CSE21 Fall 2016, Day 3
September 28, 2016

http://cseweb.ucsd.edu/classes/fa16/cse21-ab/
Why sort?

A TA facing a stack of exams needs to input all 400 scores into a spreadsheet where the students are listed in alphabetical order.

OR

It's easier to access data when it is sorted because you know exactly where to find it.

Really???
Two searching algorithms:

- One that works for any data, sorted or not
- One that is much faster, but relies on the data being sorted
Given a list

\[ a_1, a_2, \ldots, a_n \]

and a target value \( x \) find an index \( j \) where \( x = a_j \)

or determine that there is no such index.

Values can be any type. For simplicity, use integers.
Pseudocode:

```
procedure mystery (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 }

In groups, describe in English (high-level), what the algorithm is doing.
Correctness of Linear Search: WHY

What's the loop invariant?

Try to write it down yourself, first.
Correctness of Linear Search: WHY

**Step 1:** Loop invariant is:
"After the $t^{th}$ iteration of the **while** loop, guarantee that $x$ is not equal to any of the first $t$ entries of the list."

**Step 2:** Prove that this loop invariant holds.

**Step 3:** Use the invariant to prove that linear search is correct.
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!!!
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!
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
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 $m+1$ … $n$
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
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)

{ 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.

```plaintext
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=a₁ 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 }
```
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.

```plaintext
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_1 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 }
```

A. i<n
B. j<n
C. i<j
D. j<i
E. None of the above
procedure binary search (x: integer, a₁, a₂, ..., aₙ: increasing integers)
i := 1
j := n
while i<j
    m := floor((i+j)/2)
    if x > aₘ 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 do we prove this? Informally, the list to consider gets smaller and aᵢ ≤ x ≤ aⱼ
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

Comparisons of list elements!

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

How many times do we compare pairs of elements?
A. Depends on the values of the elements of the list
B. n
C. n²
D. None of the above.
Selection Sort (MinSort) Pseudocode

Rosen page 203, exercises 41-42

\begin{verbatim}
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
\end{verbatim}

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

In the body of the outer loop, when i=1, how many times do we compare pairs of elements?

A. 1  
B. n  
C. n-1  
D. n+1  
E. 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 ( aⱼ < aₘ ) then m := j
        interchange aᵢ and aₘ

{ a₁, ..., aₙ is in increasing order}
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.
(n-1) + (n-2) + ... + (1)
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}
Selection Sort (MinMaxSort) 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 ( aₗ < aₘ ) then m := j
    interchange aᵢ and aₘ

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

On course website:
- Guide to proofs and logic
- Guide to correctness proofs
- Extra practice problems on invariants and induction

HW1 due
Sunday
10/2 at 10pm
One submission per group.

Office Hours
See the calendar on the class website.