Lecture 3: How to measure efficiency
Announcements

- Course background survey due by Friday
- HW 1 is Due Friday
- Alex has Office Hours after class (2:30-4:30) CSE 006, will help with setup
  - If you have any questions about your setup please come to office hours so we can iron out all the wrinkles before the partnered projects begin next week.

- HW 2 Assigned on Friday – Partner selection forms due by 11:59pm Thursday

https://goo.gl/forms/rVrVUkFDdsqI8pkD2
**Review: Sequential Search**

**sequential search**: Locates a target value in an array / list by examining each element from start to finish.
- How many elements will it need to examine?
- Example: Searching the array below for the value 42:

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>-4</td>
<td>2</td>
<td>7</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>22</td>
<td>25</td>
<td>30</td>
<td>36</td>
<td><strong>42</strong></td>
<td>50</td>
<td>56</td>
<td>68</td>
<td>85</td>
<td>92</td>
<td>103</td>
</tr>
</tbody>
</table>

- What is the best case? \( O(1) \)
- What is the worst case? \( O(n) \)
- What is the complexity class? \( O(n) \)
**Review: Binary Search**

**binary search:** Locates a target value in a *sorted* array or list by successively eliminating half of the array from consideration.

- How many elements will it need to examine?
- Example: Searching the array below for the value 42:

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>-4</td>
<td>2</td>
<td>7</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>22</td>
<td>25</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>50</td>
<td>56</td>
<td>68</td>
<td>85</td>
<td>92</td>
<td>103</td>
</tr>
<tr>
<td>min</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>103</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- What is the best case? \( O(1) \) - the middle
- What is the worst case? \( \log(n) \)
- What is the complexity class?
Analyzing Binary Search

What is the pattern?
- At each iteration, we eliminate half of the remaining elements

How long does it take to finish?
- 1st iteration – $N/2$ elements remain
- 2nd iteration – $N/4$ elements remain
- Kth iteration - $N/2^k$ elements remain
- Done when $N/2^k = 1
Analyzing Binary Search

\[ \frac{N}{2^K} = 1 \]
\[ N = 2^K \]
\[ \log_2 N = \log_2 2^K \]
\[ \log_2 N = K \]

Logarithms
\[ \log_b a = x \quad \text{mean} \]
\[ x \quad \text{solves} \]
\[ b^x = a \]
\[ b^z = x \]
\[ b^z \rightarrow z \]
Analyzing Binary Search

Finishes when $N / 2^k = 1$

$N / 2^k = 1$

--> multiply right side by $2^k$

$N = 2^k$

--> isolate $K$ exponent with logarithm

$\log_2 N = k$

Is this exact?
- $N$ can be things other than powers of $2$
- If $N$ is odd we can’t technically use $\log_2$
- When we have an odd number of elements we select the larger half
- Within a fair rounding error

CSE 373 SP 18 - KASEY CHAMPION 7
Asymptotic Analysis

**asymptotic analysis**: how the runtime of an algorithm grows as the data set grows

**Approximations / Rules**
- Basic operations take “constant” time
  - Assigning a variable
  - Accessing a field or array index
- Consecutive statements
  - Sum of time for each statement
- Function calls
  - Time of function’s body
- Conditionals
  - Time of condition + maximum time of branch code
- Loops
  - Number of iterations x time for loop body
Modeling Case Study

**Goal:** return ‘true’ if a sorted array of ints contains duplicates

**Solution 1:** compare each pair of elements

```java
public boolean hasDuplicate1(int[] array) {
    for (int i = 0; i < array.length; i++) {
        for (int j = 0; j < array.length; j++) {
            if (i != j && array[i] == array[j]) {
                return true;
            }
        }
    }
    return false;
}
```

**Solution 2:** compare each consecutive pair of elements

```java
public boolean hasDuplicate2(int[] array) {
    for (int i = 0; i < array.length - 1; i++) {
        if (array[i] == array[i + 1]) {
            return true;
        }
    }
    return false;
}
```
Modeling Case Study: Solution 2

\[ T(n) \text{ where } n = \text{array.length} \]

\[ \rightarrow \text{ work inside out} \]

**Solution 2:** compare each consecutive pair of elements

```java
public boolean hasDuplicate2(int[] array) {
    for (int i = 0; i < array.length - 1; i++) {
        if (array[i] == array[i + 1]) {
            return true; \( +1 \)
        }
    }
    return false; \( +1 \)
}
```

\[ T(n) = 5 (n-1) + 1 \]

linear time complexity class \( O(n) \)
Modeling Case Study: Solution 1

Solution 1: compare each consecutive pair of elements

```java
public boolean hasDuplicate1(int[] array) {
    for (int i = 0; i < array.length; i++) {
        for (int j = 0; j < array.length; j++) {
            if (i != j && array[i] == array[j]) {
                return true; +1
            }
        }
    }
    return false; +1
}
```

T(n) = 6 n^2 + 1

quadratic time complexity class \(O(n^2)\)
Comparing Functions
Function growth

$n$ and $4n$ look very different up close
$n^2$ eventually dominates $n$

$n$ and $4n$ look the same over time

$n^2$ doesn’t start off dominating the linear functions
It eventually takes over…

When do we care?
Let’s do it right on small data.
Function comparison: exercise

\[ \Theta(n) = \Theta(5n + 3) \quad \alpha(n) \leq \Theta(5n + 3) \]

\[ f(n) = n \leq g(n) = 5n + 3? \quad \text{True} \quad – \text{all linear functions are treated as equivalent} \]

\[ f(n) = 5n + 3 \leq g(n) = n? \quad \text{True} \]

\[ f(n) = 5n + 3 \leq g(n) = 1? \quad \text{False} \]

\[ f(n) = 5n + 3 \leq g(n) = n^2? \quad \text{True} \quad – \text{quadratic will always dominate linear} \]

\[ f(n) = n^2 + 3n + 2 \leq g(n) = n^3? \quad \text{True} \]

\[ f(n) = n^3 \leq g(n) = n^2 + 3n + 2? \quad \text{False} \]
**Definition:** function domination

A function $f(n)$ is **dominated** by $g(n)$ when...

There exists two constants $c > 0$ and $n_0 > 0$

Such that for all values of $n \geq n_0$

$$f(n) \leq c \times g(n)$$

**Example:**

Is $f(n) = n$ dominated by $g(n) = 5n + 3$ ?

$c = 1$

$n_0 = 1$

Yes!

---

The diagram illustrates the concept of domination with a graph comparing $f(n)$ and $g(n)$, where $i.e. g(n)$ is eventually always bigger than $f(n)$, up to a constant multiple.

For the example:

- $c \times g(n)$
- $n > n_0$
- $c = 1$
- $n_0 = 1$
- $f(n) = n$
- $g(n) = n + 3$
- $c \times g(n) = n + 3 \geq n$ for $n > n_0 = 1$
- $c = 2$
- $f(n) = 2n \geq n + 3$ for $n > 10$
Exercise: Function Domination

Demonstrate that $5n^2 + 3n + 6$ is dominated by $n^3$ by finding a $c$ and $n_0$ that satisfy the definition of domination

$$5n^2 + 3n + 6 \leq 5n^2 + 3n^2 + 6n^2 \text{ when } n \geq 1$$

$$5n^2 + 3n^2 + 6n^2 = 14n^2$$

$$5n^2 + 3n + 6 \leq 14n^2 \text{ for } n \geq 1$$

$$14n^2 \leq c\cdot n^3 \text{ for } c = ? \text{ and } n \geq ?$$

$$\frac{14}{n} \rightarrow c = 14 \text{ and } n \geq 1$$
Definition: Big O

If \( f(n) = n \leq g(n) = 5n + 3 \leq h(n) = 100n \) and
\( h(n) = 100n \leq g(n) = 5n + 3 \leq f(n) \)

Really they are all the “same”

**Definition: Big O**

\( O(f(n)) \) is the “family” or “set” of all functions that are dominated by \( f(n) \)

Question: are \( O(n) \), \( O(5n + 3) \) and \( O(100n) \) all the same?

True! By convention we pick simplest of the above -> \( O(n) \) ie “linear”

\[ f(n) = 1 \text{ in } O(n) \]
Definitions: Big Ω

“f(n) is greater than or equal to g(n)”

F(n) dominates g(n) when:
There exists two constants such that c > 0 and n0 > 0
Such that for all values n ≥ n0
F(n) ≥ c * g(n) is true

Definition: Big Ω

Ω(f(n)) is the family of all functions that dominates f(n)
f(n) is dominated by g(n)

Is that the same as

“f(n) is contained inside O(g(n))”

Yes!

f(n) ∈ g(n)
Examples

$4n^2 \in \Omega(1)$
true

$4n^2 \in \Omega(n)$
true

$4n^2 \in \Omega(n^2)$
true

$4n^2 \in \Omega(n^3)$
false

$4n^2 \in \Omega(n^4)$
false

$4n^2 \in O(1)$
false

$4n^2 \in O(n)$
false

$4n^2 \in O(n^2)$
true

$4n^2 \in O(n^3)$
true

$4n^2 \in O(n^4)$
true

**Definition: Big O**

$O(f(n))$ is the “family” or “set” of all functions that are dominated by $f(n)$

**Definition: Big Omega**

$\Omega(f(n))$ is the family of all functions that dominates $f(n)$
**Definitions: Big \( \Theta \)**

We say \( f(n) \in \Theta(g(n)) \) when both
\( f(n) \in O(g(n)) \) and \( f(n) \in \Omega(g(n)) \) are true

Which is only when \( f(n) = g(n) \)

\( \Theta(f(n)) \) is the family of functions that are equivalent to \( f(n) \)

Industry uses “Big \( \Theta \)” and “Big O” interchangeably
Summary

\[ O(f(n)) \leq f(n) = \Theta(f(n)) \leq \Omega(f(n)) \]

\( f(n) \)

<table>
<thead>
<tr>
<th>Dominated by</th>
<th>Dominates</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(n) \in O(g(n)) )</td>
<td>( f(n) \in \Omega(g(n)) )</td>
</tr>
</tbody>
</table>

- \( O(1) \)
- \( O(\log n) \)
- \( O(n) \)
- \( O(n^2) \)
- \( O(n^3) \)
Justifying the “Rules”

Approximations / Rules
- Basic operations take “constant” time
  - Assigning a variable
  - Accessing a field or array index
- Consecutive statements
  - Sum of time for each statement
- Function calls
  - Time of function’s body
- Conditionals
  - Time of condition + maximum time of branch code
- Loops
  - Number of iterations x time for loop body
public void mystery(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n * n; j++) {
            System.out.println("Hello");
        }
        for (int j = 0; j < 10; j++) {
            System.out.println("world");
        }
    }
}

*Remember:* work outside in

**Solution:** $T(n) = n(n^2 + 10) = n^3 + 10n$
Modeling Complex Loops

```java
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println("Hello!");
    }
}
```

Keep an eye on loop bounds!
Modeling Complex Loops

for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println("Hello!");
    }
}

Summation
1 + 2 + 3 + 4 + ... + n = \sum_{i=1}^{n} i

Definition: Summation
\sum_{i=a}^{b} f(i) = f(a) + f(a + 1) + f(a + 2) + ... + f(b-2) + f(b-1) + f(b)

T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} c
Simplifying Summations

for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println("Hello!");
    }
}

\[
T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} c = \sum_{i=0}^{n-1} ci \quad \text{Summation of a constant}
\]

\[
= c \sum_{i=0}^{n-1} i \quad \text{Factoring out a constant}
\]

\[
= c \frac{n(n-1)}{2} \quad \text{Gauss’s Identity}
\]

\[
= \frac{c}{2} n^2 - \frac{c}{2} n \quad O(n^2)
\]
Function Modeling: Recursion

```java
public int factorial(int n) {
    if (n == 0 || n == 1) {  
        return 1;  
    } else {  
        return n * factorial(n - 1);  
    }
}
```
Function Modeling: Recursion

```java
public int factorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

\[ T(n) = \begin{cases} 
  C_1 & \text{when } n = 0 \text{ or } 1 \\
  C_2 + T(n-1) & \text{otherwise} 
\end{cases} \]

Definition: Recurrence

Mathematical equivalent of an if/else statement

\( f(n) = \)
Unfolding Method

\[ T(n) = \begin{cases} 
C_1 & \text{when } n = 0 \text{ or } 1 \\
C_2 + T(n-1) & \text{otherwise}
\end{cases} \]

\[ T(3) = C_2 + T(3-1) = C_2 + (C_2 + T(2-1)) = C_2 + (C_2 + (C_1)) = 2C_2 + C_1 \]

\[ T(n) = C_1 + \sum_{i=0}^{n-1} C_2 \]

Summation of a constant

\[ T(n) = C_1 + (n-1)C_2 \]