Performance and Asymptotics

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Instructor</th>
<th>Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lecture B</td>
<td>Jones</td>
<td>MWF 9-9:50pm</td>
<td>PCYNH 109</td>
</tr>
<tr>
<td>Lecture D</td>
<td>Russell</td>
<td>MWF 4-4:50am</td>
<td>Center 101</td>
</tr>
</tbody>
</table>

http://cseweb.ucsd.edu/classes/sp16/cse21-bd/

April 4, 2016
Apply to be a CSE Early Research Scholar!

Info session: Monday, April 11, 3-4pm in CSE 1202
Apply at [http://goo.gl/forms/YIUo4wZE5a](http://goo.gl/forms/YIUo4wZE5a)
DEADLINE APRIL 20
More information available:
[https://sites.google.com/a/eng.ucsd.edu/cse-ersp/](https://sites.google.com/a/eng.ucsd.edu/cse-ersp/)
or email [cjalvarado@eng.ucsd.edu](mailto:cjalvarado@eng.ucsd.edu)
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 operations: 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?

What other operations does MinSort do?
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 < aₘ ) 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)

\[(n-1) + (n-2) + ... + (1)\]

= \[n(n-1)/2\]
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

Input size

Number of steps the algorithm requires

Time for each of these steps on our system
Runtime performance

TritonSort is a project here at UCSD that has the world record sorting speeds, 4 TB/minute. It combines algorithms (fast versions of radix sort and quicksort), parallelism (a tuned version of Hadoop) and architecture (making good use of memory hierarchy by minimizing disc reads and pipelining data to make sure that processors always have something to compare). I think it is a good example of the different hardware, software and algorithm components that affect overall time. This is a press release

CNS Graduate Student Once Again Breaks World Record! (2014) Michael Conley, a PhD student in the CSE department, once again won a data sort world record in multiple categories while competing in the annual Sort Benchmark competition. Leading a team that included Professor George Porter and Dr. Amin Vahdat, Conley employed a sorting system called Tritonsort that was designed not only to achieve record breaking speed but also to maximize system resource utilization. Tritonsort tied for the “Daytona Graysort” category and won outright in both the “Daytona” and “Indy” categories of the new “Cloudsort” competition. To underscore the effectiveness of their system resource utilization scheme as compared to the far more resource intensive methods followed by their competitors, it’s interesting to note that the 2011 iteration of Tritonsort still holds the world record for the “Daytona” and “Indy” categories of the “Joulesort” competition.
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
Rate of growth

Focus on how time scales for large inputs

Ignore what we can't control

Which of these functions do you think has 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

Ignore what we can't control

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
Definition of Big O

Ignore what we can't control

Focus on how time scales for large inputs

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
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 \( O( n^2 ) \) ?
Big O : Potential pitfalls

"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 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 \( h(n) \) is nonnegative, then \( f(n) * h(n) \) is big-O of \( g(n) * h(n) \), 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
Selection Sort (MinSort) Pseudocode

Rosen page 203, exercises 41-42

**procedure** selection sort($a_1$, $a_2$, ..., $a_n$: real numbers with $n \geq 2$ )

for $i := 1$ to $n-1$
  
  $m := i$

  for $j := i+1$ to $n$
    
    if $a_j < a_m$ then $m := j$

  interchange $a_i$ and $a_m$

{ $a_1$, ..., $a_n$ is in increasing order}
Computing the big-O class of algorithms

How to deal with …

Basic operations

Consecutive (non-nested) code

Loops (simple and nested)

Subroutines
Computing the big-O class of algorithms

How to deal with …

**Basic operations**: operation whose time doesn't depend on input

**Consecutive (non-nested) code**: one operation followed by another

**Loops (simple and nested)**: while loops, for loops

**Subroutines**: method calls
Consecutive (non-nested) code: Run Prog$_1$ followed by Prog$_2$

If Prog$_1$ takes $O(f(n))$ time and Prog$_2$ takes $O(g(n))$ time, what's the big-O class of runtime for running them consecutively?

A. $O(f(n) + g(n))$ [sum]
B. $O(f(n) g(n))$ [multiplication]
C. $O(g(f(n)))$ [function composition]
D. $O(\max(f(n), g(n)))$
E. None of the above.
Computing the big-O class of algorithms

Simple loops:

```
while (Guard Condition)
  Body of the Loop
```

What's the runtime?

A. Constant
B. Same order as the number of iterations through the loop.
C. Same order as the runtime of the guard condition
D. Same order as the runtime of the body of the loop.
E. None of the above.
Computing the big-O class of algorithms

**Simple loops:**

```
while (Guard Condition)
  Body of the Loop
```

If Guard Condition uses basic operations and body of the loop is constant time, then runtime is of the same order as the number of iterations.
Computing the big-O class of algorithms

Nested code:

```plaintext
while (Guard Condition)
    Body of the Loop,
    May contain other loops, etc.
```

If Guard Condition uses basic operations and body of the loop has constant time runtime $O(T_2)$ in the worst case, then runtime is

$$O(T_1 T_2)$$

where $T_1$ is the bound on the number of iterations through the loop.

Product rule
Subroutine Call method $S$ on (some part of) the input.

If sub-routine $S$ has runtime $T_S(n)$ and we call $S$ at most $T_1$ times,

A. Total time for all uses of $S$ is $T_1 + T_S(n)$
B. Total time for all uses of $S$ is $\max(T_1, T_S(n))$
C. Total time for all uses of $S$ is $T_1 T_S(n)$
D. None of the above
**Subroutine**  Call method S on (some part of) the input.

If sub-routine S has runtime is \( O( T_S(n) ) \) and if we call S at most \( T_1 \) times, then runtime is

\[
O( T_1 T_S(m) )
\]

where \( m \) is the size of biggest input given to S.

*Distinguish between the size of input to subroutine, \( m \), and the size of the original input, \( n \), to main procedure!*
Selection Sort (MinSort) Pseudocode

Before, we counted comparisons, and then went to big-O

\[
\text{procedure} \ \text{selection\ sort}(a_1, a_2, \ldots, a_n:\ \text{real\ numbers\ with\ } n \geq 2)\\
\text{for}\ i := 1\ \text{to}\ n-1\\
\quad m := i\\
\text{for}\ j := i+1\ \text{to}\ n\\
\quad \text{if}\ (a_j < a_m)\ \text{then}\ m := j\\
\quad \text{interchange}\ a_i\ \text{and}\ a_m\\
\{ a_1, \ldots, a_n \text{ is in increasing order}\}
\]

\[
(n-1) + (n-2) + \ldots + (1) = n(n-1)/2 \\
\in O(n^2)
\]
Selection Sort (MinSort) Pseudocode

```
procedure selection sort(a_1, a_2, ..., a_n: real numbers with n >=2 )
for i := 1 to n-1
    m := i
    for j:= i+1 to n
        if ( a_j < a_m ) then m := j
    interchange a_i and a_m

{ a_1, ..., a_n is in increasing order}
```

Strategy: work from the inside out
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
    \textbf{if} ( aₗ < aᵣ ) \textbf{then} m := j \hspace{1cm} O(1)
interchange aᵢ and aᵣ
{ a₁, ..., aₙ is in increasing order}
**Selection Sort (MinSort) Pseudocode**

Now, straight to big O

```plaintext
procedure selection sort(a_1, a_2, ..., a_n: real numbers with n >=2 )
for i := 1 to n-1
  m := i
  for j := i+1 to n
    if (a_j < a_m) then m := j
  interchange a_i and a_m

{ a_1, ..., a_n is in increasing order}
```

**Strategy: work from the inside out**

Simple for loop, repeats n-i times
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 < aₘ) then m := j 
interchange aᵢ and aₘ 
{ a₁, ..., aₙ is in increasing order} 

Strategy: work from the inside out
Selection Sort (MinSort) Pseudocode

Now, straight to big O

procedure selection sort(a_1, a_2, ..., a_n: real numbers with n >=2 )
for i := 1 to n-1
    m := i
    for j := i+1 to n
        if ( a_j < a_m ) then m := j
    interchange a_i and a_m

{ a_1, ..., a_n is in increasing order}

Worst case: when i =1, O(n)

Strategy: work from the inside out
Selection Sort (MinSort) Pseudocode

Now, straight to big O

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

{ a₁, ..., aₙ is in increasing order}
```

Strategy: work from the inside out
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}

Strategy: work from the inside out
procedure selection sort(a_1, a_2, ..., a_n: real numbers with n >= 2)
for i := 1 to n-1
    m := i
    for j := i+1 to n
        if a_j < a_i then
            m := j
    interchange a_i and a_m

{ a_1, ..., a_n is in increasing order}

Strategy: work from the inside out

Now, straight to big O

O(n)

Nested for loop, repeats O(n) times

Total: O(n^2)
Next Time

Analyzing algorithms that solve other problems (besides sorting and searching)

Designing better algorithms
• pre-processing
• re-use of computation