big-Oh

```
for (i=1; i<n; i=i+2)          Executed n/2 times
  sum++;                        O(1)
```

Running time: $O(n), \Omega(n), \Theta(n)$

02-37: Calculating Big-Oh

```
for (i=1; i<n; i++)
  for (j=1; j < n/2; j++)
    sum++;
```

02-38: Calculating Big-Oh

```
for (i=1; i<n; i++)               Executed n times
  for (j=1; j < n/2; j++)          Executed n/2 times
    sum++;                           O(1)
```

Running time: $O(n^2), \Omega(n^2), \Theta(n^2)$

02-39: Calculating Big-Oh

```
for (i=1; i<n; i=i*2)
   sum++;
```

02-40: Calculating Big-Oh

```
for (i=1; i<n; i=i*2)    Executed lg n times
  sum++;                  O(1)
```

Running Time: $O(\lg n), \Omega(\lg n), \Theta(\lg n)$

02-41: Calculating Big-Oh

```
for (i=0; i<n; i++)
  for (j = 0; j<i; j++)
    sum++;
```

Calculating Big-Oh

```
for (i=0; i<n; i++)          Executed n times
  for (j = 0; j<i; j++)       Executed <= n times
    sum++;                     O(1)
```

Running Time: $O(n^2)$. Also $\Omega(n^2)$?

02-43: Calculating Big-Oh

```
for (i=0; i<n; i++)
  for (j = 0; j<i; j++)
    sum++;
```

Exact # of times `sum++` is executed:

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

$$\in \Theta(n^2)$$

02-44: Calculating Big-Oh

```
sum = 0;
for (i=0; i<n; i++)
  sum++;
for (i=1; i<n; i=i*2)
  sum++;
```

02-45: Calculating Big-Oh

```
sum = 0;                        O(1)
for (i=0; i<n; i++)             Executed n times
  sum++;                         O(1)
for (i=1; i<n; i=i*2)           Executed lg n times
  sum++;                         O(1)
```

Running Time: $O(n), \Omega(n), \Theta(n)$

02-46: Calculating Big-Oh

```
sum = 0;
for (i=0; i<n; i=i+2)
  sum++;
for (i=0; i<n/2; i=i+5)
  sum++;
```

Calculating Big-Oh

```
sum = 0;                          O(1)
for (i=0; i<n; i=i+2)             Executed n/2 times
  sum++;                           O(1)
for (i=0; i<n/2; i=i+5)           Executed n/10 times
  sum++;                           O(1)
```

Running Time: $O(n), \Omega(n), \Theta(n)$

Calculating Big-Oh

```
for (i=0; i<n;i++)
  for (j=1; j<n; j=j*2)
    for (k=1; k<n; k=k+2)
      sum++;
```

02-49: Calculating Big-Oh

```
for (i=0; i<n;i++)                 Executed n times
  for (j=1; j<n; j=j*2)             Executed lg n times
    for (k=1; k<n; k=k+2)            Executed n/2 times
      sum++;                          O(1)
```

Running Time: $O(n^2 \lg n), \Omega(n^2 \lg n), \Theta(n^2 \lg n)$

02-50: Calculating Big-Oh

```
sum = 0;
for (i=1; i<n; i=i*2)
  for (j=0; j<n; j++)
    sum++;
```

02-51: Calculating Big-Oh

```
sum = 0;                          O(1)
for (i=1; i<n; i=i*2)             Executed lg n times
  for (j=0; j<n; j++)              Executed n times
    sum++;                          O(1)
```

Running Time: $O(n \lg n), \Omega(n \lg n), \Theta(n \lg n)$

02-52: Calculating Big-Oh

```
sum = 0;
for (i=1; i<n; i=i*2)
  for (j=0; j<i; j++)
    sum++;
```

Calculating Big-Oh

```
sum = 0;                              O(1)
for (i=1; i<n; i=i*2)                 Executed lg n times
  for (j=0; j<i; j++)                  Executed <= n times
    sum++;                              O(1)
```

Running Time: $O(n \lg n)$. Also $\Omega(n \lg n)$?

02-54: Calculating Big-Oh

```
sum = 0;
for (i=1; i<n; i=i*2)
  for (j=0; j<i; j++)
    sum++;
```

# of times `sum++` is executed:

$$\begin{aligned} \sum_{i=0}^{\lg n} 2^i &= 2^{\lg n + 1} - 1 \\ &= 2n - 1 \\ &\in \Theta(n) \end{aligned}$$

02-55: Calculating Big-Oh

Of course, a little change can mess things up a bit ...

```
sum = 0;
for (i=1; i<=n; i=i+1)
  for (j=1; j<=i; j=j*2)
    sum++;
```

02-56: Calculating Big-Oh

Of course, a little change can mess things up a bit ...

```
sum = 0;
for (i=1; i<=n; i=i+1)          Executed n times
  for (j=1; j<=i; j=j*2)        Executed <= lg n times
    sum++;                      O(1)
```

So, this is code is $O(n \lg n)$ – but is it also $\Omega(n \lg n)$?

02-57: Calculating Big-Oh

Of course, a little change can mess things up a bit ...

```
sum = 0;
for (i=1; i<=n; i=i+1)        Executed n times
  for (j=1; j<=i; j=j*2)      Executed <= lg n times
    sum++;                    O(1)
```

Total time `sum++` is executed:

$$\sum_{i=1}^{n} \lg i$$

This can be tricky to evaluate, but we only need a bound ...

02-58: Calculating Big-Oh

Total # of times `sum++` is executed:

$$
\begin{aligned}
\sum_{i=1}^{n} \lg i &= \sum_{i=1}^{n/2-1} \lg i + \sum_{i=n/2}^{n} \lg i \\
&\geq \sum_{i=n/2}^{n} \lg i \\
&\geq \sum_{i=n/2}^{n} \lg n/2 \\
&= n/2 \lg n/2 \\
&\in \Omega(n \lg n)
\end{aligned}
$$