UNIVERSITY AT ALBANY

State University of New York

Lecture 13: Algorithms

Dr. Chengjiang Long

Computer Vision Researcher at Kitware Inc.

Adjunct Professor at SUNY at Albany.

Email: **clong2@albany.edu**

Outline

- Introduction to Algorithms
- Searching Algorithms
- Sorting Algorithms
- Greedy Algorithms

Outline

- **Introduction to Algorithms**
- Searching Algorithms
- Sorting Algorithms
- Greedy Algorithms

Algorithms

- When presented a problem, e.g., given a sequence of integers, find the larges one
- Construct a model that translates the problem into a mathematical context
  - Discrete structures in such models include sets, sequences, functions, graphs, relations, etc.
- A method is needed that will solve the problem (using a sequence of steps)
- **Algorithm**: a sequence of steps

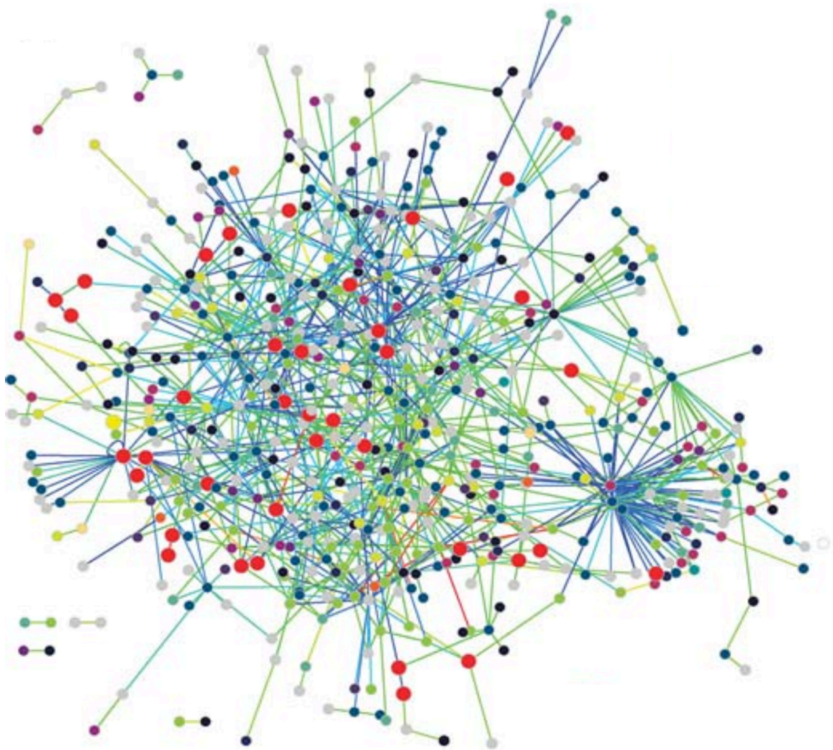
Algorithm

- **Algorithm**: a finite set of precise instructions for performing a computation or for solving a problem

- Some problems have no solution.
- Some people know solution for some problem.
- Some solutions for some problems could be described as a sequence of instructions.
- Some sequences of instructions are **algorithms**.

Algorithm

- Example: describe an algorithm for finding the maximum (largest) value in a finite sequence of integers



Example

- Perform the following steps
  - Set up temporary maximum equal to the first integer in the sequence
  - Compare the next integer in the sequence to the temporary maximum, and if it is larger than the temporary maximum, set the temporary maximum equal to this
  - Repeat the previous step if there are more integers in the sequence
  - Stop when there are no integers left in the sequence. The temporary maximum at this point is the largest integer in the sequence.

Pseudo code

- Provide an intermediate step between English and real implementation using a particular programming language

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

$max := a_1$

for i:=2 **to** *n*

    if $max < a_i$ **then** $max := a_i$

{*max* is the largest element}

Prosperities of algorithm

- **Input**: input values from a specified set
- **Output**: for each set of input values, an algorithm produces output value from a specified set
- **Definiteness**: steps must be defined precisely
- **Correctness**: should produce the correct output values for each set of input values
- **Finiteness**: should produce the desired output after a finite number of steps
- **Effectiveness**: must be possible to perform each step exactly and in a finite amount of time
- **Generality**: applicable for all problems of the desired form, not just a particular set of input values

Algorithmic Problems

As we know not all of the problems have algorithmic solution. Those that have are called algorithmic problem.

Prove that programming languages may be used to solve only algorithmic problem.

The very common application of algorithms:

- Searching Problems: finding the position of a particular element in a list.
- Sorting problems: putting the elements of a list into increasing order.
- Optimization Problems: determining the optimal value (maximum or minimum) of a particular quantity over all possible inputs.

Outline

- Introduction to Algorithms
- **Searching Algorithms**
- Sorting Algorithms
- Greedy Algorithms

Searching Problems

- The general *searching problem* is to locate an element x in the list of distinct elements $a_1, a_2, \ldots, a_n$, or determine that it is not in the list.
- The solution to a searching problem is the location of the term in the list that equals x (that is, i is the solution if $x = a_i$) or 0 if x is not in the list.

Linear Search

procedure *linear search*(x:integer, $a_1, a_2, \ldots, a_n$: distinct integers)

$i := 1$

while ($i \leq n$ **and** $x \neq a_i$)

    $i := i+1$

if $i < n$ **then** *location*$:=n$

else *location*$:=0$

{*location* is the index of the term equal to x, or is 0 if x is not found}

Binary search

- Given a sorted list, by comparing the element to be located to the middle term of the list
- The list is split into two smaller sublists (of equal size or one has one fewer term)
- Continue by restricting the search to the appropriate sublist
- Search for 19 in the (sorted) list

  1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

Binary search

- First split the list

  1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

- Then compare 19 and the largest term in the first list, and determine to use the list
- Continue

  12 13 15 16 18 19 20 22

  18 19 20 22

  19 (down to one term)

Binary search

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

i:=1 (left endpoint of search interval)

j:=1 (right end point of search interval)

while ($i<j$)

begin

m:=$\lfloor (i+j)/2 \rfloor$

if $x>a_m$ **then** i:=m+1

else j:=m

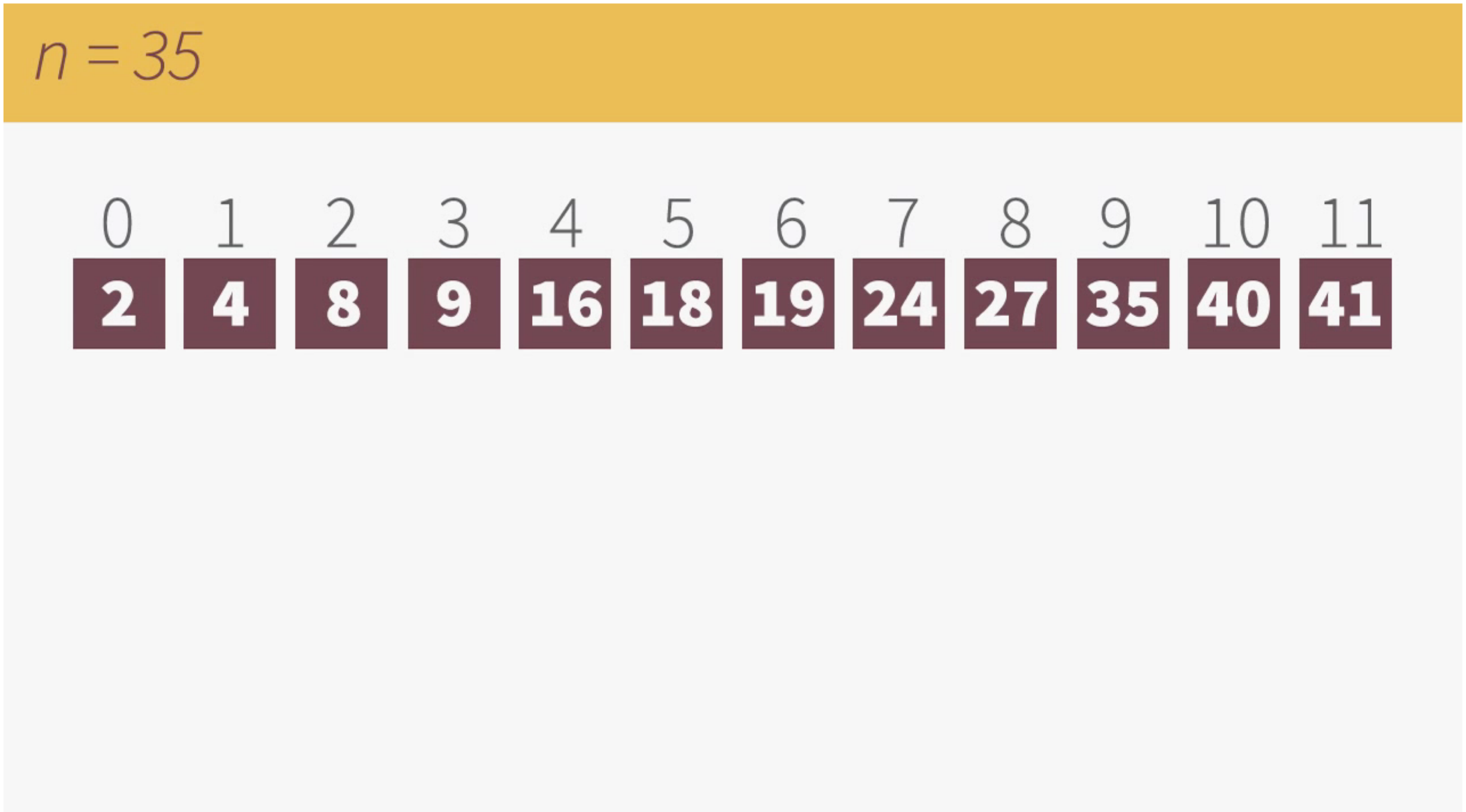
end

if $x=a_i$ **then** *location*:=*i*

else *location:=0*

{*location* is the index of the term equal to x, or is 0 if x is not found}

Binary Search Animation



Outline

- Introduction to Algorithms
- Searching Algorithms
- **Sorting Algorithms**
- Greedy Algorithms

Sorting

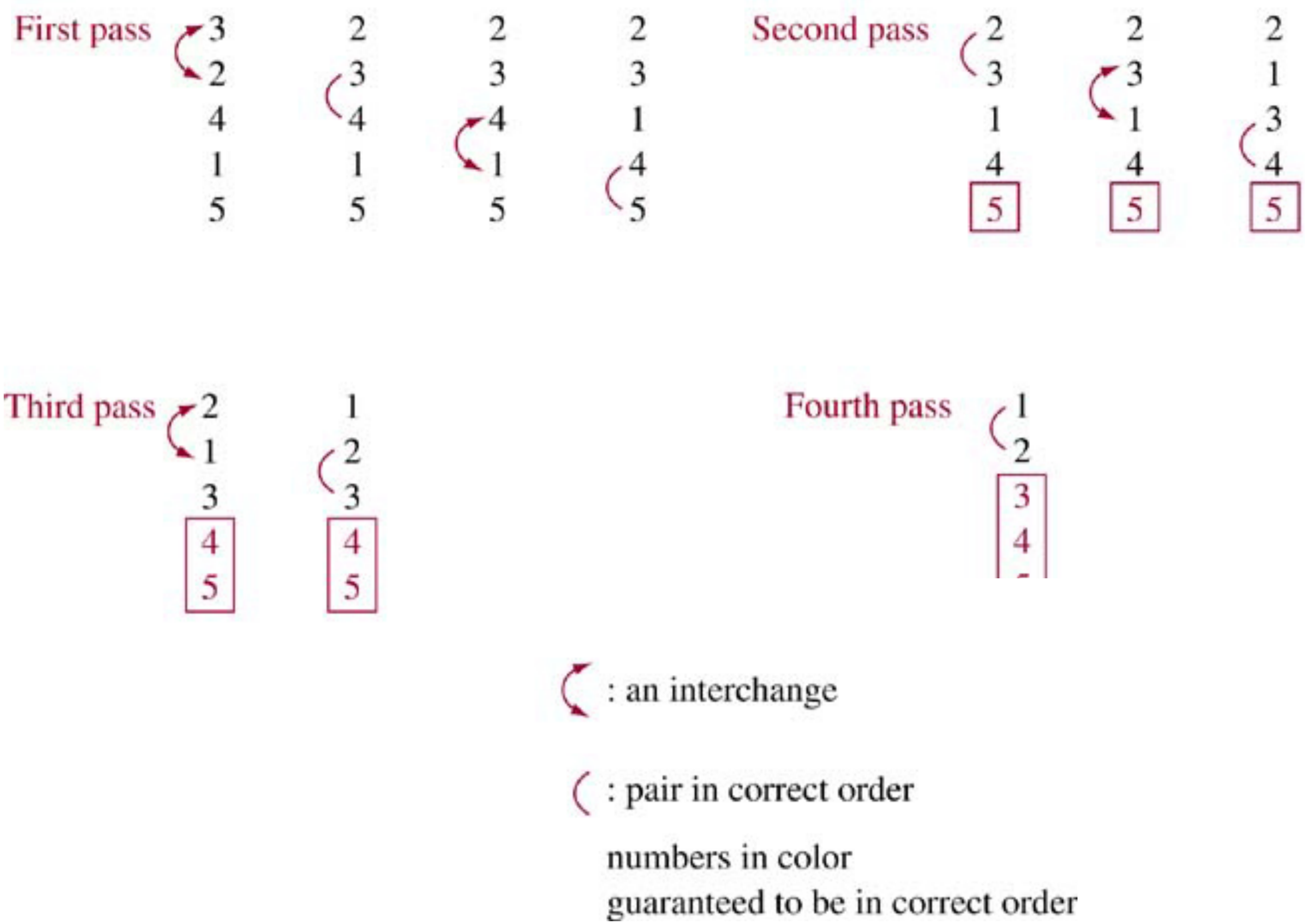
- To *sort* the elements of a list is to put them in increasing order (numerical order, alphabetic, and so on).
- Sorting is very important problem both from practical and theoretical points of view.
- There is no “ideal” sorting algorithm.
- See, https://www.toptal.com/developers/sorting-algorithms

Bubble Sort

- *Bubble sort* makes multiple passes through a list. Every pair of elements that are found to be out of order are interchanged.

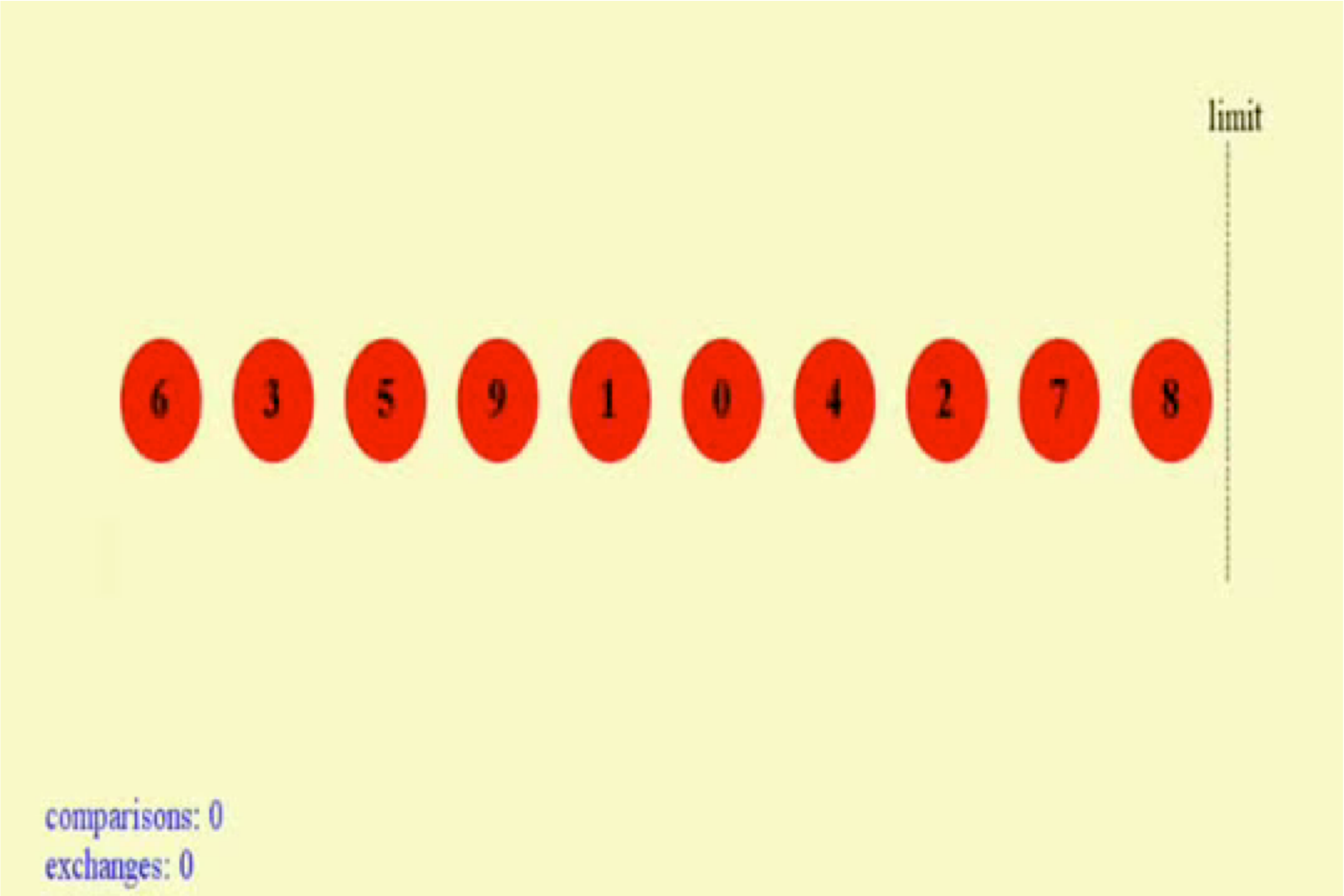
procedure *bubblesort*($a_1,\ldots,a_n$: real numbers
with $n \geq 2$)
  for i := 1 to $n - 1$
    for j := 1 to $n - i$
      if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
{$a_1,\ldots, a_n$ is now in increasing order}

Bubble Sort



First pass
Second pass
Third pass
Fourth pass
: an interchange
: pair in correct order
numbers in color
guaranteed to be in correct order

Bubble Sort Animation



Insertion Sort

- *Insertion sort* begins with the 2^{nd} element. It compares the 2^{nd} element with the 1^{st} and puts it before the first if it is not larger.
- Next the 3rd element is put into the correct position among the first 3 elements.
- In each subsequent pass, the $n+1^{st}$ element is put into its correct position among the first $n+1$ elements.
- Linear search is used to find the correct position.

Insertion Sort

procedure *insertion sort*
($a_1,\ldots,a_n$:
real numbers with $n \geq 2$)
for j := 2 to n
i := 1
while $a_j > a_i$
i := i + 1
$m := a_j$
for k := 0 to $j - i - 1$
$a_{j-k} := a_{j-k-1}$
$a_i := m$
{Now $a_1,\ldots,a_n$ is in increasing order}

Example

- Apply insertion sort to 3, 2, 4, 1, 5
- First compare 3 and 2 → *2, 3*, 4, 1, 5
- Next, insert 3rd item, 4>2, 4>3 → *2, 3, 4*, 1, 5
- Next, insert 4th item, 1<2 → *1, 2, 3, 4*, 5
- Next, insert 5th item, 5>1, 5>2, 5>3, 5>4→*1, 2, 3, 4, 5*

Insertion Sort Animation

Outline

- Introduction to Algorithms
- Searching Algorithms
- Sorting Algorithms
- **Greedy Algorithms**

Optimization Problems

- *Optimization problems* minimize or maximize some parameter over all possible inputs.
- Among the many optimization problems we will study are:
  - Finding a route between two cities with the smallest total mileage.
  - Determining how to encode messages using the fewest possible bits.
  - Finding the fiber links between network nodes using the least amount of fiber.

Greedy algorithm

- Many algorithms are designed to solve optimization problems
- Greedy algorithm:
  - Simple and naïve
  - Select the best choice at each step, instead of considering all sequences of steps
  - Once find a feasible solution
  - Either prove the solution is optimal or show a counterexample that the solution is non-optimal

Example

- Given n cents change with quarters, dimes, nickels and pennies, and use the least total number of coins
- Say, 67 cents
- Greedy algorithm
  - First select a quarter (leaving 42 cents)
  - Second select a quarter (leaving 17 cents)
  - Select a dime (leaving 7 cents)
  - Select a nickel (leaving 2cents)
  - Select a penny (leaving 1 cent)
  - Select a penny

Greedy change-making algorithm

procedure *change*(c_1, c_2, …, c_n: values of denominations of coins, where $c_1>c_2>\ldots>c_n$; n: positive integer)

```
for i:=1 to r
    while n≥c_i then
        add a coin with value c_i to the change
        n:=n- c_i
    end
```

Example

- Change of 30 cents
- If we use only quarters, dimes, and pennies (no nickels)
- Using greedy algorithm:
  - 6 coins: 1 quarter, 5 pennies
  - Could use only 3 coins (3 dimes)

Next class

- Topic: The Growth of Functions and Complexity
- Pre-class reading: Chap 3.2-3.3