University of Colorado **Boulder**

Department of Computer Science
CSCI 2824: Discrete Structures
Chris Ketelsen

Algorithms and Complexity

Algorithms

Searching Algorithms

Task: Find the location of an element in a sequence or determine that the desired element is not in the list

Application: Check if a word is in a dictionary

Example: Find the location of 19 in the sequence

$$\{1, 25, 3, 16, 7, 10, 5, 19, 21, 2\}$$

We'll look at two algorithms: **Linear Search** and **Binary Search**

Linear Search

procedure `LinearSearch`$(x, a_1, a_2, \ldots, a_n)$

$i := 1$

while $(i \leq n \text{ and } x \neq a_i)$

$\qquad i := i + 1$

if $i \leq n$ **then** $location := i$

else $location := -1$

return $location$

Question: How fast is this?

Binary Search

Task: Find the location of an element in an *ordered* sequence or determine that the desired element is not in the list

Example: Find 19 in the sequence $\{1, 2, 3, 5, 7, 10, 16, 18, 19, 25\}$

Binary Search leverages the fact that the list is sorted

Binary Search

Example: Find 19 in the sequence $\{1, 2, 3, 5, 7, 10, 16, 18, 19, 25\}$

1. Break list into two halves: 1 2 3 5 7 10 16 18 19 25

2. Compare x to largest item in left list. Since $19 > 7$, look right

10 16 18 19 25

3. Compare x to largest item in left list. Since $19 > 18$, look right

19 25

4. Compare x to largest item in left list. Since $19 \not> 19$, look left

19

5. Only one entry left. Check if $x =$ last entry. Found it!

Binary Search

procedure $\texttt{BinarySearch}(x, a_1, a_2, \ldots, a_n)$

$\ell := 1$ # ℓ is left endpoint of search interval

$r := n$ # r is right endpoint of search interval

while $\ell < r$

$m := \lfloor(\ell + r)/2\rfloor$ # m is index of largest in left list

if $x > a_m$ **then** $\ell := m + 1$, **else** $r := m$

if $x = a_\ell$ **then** $location := \ell$, **else** $location := -1$

return $location$

Binary Search

procedure $\texttt{BinarySearch}(x, a_1, a_2, \ldots, a_n)$

$\ell := 1$ # ℓ is left endpoint of search interval

$r := n$ # r is right endpoint of search interval

while $\ell < r$

  $m := \lfloor (\ell + r)/2 \rfloor$ # m is index of largest in left list

  if $x > a_m$ **then** $\ell := m + 1$, **else** $r := m$

if $x = a_\ell$ **then** *location* $:= \ell$, **else** *location* $:= -1$

return *location*

Searching Algorithms

Question: Which of the two searching algorithms seems faster?

Searching Algorithms

Question: Which of the two searching algorithms seems faster?

Be Careful: What do we mean by faster?

- Which is faster in the best-case scenario?
- Which is faster in the worst-case scenario?
- Which is faster on average?

Sorting Algorithms

Task: Given an unordered list of elements, organize them according to some notion of order

Applications: Ordering mail by location on route. Alphabetizing.

Goal: Sometimes ordering a list is the goal in-and-of itself. Sometimes we order a list so that other tasks become easier.

- binary search
- finding duplicates in a list

Huuuuge amount of computing power spent on sorting

Many many sophisticated algorithms

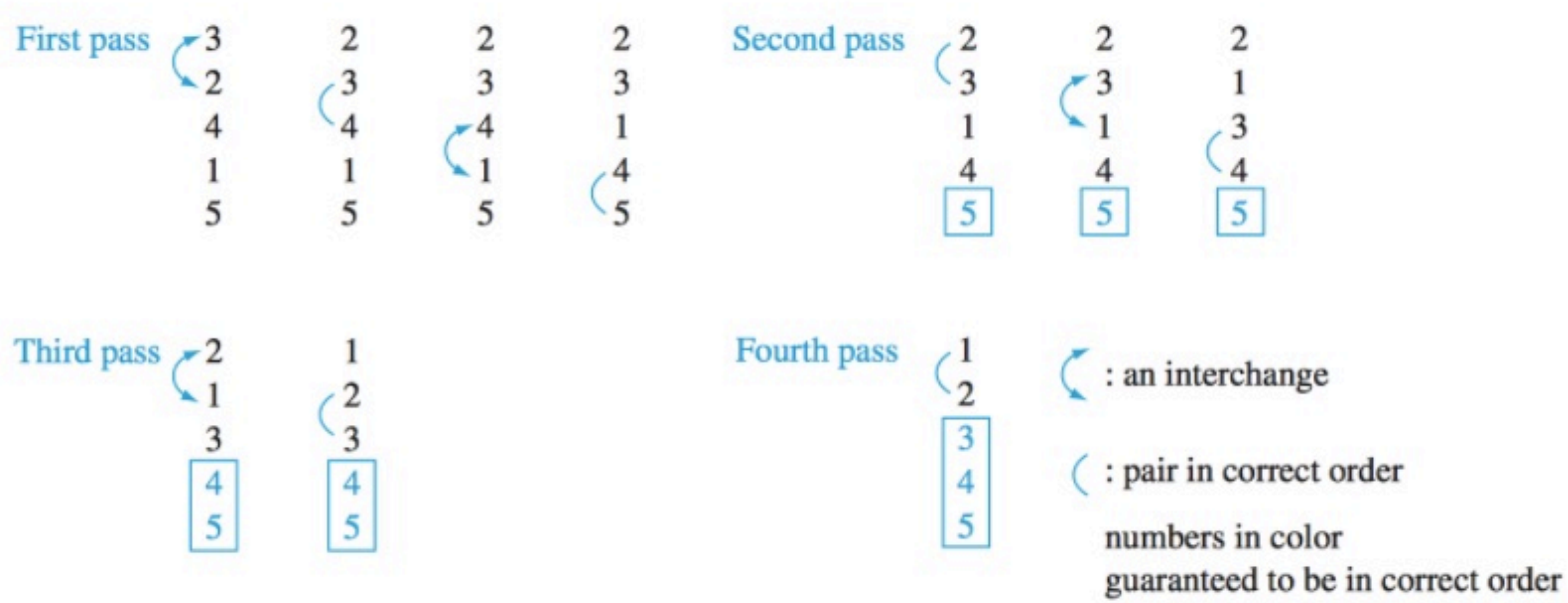
We'll look at the **bubble sort** algorithm

Bubble Sort

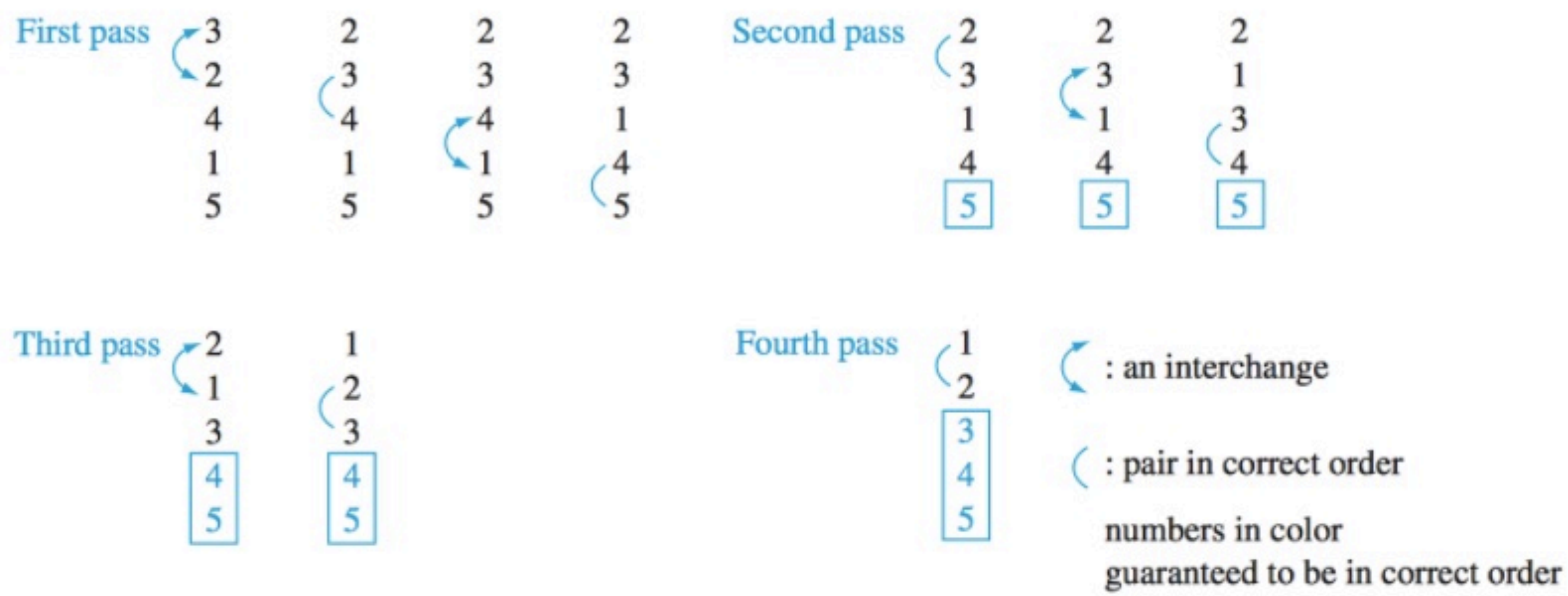
Make passes through list, interchanging adjacent pairs that are in the wrong order

Repeat until the list is sorted

Large elements sink to bottom, small elements *bubble* to top



Bubble Sort



procedure $\texttt{BubbleSort}(a_1, a_2, \ldots, a_n)$

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

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

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

Greedy Algorithms

Many algorithms are designed to solve **optimization problems**

Goal: Find solution that minimizes or maximizes some parameter

Applications

- Finding a route between cities that minimizes distance
- Encode a message using fewest bits possible

Greedy Algorithms select the best choice at each step

Note: Not guaranteed to find optimal solution. Have to check once a solution is found.

Greedy Algorithms

Task: Consider making n cents change with quarters, dimes, nickels, and pennies, using the least total amount of coins.

Greedy Strategy: At each step, choose the largest denomination coin possible to add to the pile that does not exceed n cents.

Example: Suppose we want to make change for 67 cents.

1.
2.
3.
4.
5.
6.

Greedy Algorithms

Let $c_1 > c_2 > \cdots > c_r$ denote coin denominations

procedure $\texttt{Change}(n, c_1, c_2, \ldots, c_r)$

for $i := 1$ **to** r

$d_i := 0$ # d_i counts coins of denom. c_i

while $n \geq c_i$

$d_i := d_i + 1$ # add one coin of denom. c_i

$n := n - c_i$

Greedy Algorithms

Fact: The number of coins used to make n cents using quarters, dimes, nickels, and pennies in the previous algorithm is optimal.

Fact: If we only used quarters, dimes, and pennies (and no nickels) then the algorithm would not produce an optimal solution

Example: Suppose we wanted to make change for 30 cents using only quarters, dimes, and pennies

Our algorithm would use 1 quarter (25 cents) and 5 pennies (5 cents) for a total of 6 coins used

But a more optimal solution would be to use 3 dimes