Performance and Asymptotics

CSE21 Winter 2017, Day 4 (B00), Day 3 (A00)

January 18, 2017

Sections @ ucsd.edu.
Using order in search

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

We didn't assume \textbf{anything} about the list!!!
Using order in search

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

We didn't assume anything about the list!!!

Can we do better if the list is sorted?

A. Yes, we can modify Linear Search to take advantage of sorted order.
B. Yes, we can devise a totally new algorithm which uses the sorted order.
C. Yes, even though we might always need to look at all list elements in the worst case, we'll be able to do better on average.
D. No.
Using order in search

Suppose our list is sorted. If we probe the list at position $m$, what do we learn if …

$x = a_m$? We're done!
Using order in search

Suppose our list is sorted. If we probe the list at position $m$, what do we learn if …

$x = a_m$ ? We're done!

$x < a_m$ ?

A. if $x$ is in the list, it occurs BEFORE position $m$
B. if $x$ is in the list, it occurs AFTER position $m$
C. $x$ occurs at position $m$
D. $x$ is not in the list
Suppose our list is sorted. If we probe the list at position \( m \), what do we learn if …

\[
\begin{align*}
    x &= a_m \? & \text{We're done!} \\
    x &< a_m \? & \text{Need to check positions } 1 \ldots m-1 \\
    x &> a_m \? & \text{Need to check positions } m+1 \ldots n
\end{align*}
\]
Using order in search

Suppose our list is sorted. If we probe the list at position $m$, what do we learn if …

$x = a_m$ ? We're done!

$x < a_m$ ? Need to check positions 1 … $m-1$

$x > a_m$ ? Need to check positions $m+1$ … $n$

Have we made any progress?
Binary Search: HOW

Starting at the middle of the list and based on what you find, determine which half to search next. Continue until target is found or sure that it is missing.

procedure binary search (x: integer, a_1, a_2, ..., a_n: increasing integers )

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

Starting at the middle of the list and based on what you find, determine which half to search next. Continue until target is found or sure that it is missing.

procedure binary search (x: integer, a₁, a₂, ..., aₙ: increasing integers )
i := 1
j := n
while ????

    m := floor( (i+j)/2 )
    if x > aₘ then i := m+1
    else j := m

if x=aᵢ then location := 1
else location := 0
return location

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

Starting at the middle of the list and based on what you find, determine which half to search next. Continue until target is found or sure that it is missing.

procedure binary search (x: integer, a_1, a_2, ..., a_n: increasing integers )
i := 1
j := n
while ????
    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 }
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 }

Example: x=5

i=1, j=8
i=5, j=7

m=4, x>a_4
m=6, x<a_6

Exercise: finish up this trace through example.

How do we prove this? Informally, the list to consider gets smaller and a_i <= x <= a_j
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**
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 …

<table>
<thead>
<tr>
<th>Time</th>
<th>Number of operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Comparisons of list elements!</td>
</tr>
</tbody>
</table>

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?
procedure selection sort(\(a_1, a_2, \ldots, a_n\): 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.

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
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 (+, -, *, …) (for small numbers)
- etc.

"Single step" depends on context
Runtime performance

How long does a “single step” take?

Some factors
- Hardware
- Software

Let’s discuss and list 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:

- Ignore what we can't control
- Estimate time as a function of the size of the input, n
- Focus on how time scales for large inputs
Focus on how time scales for large inputs

Ignore what we can't control

In this class two functions have the same rate of growth if they differ by a constant factor.

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
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
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

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

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 

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

**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 \( g(n) \).
Big-O : Notation and terminology

"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)

Name some functions that are in the family O( n^2 ) …

\[ n, \quad 3n^2, \quad 1, \quad \log n, \quad 500,000 n^2 \]
"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 \( g(n) \).
  - True

• \( O(g(n)) \) contains functions that grow strictly slower than \( g(n) \).
  - False
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. 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: 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 and notation

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

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

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

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

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

\[ f(n) \in O(g(n)) \text{ and } g(n) \in O(f(n)) \]

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

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

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
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 }
How fast is Linear Search: WHEN

Rosen page 220, part of example 2

The time it takes to find \( x \) (or determine it is not present) depends on the number of \textit{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 \textit{first} element in the list?

✧ if \( x \) happens to equal the \textit{last} element in the list?

✧ if \( x \) happens to equal an element somewhere in the \textit{middle} of the list?

✧ if \( x \) \textit{doesn't equal any} element in the list?
<table>
<thead>
<tr>
<th>Case</th>
<th>Probes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best case:</td>
<td>1 probe</td>
<td>target appears first</td>
</tr>
<tr>
<td>Worst case:</td>
<td>n probes</td>
<td>target appears last or not at all</td>
</tr>
<tr>
<td>Average case:</td>
<td>n/2 probes</td>
<td>target appears in the middle, (expect to have to search about half of the array ... more on expected value later in the course)</td>
</tr>
</tbody>
</table>

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>
How fast is Binary Search: WHEN

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></td>
</tr>
<tr>
<td>ceil(log₂ n)</td>
<td></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

<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>

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># 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>
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>

Is it worth it to sort our list first?
Announcements

Need help? Office Hours!
See the calendar on the class website.

Need 1-1 help?
Mechanism announced on Piazza (by end of Week 2)

Useful Puzzles!
(for interviews, etc.)
Will try out a pinned Piazza post on this…