Performance and asymptotics

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Instructor</th>
<th>Days, Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lecture A</td>
<td>Tiefenbruck</td>
<td>Tu, Th 11am-12:20pm</td>
<td>Center 119</td>
</tr>
<tr>
<td>Lecture B</td>
<td>Tiefenbruck</td>
<td>Tu, Th 9:30am-10:50am</td>
<td>Center 119</td>
</tr>
<tr>
<td>Lecture C</td>
<td>Minnes</td>
<td>Tu, Th 3:30pm-4:50pm</td>
<td>WLH 2005</td>
</tr>
</tbody>
</table>

http://cseweb.ucsd.edu/classes/fa15/cse21-abc/

Oct 1, 2015
1) What problem are we solving?   SPECIFICATION
2) How do we solve the problem?  ALGORITHM DESCRIPTION
3) Why do these steps solve the problem?  CORRECTNESS
4) When do we get an answer?    RUNNING TIME PERFORMANCE
Counting comparisons: WHEN

Measure …

Time

Number of operations

For selection sort (MinSort), how many times do we have to compare the values of some pair of list elements?
Selection Sort (MinSort) Pseudocode

Rosen page 203, exercises 41-42

procedure selection sort(a₁, a₂, ..., aₙ: real numbers with n >=2 )
for i := 1 to n-1
    m := i
    for j:= i+1 to n
        if (aⱼ < aᵢ) then m := j
    interchange aᵢ and aₘ

{ a₁, ..., aₙ is in increasing order}

For each value of i, compare (n-i) pairs of elements.

Sum of positive integers up to (n-1)

\[ (n-1) + (n-2) + \ldots + (1) = \frac{n(n-1)}{2} \]
Counting operations

**When** do we get an answer? **RUNNING TIME PERFORMANCE**

Counting number of times list elements are compared
Runtime performance

**Algorithm**: problem solving strategy as a sequence of steps

**Examples of steps**
- Comparing list elements (which is larger?)
- Accessing a position in a list (probe for value)
- Arithmetic operation (+, -, *, ...)
- etc.

"Single step" depends on context
How long does a "single step" take?

Some factors
- Hardware
- Software

Discuss & list the factors that could impact how long a single step takes
How long does a "single step" take?

Some factors
- Hardware (CPU, climate, cache …)
- Software (programming language, compiler)
The time our program takes will depend on

- Number of steps the algorithm requires
- Time for each of these steps on our system
- Input (size and ???)
Estimate time as a function of the size of the input, $n$
Rate of growth

Ignore what we can't control

Focus on how time scales for large inputs

Which of these functions have the "same" rate of growth?

A. All of them
B. $2^n$ and $n^2$
C. $n^2$ and $3n^2$
D. They're all different
Focus on how time scales for large inputs

For functions $f(n) : \mathbb{N} \to \mathbb{R}, g(n) : \mathbb{N} \to \mathbb{R}$ we say

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

to mean there are constants, C and k such that $|f(n)| \leq C|g(n)|$ for all $n > k$. 

Rosen p. 205
Focus on how time scales for large inputs

Ignore what we can't control

For functions \( f(n) : \mathbb{N} \rightarrow \mathbb{R}, g(n) : \mathbb{N} \rightarrow \mathbb{R} \) we say

\[
f(n) \in O(g(n))
\]

to mean there are constants, \( C \) and \( k \) such that

\[
|f(n)| \leq C|g(n)| \quad \text{for all } n > k.
\]

Rosen p. 205
Definition of Big O

For functions \( f(n) : \mathbb{N} \rightarrow \mathbb{R}, g(n) : \mathbb{N} \rightarrow \mathbb{R} \) we say

\[
f(n) \in O(g(n))
\]

for all \( n > k \).

Example:

\[
f(n) = 3n^2 + 2n \quad g(n) = n^2
\]

What constants can we use to prove that \( f(n) \in O(g(n)) \)?

A. \( C = 1/3, k = 2 \)
B. \( C = 5, k = 1 \)
C. \( C = 10, k = 2 \)
D. None: \( f(n) \) isn't big O of \( g(n) \).
"f(n) is big O of g(n)"

\[ f(n) \in O(g(n)) \]

What functions are in the family \( O(n^2) \)?
"f(n) is big O of g(n)"

\[ f(n) \in O(g(n)) \]

- The value of \( f(n) \) might always be bigger than the value of \( g(n) \).
- \( O(g(n)) \) contains functions that grow \textit{strictly slower} than \( g(n) \).
Is \( f(n) \) big \( O \) of \( g(n) \) ? i.e. is \( f(n) \in O(g(n)) \) ?

**Approach 1:** Look for constants \( C \) and \( k \).

**Approach 2:** Use properties

- **Domination**  
  If \( f(n) \leq g(n) \) for all \( n \) then \( f(n) \) is big-O of \( g(n) \).

- **Transitivity**  
  If \( f(n) \) is big-O of \( g(n) \), and \( g(n) \) is big-O of \( h(n) \), then \( f(n) \) is big-O of \( h(n) \).

- **Additivity/Multiplicativity**  
  If \( f(n) \) is big-O of \( g(n) \), and if \( h(n) \) is nonnegative, then \( f(n) \times h(n) \) is big-O of \( g(n) \times h(n) \) … where \( \times \) is either addition or multiplication.

- **Sum is maximum**  
  \( f(n) + g(n) \) is big-O of the max(\( f(n) \), \( g(n) \))

- **Ignoring constants**  
  For any constant \( c \), \( cf(n) \) is big-O of \( f(n) \)

Rosen p. 210-213
Big O : How to compute?

Is f(n) big O of g(n) ? i.e. is \( f(n) \in O(g(n)) \) ?

**Approach 1:** Look for constants C and k.

**Approach 2:** Use properties

- **Domination**: If \( f(n) \leq g(n) \) for all \( n \), then \( f(n) \) is big-O of \( g(n) \).
- **Transitivity**: If \( f(n) \) is big-O of \( g(n) \), and \( g(n) \) is big-O of \( h(n) \), then \( f(n) \) is big-O of \( h(n) \).
- **Additivity/Multiplicativity**: If \( f(n) \) is big-O of \( g(n) \), and \( h(n) \) is nonnegative, then \( f(n) \cdot h(n) \) is big-O of \( g(n) \cdot h(n) \) where \( * \) is either addition or multiplication.
- **Sum is maximum**: \( f(n) + g(n) \) is big-O of the max(\( f(n) \), \( g(n) \)).
- **Ignoring constants**: For any constant \( c \), \( cf(n) \) is big-O of \( f(n) \).

Rosen p. 210-213
Is \( f(n) \) big \( O \) of \( g(n) \)? i.e. is \( f(n) \in O(g(n)) \)?

**Approach 3.** The limit method. Consider the limit

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)}.
\]

I. If this limit exists and is 0: \( f(n) \) grows strictly slower than \( g(n) \).

II. If this limit exists and is a constant \( c > 0 \): \( f(n), g(n) \), grow at the same rate.

III. If the limit tends to infinity: \( f(n) \) grows strictly faster than \( g(n) \).

IV. If the limit doesn't exist for a different reason … use another approach!

In which cases can we conclude \( f(n) \in O(g(n)) \)?

A. I, II, III
B. I, III
C. I, II
D. None of the above
Other asymptotic classes

\[ f(n) \in O(g(n)) \]

means there are constants, \( C \) and \( k \) such that \( |f(n)| \leq C|g(n)| \) for all \( n > k \).

\[ f(n) \in \Omega(g(n)) \]

means \( g(n) \in O(f(n)) \)

\[ f(n) \in \Theta(g(n)) \]

means \( f(n) \in O(g(n)) \) and \( g(n) \in O(f(n)) \)

What functions are in the family \( \Theta(n^2) \)?
Number of comparisons of list elements

\[(n-1) + (n-2) + \ldots + (1) = \frac{n(n-1)}{2}\]

Rewrite this formula in order notation:

A. \(O(n)\)
B. \(O(n(n-1))\)
C. \(O(n^2)\)
D. \(O(1/2)\)
E. None of the above
Linear Search: HOW

Starting at the beginning of the list, compare items one by one with \( x \) until find it or reach the end.

\[
\text{procedure linear search (\( x: \text{ integer}, a_1, a_2, \ldots, a_n: \text{ distinct integers} \))} \\
i := 1 \\
\text{while } (i <= n \text{ and } x \neq a_i) \\
\hspace{1em} i := i+1 \\
\text{if } i <=n \text{ then location := } i \\
\text{else location := 0} \\
\text{return location} \\
\{ \text{location is the subscript of the term that equals } x, \text{ or} \\
\text{is 0 if } x \text{ is not found } \} 
\]
The time it takes to find $x$ (or determine it is not present) depends on the number of probes, that is the number of list entries we have to retrieve and compare to $x$.

Rosen page 220, part of example 2

How many probes do we make when doing Linear Search on a list of size $n$

- if $x$ happens to equal the **first** element in the list?
- if $x$ happens to equal the **last** element in the list?
- if $x$ happens to equal an element somewhere in the **middle** of the list?
- if $x$ **doesn't equal any** element in the list?
How fast is Linear Search: WHEN

Best case: 1 probe  target appears first

Worst case: n probes  target appears last or not at all

Average case: n/2 probes  target appears in the middle, (expect to have to search about half of the array ... more on expected value later in the course)

Running time depends on more than size of the input!

Rosen p. 220
procedure binary search (x: integer, a_1, a_2, ..., a_n: increasing integers )
i := 1
j := n
while i<j
    m := floor( (i+j)/2 )
    if x > a_m then i := m+1
    else j := m
if x=a_i then location := i
else location := 0
return location

{ location is the subscript of the term that equals x, or is 0 if x is not found }
How fast is Binary Search: WHEN

Rosen page 220, example 3

Number of comparisons (probes) depends on number of iterations of loop, hence size of set.

<table>
<thead>
<tr>
<th>After … iterations of loop</th>
<th>(Max) size of list &quot;in play&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>n</td>
</tr>
<tr>
<td>1</td>
<td>n/2</td>
</tr>
<tr>
<td>2</td>
<td>n/4</td>
</tr>
<tr>
<td>3</td>
<td>n/8</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>??</td>
<td>1</td>
</tr>
</tbody>
</table>
Number of comparisons (probes) depends on number of iterations of loop, hence size of set.

<table>
<thead>
<tr>
<th>After … iterations of loop</th>
<th>(Max) size of list &quot;in play&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>n</td>
</tr>
<tr>
<td>1</td>
<td>n/2</td>
</tr>
<tr>
<td>2</td>
<td>n/4</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>ceil(log₂ n)</td>
</tr>
</tbody>
</table>

Rewrite this formula in order notation:

A. \( \Theta(\log n) \)
B. \( \Theta(\log n + 1) \)
C. \( \Theta(n) \)
D. \( \Theta(\log_{10} n) \)
E. None of the above
## Comparing linear search and binary search

Rosen pages 220-221

### Assumptions

<table>
<thead>
<tr>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assumptions</td>
<td>None</td>
</tr>
</tbody>
</table>

### # probes in

<table>
<thead>
<tr>
<th></th>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>* best case</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1) \text{or} \Theta(\log n)$</td>
</tr>
<tr>
<td>* worst case</td>
<td>$\Theta(n)$</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>* average case</td>
<td>$\Theta(n)$</td>
<td>??</td>
</tr>
</tbody>
</table>

Best case analysis depends on whether we check if midpoint agrees with target right away or wait until list size gets to 1
Comparing linear search and binary search

Rosen pages 220-221

<table>
<thead>
<tr>
<th>Assumptions</th>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Θ(1)</td>
<td>Θ(1) or Θ(log n)</td>
</tr>
<tr>
<td>Sorted list</td>
<td>Θ(n)</td>
<td>Θ(log n)</td>
</tr>
</tbody>
</table>

* best case

* worst case

* average case

??
Comparing linear search and binary search

<table>
<thead>
<tr>
<th></th>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assumptions</td>
<td>None</td>
<td>Sorted list</td>
</tr>
<tr>
<td># probes in</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* best case</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$ or $\Theta(\log n)$</td>
</tr>
<tr>
<td>* worst case</td>
<td>$\Theta(n)$</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>* average case</td>
<td>$\Theta(n)$</td>
<td>??</td>
</tr>
</tbody>
</table>

Is it worth it to sort our list first?
Computing runtime performance of more complicated code and algorithms

- Consecutive code segments
- Nested code segments
Reminders

HW 2 due (tomorrow) **Friday 11:59pm.**

Assignment submission through [Gradescope](#) (see Piazza).