University of Hawaii

ICS141: Discrete Mathematics for Computer Science I

Dept. Information & Computer Sci., University of Hawaii

Jan Stelovsky

based on slides by Dr. Baek and Dr. Still

Originals by Dr. M. P. Frank and Dr. J.L. Gross

Provided by McGraw-Hill

Lecture 14

Chapter 3. The Fundamentals

3.1 Algorithms

Algorithms

- Previously…
  - Characteristics of algorithms
  - Pseudocode
  - Examples: Max algorithm
- Today…
  - Examples: Sum algorithm
  - Problem of searching an ordered list
    - Linear search & binary search algorithms
  - Sorting problem
    - Bubble sort & insertion sort algorithms

Practice Exercises

University of Hawaii

- Devise an algorithm that finds the sum of all the integers in a list.

- **procedure** *sum*($a_1, a_2, \ldots, a_n$: integers)

  $s := 0$ {sum of elements so far}

  for $i := 1$ **to** n {go thru all elements}

  $\quad s := s + a_i$ {add current item}

  {now s is the sum of all items}

  return s

Searching Algorithms

University of Hawaii

- Problem of *searching an ordered list*.
  - Given a list L of n elements that are sorted into a definite order (*e.g.*, numeric, alphabetical),
  - And given a particular element x,
  - Determine whether x appears in the list,
  - And if so, return its index (position) in the list.
- Problem occurs often in many contexts.
- Let's find an *efficient* algorithm!

Linear Search (Naïve)

{Given a list of integers and an integer x to look up, returns the index of x within the list or 0 if x is not in the list}

procedure *linear_search*

(x: integer, $a_1, a_2, \ldots, a_n$: distinct integers)

$i := 1$ {start at beginning of list}

while $(i \leq n \wedge x \neq a_i)$ {not done and not found}

$i := i + 1$ {go to the next position}

if $i \leq n$ **then** $index := i$ {it was found}

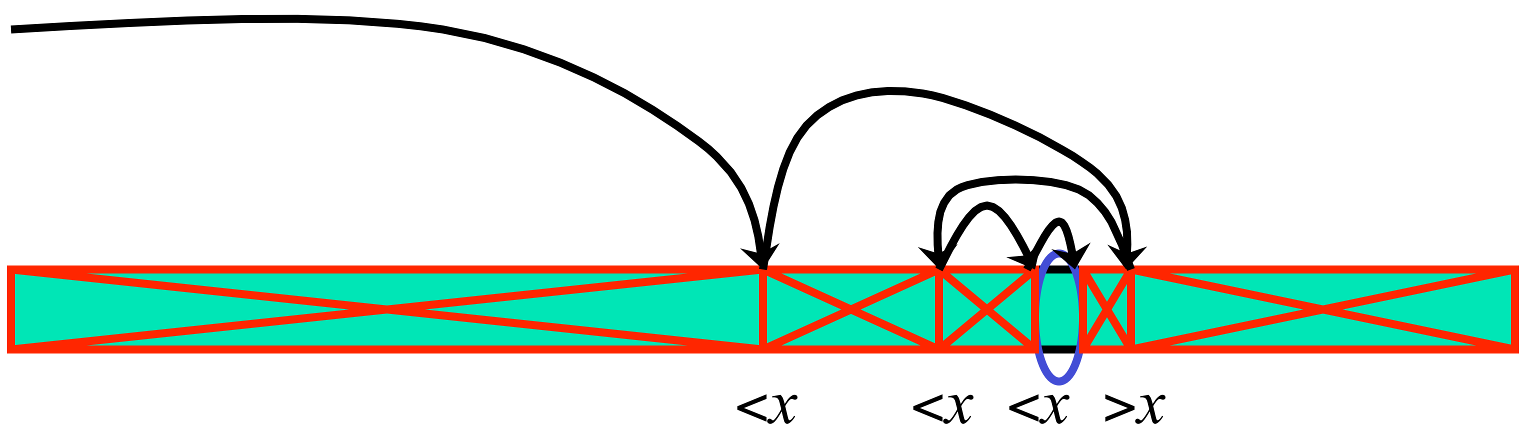
else $index := 0$ {it wasn't found}

return *index* {index where found or 0 if not found}

Alg. #2: Binary Search

University of Hawaii

- Basic idea: At each step, look at the *middle* element of the remaining list to eliminate half of it, and quickly zero in on the desired element.



Search Alg. #2: Binary Search

procedure *binary_search*
(x: integer, $a_1, a_2, \ldots, a_n$: increasing integers)
$i := 1$ {left endpoint of search interval}
$j := n$ {right endpoint of search interval}
while $i < j$ **begin** {while interval has > 1 item}
$m := \lfloor (i + j)/2 \rfloor$ {midpoint}
if $x > a_m$ **then** $i := m + 1$ **else** $j := m$
end
if $x = a_i$ **then** $location := i$ **else** $location := 0$
return *location* {index or 0 if not found}

University of Hawaii

Search Example

- Search for 19 in the list

Index:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	1,	2,	3,	5,	6,	7,	8,	10,	12,	13,	15,	16,	18,	19,	20,	22
	i								*i*							*j*
													i	*j*		
														i		

  - using linear search
  - using binary search
    - *Entering while loop*: $i = 1$, $j = 16$
    - $m = \lfloor (i+j)/2 \rfloor = \lfloor (1+16)/2 \rfloor = \lfloor 8.5 \rfloor = 8$,
    - $m = \lfloor (i+j)/2 \rfloor = \lfloor (9+16)/2 \rfloor = \lfloor 12.5 \rfloor = 12$,
    - $m = \lfloor (i+j)/2 \rfloor = \lfloor (13+16)/2 \rfloor = \lfloor 14.5 \rfloor = 14$,
    - $m = \lfloor (i+j)/2 \rfloor = \lfloor (13+14)/2 \rfloor = \lfloor 13.5 \rfloor = 13$,
    - *Exit loop*

Sorting Algorithms

University of Hawaii

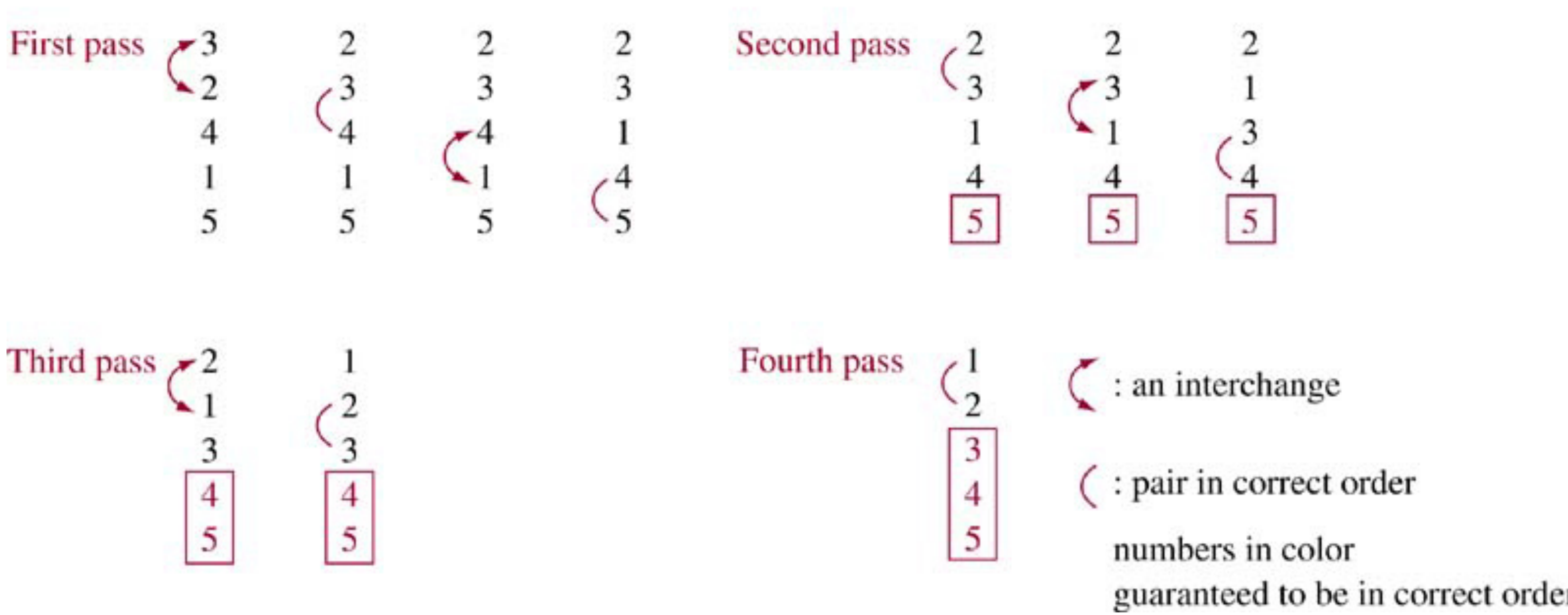
- Sorting is common in many applications.
  - *E.g.* spreadsheets and databases
  - We can search quickly when data is odrered!
- Sorting is also widely used as a subroutine in other data-processing algorithms.
- Two sorting algorithms shown in textbook:
  - Bubble sort
  - Insertion sort

However, these are *not* very efficient, and you should not use them on large data sets!

We'll see some more efficient algorithms later in the course.

Bubble Sort

- Smaller elements “float” up to the top of the list, like bubbles in a container of liquid, and the larger elements “sink” to the bottom.



Bubble Sort Algorithm

procedure *bubble_sort*

($a_1, a_2, \ldots, a_n$: real numbers, $n \geq 2$)

for $i := 1$ **to** $n - 1$ {iterate $n - 1$ passes}

for $j := 1$ **to** $n - i$

if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

{$a_{n-i+1}, \ldots, a_n$ is sorted and $\leq a_1, \ldots, a_{n-i}$}

{$a_1, a_2, \ldots, a_n$ is sorted}

- Example 4: Use the bubble sort to put 3, 2, 4, 1, 5 into increasing order. (See previous slide)

Insertion Sort

University of Hawaii

- English description of algorithm:
  - Start with the second element,
    for each item in the input list:
    - "Insert" it into the correct place in the sorted output list generated so far. Like so:
      - Find the location where the new item should be inserted using linear or binary search.
      - Then, shift the items from that position onwards up by one position.
      - Put the new item in the remaining hole.

Insertion Sort Example

- Use the insertion sort to put 3, 2, 4, 1, 5 into increasing order
  - Insert the 2nd element 2 in the right position:
    - $3 > 2 \Rightarrow$ put 2 in front of 3. $\Rightarrow$ 2, 3, 4, 1, 5
  - Insert the 3rd element 4 in the right position:
    - $4 > 2 \Rightarrow$ do nothing. Move to the next comparison.
    - $4 > 3 \Rightarrow$ do nothing. Done. $\Rightarrow$ 2, 3, 4, 1, 5
  - Insert the 4th element 1 in the right position:
    - $2 > 1 \Rightarrow$ put 1 in front of 2. $\Rightarrow$ 1, 2, 3, 4, 5
  - Insert the 5th element 5 in the right position:
    - $5 > 1 \Rightarrow$ do nothing. Move to the next comparison.
    - $5 > 2 \Rightarrow$ do nothing. Move to the next comparison.
    - $5 > 3 \Rightarrow$ do nothing. Move to the next comparison.
    - $5 > 4 \Rightarrow$ do nothing. Done. $\Rightarrow$ 1, 2, 3, 4, 5

Insertion Sort Algorithm

University of Hawaii

procedure *insertion_sort*
($a_1, a_2, \ldots, a_n$: real numbers, $n \geq 2$)
for $i := 2$ **to** n **begin**
$m := a_i$ {the element to be inserted}
$j := 1$
while $a_j < m$ {look for the index of the hole with j }
$j := j + 1$
{now $a_1, \ldots, a_{j-1} < m \leq a_j, \ldots, a_i$}
{the hole is at j; $j \leq i$, i.e. possibly $j = i$ }
for $k := j + 1$ **to** i
$a_k := a_{k+1}$
$a_j := m$
{$a_1, a_2, \ldots, a_i$ are sorted in increasing order}
end {$a_1, a_2, \ldots, a_n$ are sorted in increasing order}

Efficiency of Algorithms

- Intuitively we see that binary search is much faster than linear search,
- Also, we may see that insertion sort is better than bubblesort in some cases (why? when?),
- But how do we analyze the efficiency of algorithms formally?
- Use methods of ***algorithmic complexity***, which utilize the order-of-growth concepts