PCNYH 106

CSE21 Winter 2017, Day 5 (B00), Day 3 (A00)

January 20, 2017

http://vlsicad.ucsd.edu/courses/cse21-w17
Big-O : How to determine?

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 \( \cdot \) 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 determine?

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

**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 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
\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \Rightarrow \text{for every } \varepsilon > 0 \exists N \in \mathbb{N} \text{ such that } \\
\frac{f(n)}{g(n)} < \varepsilon \text{ for all } n \geq N
\]

\[
f(n) < \varepsilon \cdot g(n) \text{ for all } n \geq N
\]

\[\Rightarrow \forall \varepsilon \in \mathbb{R}_{>0} \exists N \in \mathbb{N} \text{ such that } f(n) \in O(g(n))\]
Other asymptotic classes and notation

For some $c > 0$

$f(n) \in O(g(n))$ \quad \Rightarrow \quad \lim_{n \to \infty} \frac{f(n)}{g(n)} \leq c \quad \text{for some constant } c > 0$

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

$f(n) \in \Theta(g(n))$ \quad \Rightarrow \quad \lim_{n \to \infty} \frac{f(n)}{g(n)} = c \quad \text{for some } c > 0$

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 (Again)
Rosen page 210, example 5

Number of comparisons of list elements

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

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

Rewrite this expression using big-O notation:

A. \(O(n)\)
B. \(O(n(n-1))\)
C. \(O(n^2)\)
D. \(O(1/2)\)
E. None of the above
Procedure linear search (x: integer, a₁, a₂, ..., aₙ: distinct integers)
    i := 1
    while (i <= n and x ≠ aᵢ)
        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? \( 1 \)
- if \( x \) happens to equal the **last** element in the list? \( n \)
- if \( x \) happens to equal an element somewhere in the **middle** of the list? \( \sim \frac{n}{2} \)
- if \( x \) **doesn't equal any** element in the list? \( n \)

The runtime of Linear Search is \( O(n) \).
Best case: 1 probe  
Worst case: n probes  
Average case: n/2 probes

target appears first  
target appears last or not at all  
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, a₁, a₂, ..., aₙ: increasing integers)
i := 1
j := n
while i<j
    m := floor((i+j)/2) 
    if x > aₗ then i := m+1
    else j := m
if x=aᵢ 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>

\( \text{ceil}(\log_2(n)) \)
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
\[ \log_a(n) = \Theta\left(\log_b(n)\right) \]

for all \( a, b > 0 \)

\[ \log_a(n) = \frac{\log_b(n)}{\log_b(a)} \]

Want to find some \( k \) such that

\[ \frac{n}{2^k} = 1 \quad k = \log_2(n) \quad \left(\log_2(n) \leq \log_2(n) \leq \left\lceil \log_2(n) \right\rceil \right) \]
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># 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>

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

<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>
<tr>
<td>Assumptions</td>
<td>Linear Search</td>
<td>Binary Search</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td># probes in</td>
<td></td>
<td></td>
</tr>
<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>

Comparing linear search and binary search

Rosen pages 220-221

Is it worth it to sort our list first?

Sorting takes $\Omega(n \log n)$ runtime.
Determining the big-O class of algorithms

How to deal with …

Basic operations

Consecutive (non-nested) code

Loops (simple and nested)

Subroutines
Determining 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 Prog₁ followed by Prog₂

If Prog₁ takes \(O(f(n))\) time and Prog₂ 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. \(O(f(n) \cdot g(n))\) [[multiplication]]
C. \(O(g(f(n)))\) [[function composition]]
D. \(O(\max(f(n), g(n)))\)
E. None of the above.
Determining the big-O class of algorithms

Simple loops: \( \textbf{while} \ (\text{Guard Condition}) \)

Body of the Loop

What's the runtime? (assuming th Guard Condition is O(1))? 

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.
Determining the big-O class of algorithms

Simple loops:

\[ \text{while (Guard Condition)} \]

\[ \text{Body of the Loop} \]

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.
Determining the big-O class of algorithms

Nested code:

\[
\text{while (Guard Condition)}
\]

Body of the Loop,
May contain other loops, etc.

If Guard Condition uses basic operations and body of the loop has 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 subroutine 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 $T_1 \cdot T_S(n)$
D. None of the above
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 (a_j < a_m) then m := j
  interchange a_i and a_m

{ a₁, ..., aₙ is in increasing order}

\[(n-1) + (n-2) + \ldots + 1\]
\[= \frac{n(n-1)}{2}\]
\[\in O(n^2)\]
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}
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
        if (aⱼ < aₘ) then m := j
    interchange aᵢ and aₘ

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

Strategy: work from the inside out

Now, straight to big O  

Simple for loop, repeats n-i times  

worse case is when i = 1

n-1 = O(n)
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}
Selection Sort (MinSort) Pseudocode

Now, straight to big O

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}

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

Strategy: work from the inside out
procedure selection sort($a_1, a_2, \ldots, a_n$: real numbers with $n \geq 2$)
for $i := 1$ to $n-1$
\hspace{1em} $m := i$
\hspace{1em} $O(1)$
\hspace{1em} $m := j$
\hspace{1em} $O(n)$
\hspace{1em} interchange $a_i$ and $a_m$
\hspace{1em} $O(1)$
\hspace{1em} { $a_1, \ldots, a_n$ is in increasing order}

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

Now, straight to big O

procedure selection sort(a₁, a₂, ..., aₙ: real numbers with n >=2 )
for i := 1 to n-1

O(n)

{ 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

{ a₁, ..., aₙ is in increasing order}

Total: O(n²)

Strategy: work from the inside out
Next Time

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

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