# Performance and Asymptotics

<table>
<thead>
<tr>
<th>Lecture A</th>
<th>Tiefenbruck</th>
<th>MWF 9-9:50am</th>
<th>Center 212</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lecture B</td>
<td>Jones</td>
<td>MWF 2-2:50pm</td>
<td>Center 214</td>
</tr>
<tr>
<td>Lecture C</td>
<td>Tiefenbruck</td>
<td>MWF 11-11:50am</td>
<td>Center 212</td>
</tr>
</tbody>
</table>

[http://cseweb.ucsd.edu/classes/wi16/cse21-abc/](http://cseweb.ucsd.edu/classes/wi16/cse21-abc/)

January 11, 2016
General questions to ask about algorithms

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\( (a_1, a_2, \ldots, a_n: \text{ real numbers with } n \geq 2) \)

for \( i := 1 \) to \( n-1 \)

\[ m := i \]

for \( j := i+1 \) to \( n \)

\[ \text{if } (a_j < a_m) \text{ then } m := j \]

interchange \( a_i \) and \( a_m \)

\{ \( a_1, \ldots, a_n \) is in increasing order\}

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

\( (n-1) + (n-2) + \ldots + 1 \)

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

Sum of positive integers up to \( (n-1) \)
Counting operations

When do we get an answer?  

RUNNING TIME PERFORMANCE

Counting number of times list elements are compared
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
Runtime performance

How long does a "single step" take?

Some factors
- Hardware
- Software

Discuss & list the factors that could impact how long a single step takes
Runtime performance

How long does a "single step" take?

Some factors
- Hardware (CPU, climate, cache …)
- Software (programming language, compiler)
Runtime performance

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 ???)
Runtime performance

Goal:

Estimate time as a function of the size of the input, n

Ignore what we can't control

Focus on how time scales for large inputs
Focus on how time scales for large inputs.

Rate of growth

Ignore what we can't control

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
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 $|f(n)| \leq C|g(n)|$ for all $n > k$. 

Rosen p. 205
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
\[
|f(n)| \leq C|g(n)| \quad \text{for all } n > k.
\]
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

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

to mean there are constants, C and k such that \( |f(n)| \leq C|g(n)| \) 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)"

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

A family of functions which grow no faster than 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 \textbf{value} of f(n) might always be bigger than the \textbf{value} of g(n).
- \( O(g(n)) \) contains functions that grow \textbf{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
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 \( f(n) \cdot h(n) \) 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: then \( f(n) \) grows strictly slower than \( g(n) \).

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

III. If the limit tends to infinity: then \( f(n) \) 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 \( \Theta(n^2) \)?
Selection Sort (MinSort) Performance

Rosen page 210, example 5

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.

**procedure** linear search (x: integer, $a_1$, $a_2$, ..., $a_n$: distinct integers )

i := 1

while (i <= n and x ≠ $a_i$)
    i := i+1

if i <=n 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 }
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 \( x \).

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
Computing runtime performance of more complicated code and algorithms

- Binary Search algorithm
- Consecutive code segments
- Nested code segments
Reminders

HW 2 due Wednesday 11:59pm.

Lots of office hours today and tomorrow – see course website.

Submit assignments through Gradescope – one submission per group.