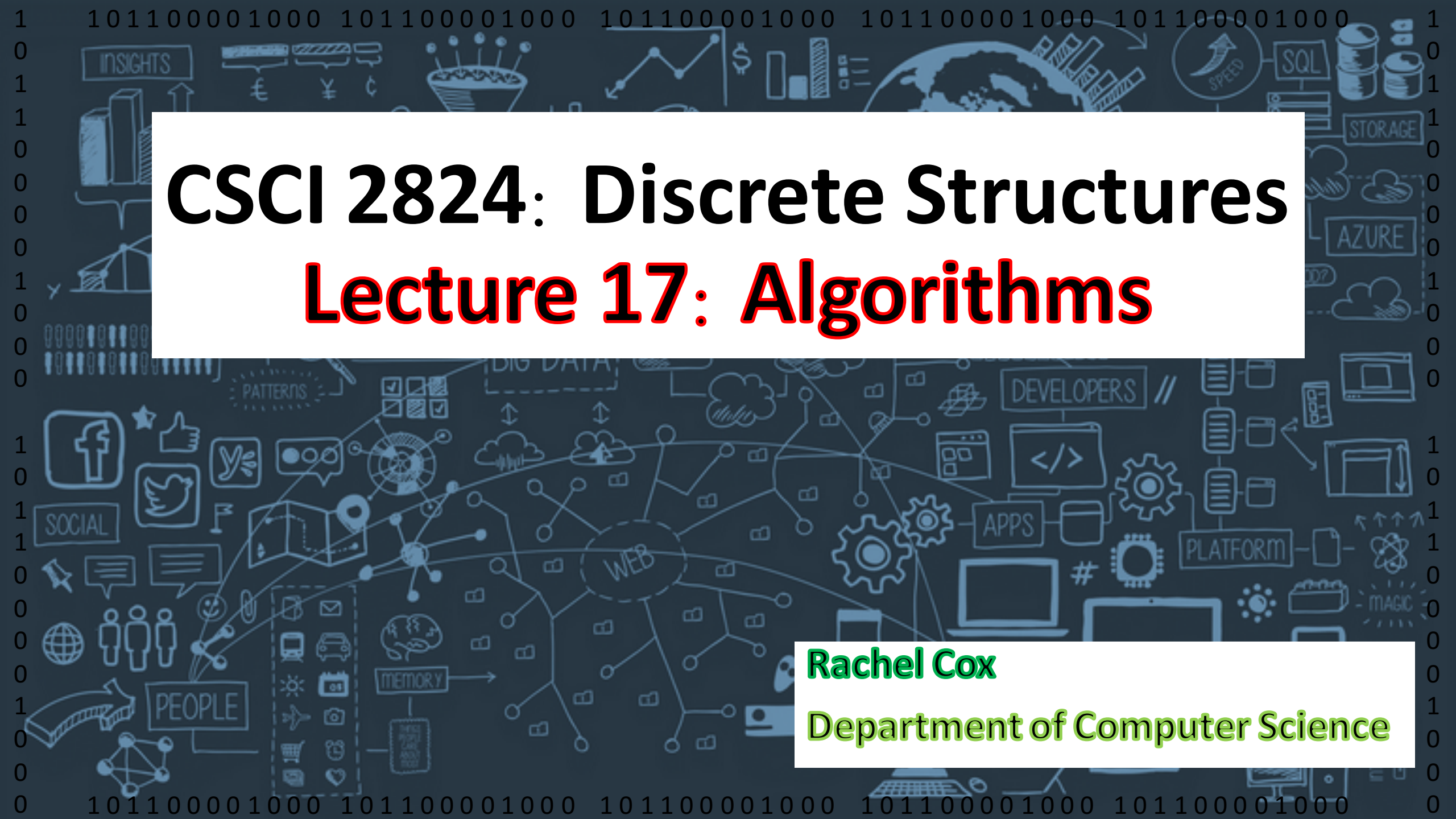


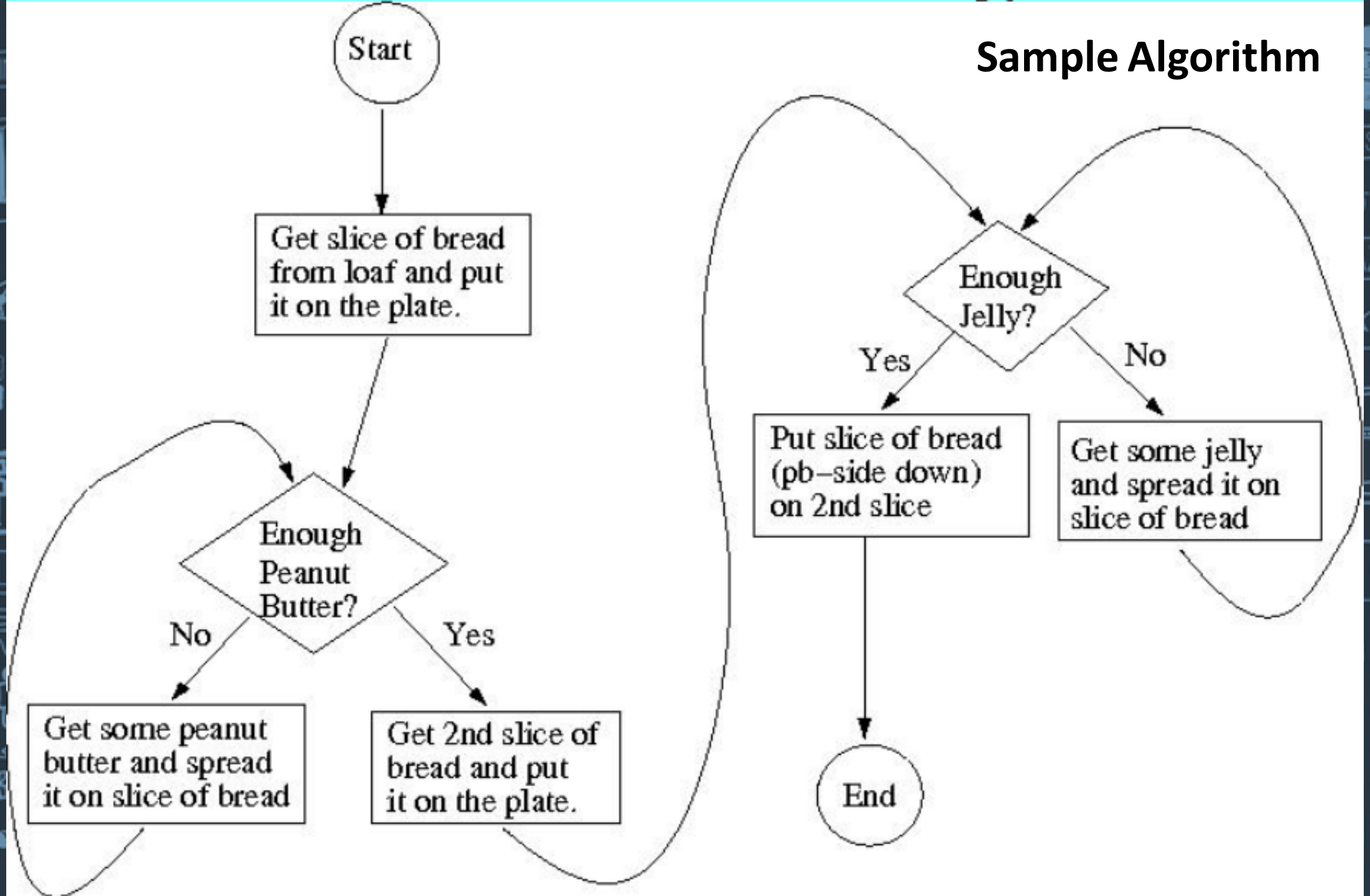
CSCI 2824: Discrete Structures

Lecture 17: Algorithms

Rachel Cox

Department of Computer Science

Sample Algorithm

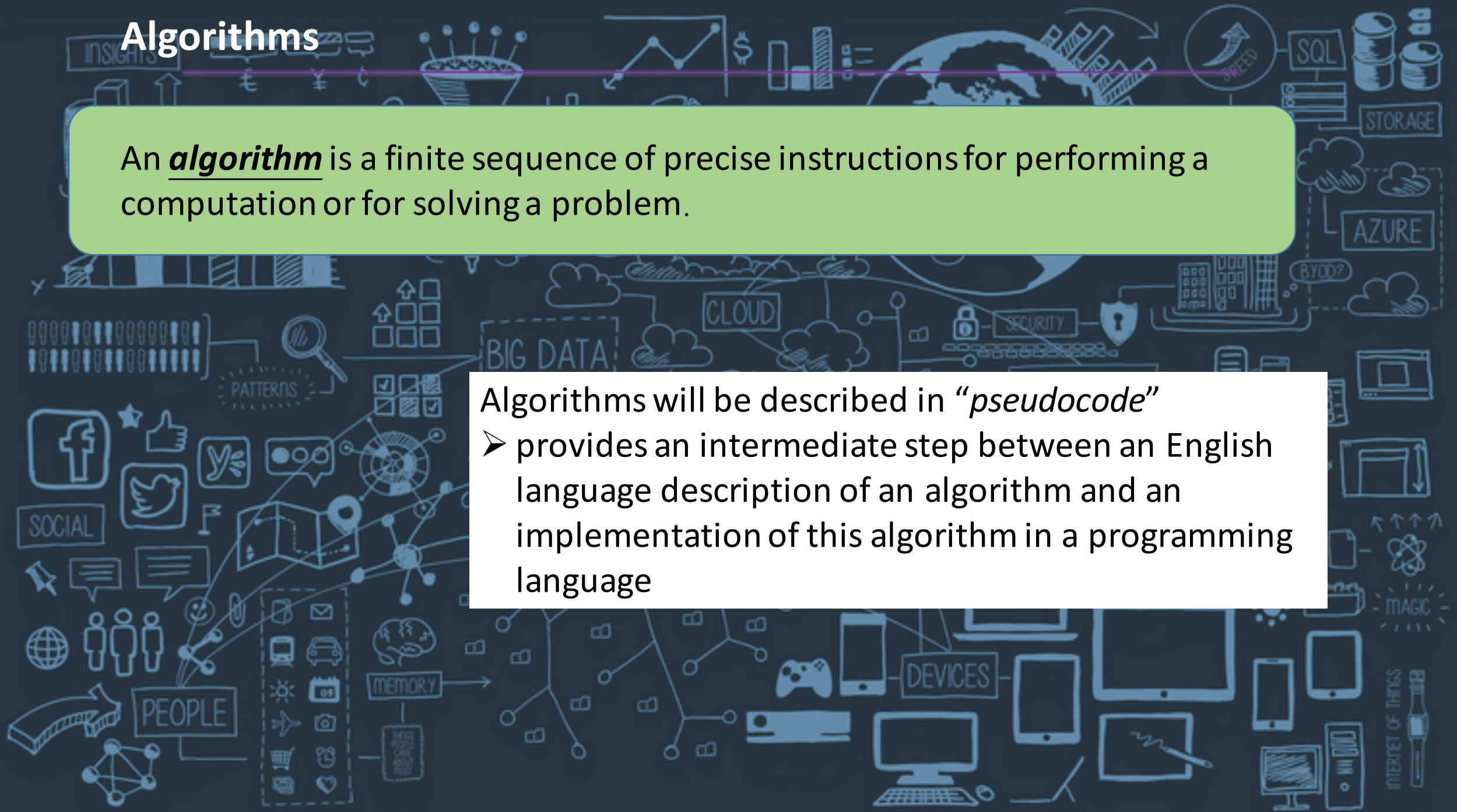


Algorithms

An ***algorithm*** is a finite sequence of precise instructions for performing a computation or for solving a problem.

Algorithms will be described in "*pseudocode*"

- provides an intermediate step between an English language description of an algorithm and an implementation of this algorithm in a programming language



Algorithms – Maximum Element

Task: Find the largest element in a finite sequence

Example: Given the sequence $\{1, 3, 12, 8, 2, 47, 58, 54, 7\}$ return 58

procedure $\max(a_1, a_2, \ldots, a_n)$

$\max := a_1$

for $i := 2$ to n

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

return $\max$

1. Initialize temporary max to the 1st term in the sequence
2. Compare the 2nd term in sequence to max. If it's larger, set max to this integer
3. Repeat previous step if there are more terms in sequence
4. Stop when there are no more terms.

Algorithms – Properties

Input: An algorithm has inputs from a particular set.

Output: From each set of inputs, the algorithm produces outputs. The outputs are the solution to the problem.

Definiteness: The steps in the algorithm are defined precisely.

Correctness: The algorithm should produce the correct output for each set of inputs.

Finiteness: An algorithm should terminate in finite time.

Generality: An algorithm should be applicable for all problems of the desired form.

Algorithms – Linear Search

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

Example: Find 34 in the sequence $\{1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89\}$

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

$i := 1$

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

$i := i + 1$

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

else *location* $:= -1$

return *location*

1. Compare x with a_1. If $x = a_1$, the solution is the location of a_1 (namely 1)
2. When $x \neq a_1$, compare x with a_2.
3. Continue comparing x successively with each term on the list until a match is found.
4. If entire list is searched without locating x, return -1.

Algorithms – Linear Search

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

Example: Find 34 in the sequence $\{1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89\}$

$x = 34$

check 1st element $34 = 1$ no

2nd element $34 = 1$ no

⋮

Algorithms – Binary Search

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

Assuming you have an ordered list.

Example: Find 34 in the sequence $\{1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89\}$

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

$i := 1$ # i is left endpoint of search interval

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

while $i < j$

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

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

if $x = a_i$ **then** *location* $:= i$, **else** *location* $:= -1$

return *location*

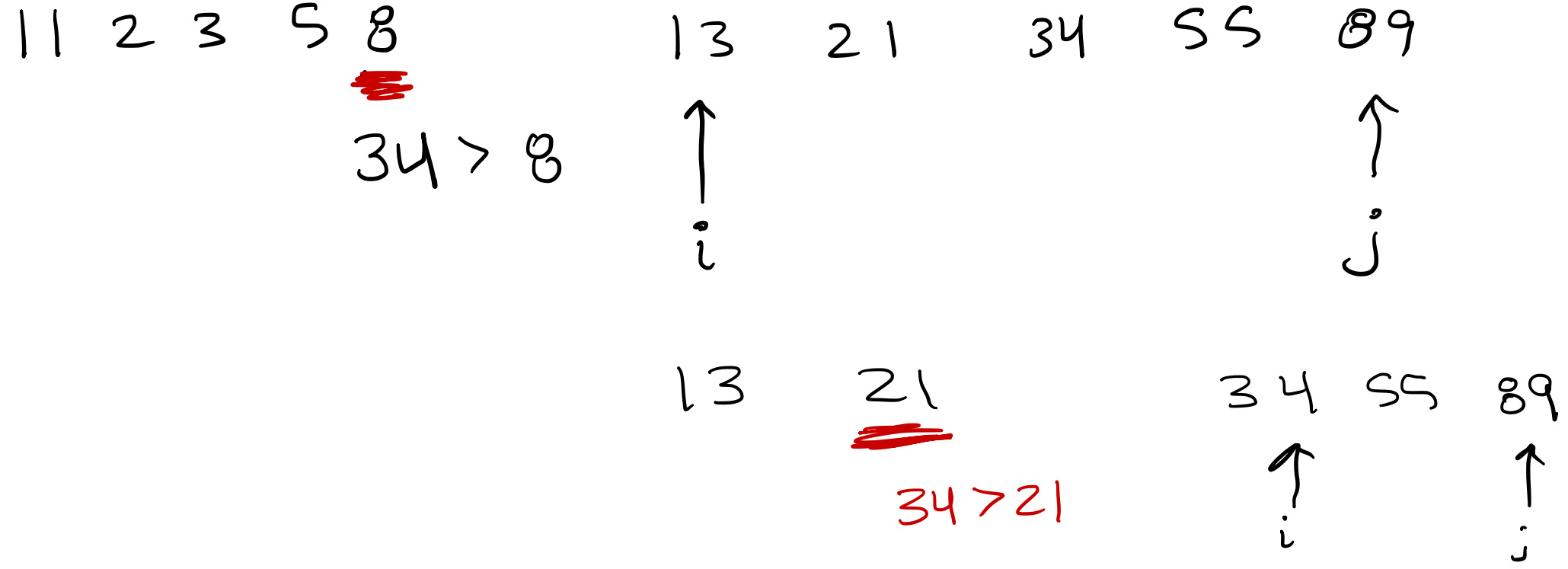
1. Cut list in half.
2. Compare x with last term in first half of list.
3. If x is larger, x must be in second half of list (assuming it's in list).
4. Repeat
5. If entire list is searched without locating x, return 0.

Algorithms – Binary Search

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

$m = \frac{i+j}{2}$

Example: Find 34 in the sequence $\{1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89\}$



Algorithms – Sorting

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

e.g. Alphabetizing

- Ordering a list can be the goal in-and-of itself.
- Ordering a list can make other tasks easier.

Algorithms – Bubble Sort

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

Example: Sort the list $\{3, 2, 4, 1, 5\}$

procedure BubbleSort($a_1, a_2, \ldots, a_n$)

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

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

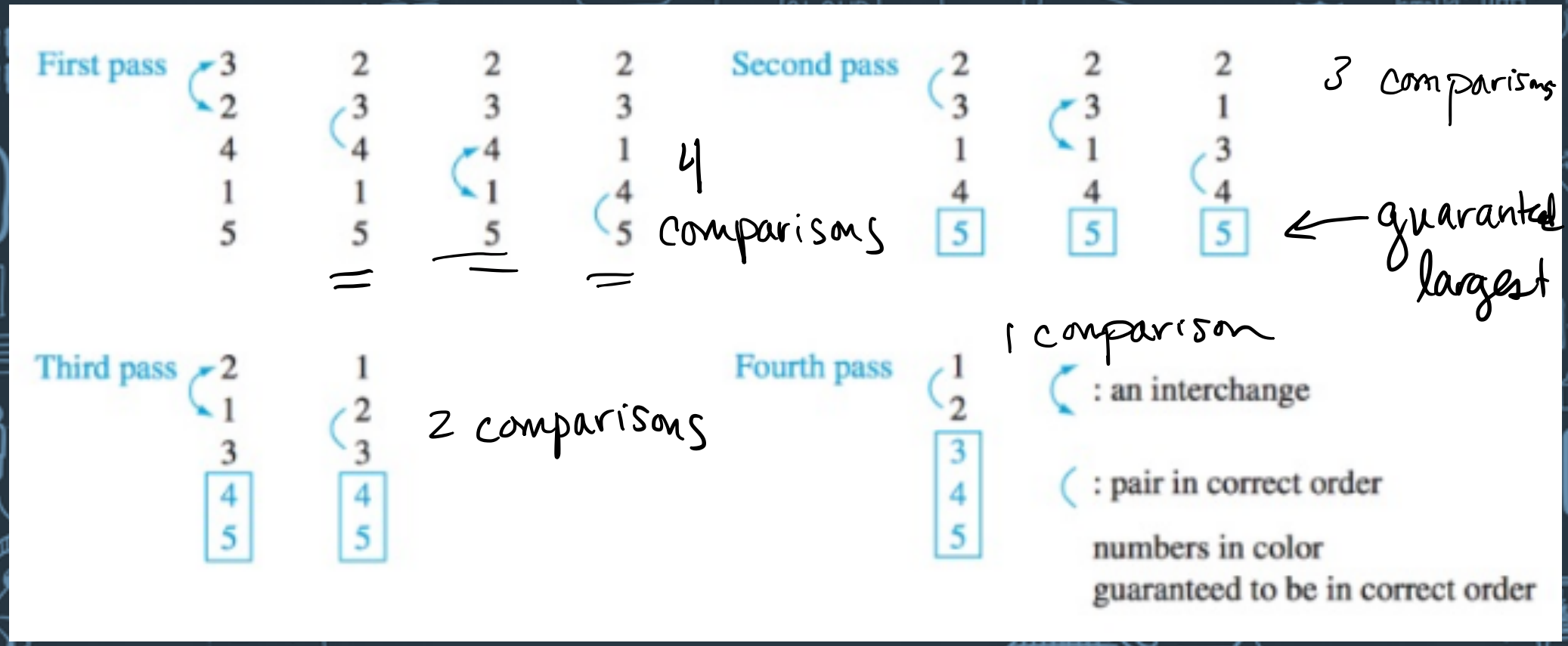
    if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}

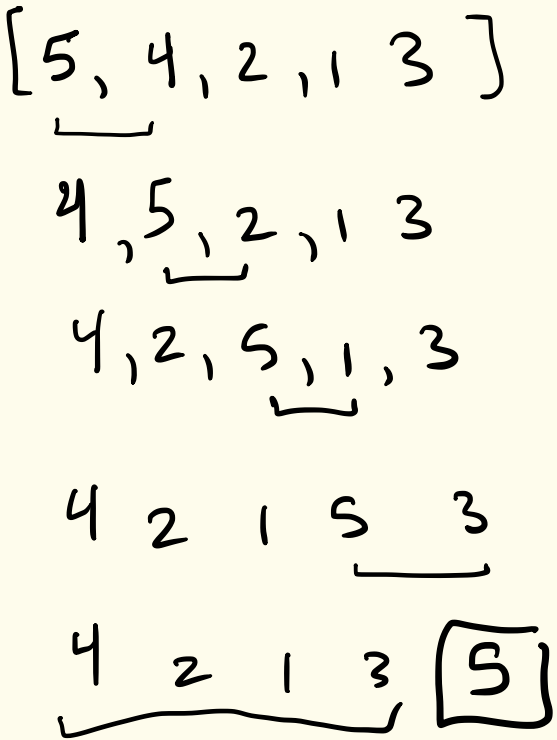
1. Make passes through the list, interchanging adjacent pairs that are in the wrong order.
2. Repeat until the list is sorted.
3. Large elements sink to bottom, small elements bubble to top.

Algorithms – Bubble Sort

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

Example: Sort the list $\{3, 2, 4, 1, 5\}$

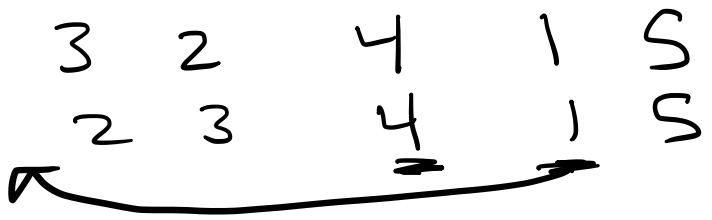



[5, 4, 2, 1 3]
4, 5, 2, 1 3
4, 2, 5, 1, 3
4 2 1 5 3
4 2 1 3 5

Algorithms – Insertion Sort

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

Example: Sort the list $\{3, 2, 4, 1, 5\}$



procedure *insertion sort*($a_1, a_2, \ldots, a_n$: 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** $j - i - 1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
{$a_1, \ldots, a_n$ is in increasing order}

1. Make passes through the list, successively inset next unsorted element into the already sorted front end of the list.
2. Repeat until the list is sorted.

Algorithms – Insertion Sort

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

Example: Sort the list $\{3, 1, 5, 4, 2\}$

- $\{\mathbf{3}, \mathit{1}, 5, 4, 2\} \Rightarrow \{\mathbf{1}, \mathbf{3}, 5, 4, 2\}$
- $\{\mathbf{1}, \mathbf{3}, \mathit{5}, 4, 2\} \Rightarrow \{\mathbf{1}, \mathbf{3}, \mathbf{5}, 4, 2\}$
- $\{\mathbf{1}, \mathbf{3}, \mathbf{5}, \mathit{4}, 2\} \Rightarrow \{\mathbf{1}, \mathbf{3}, \mathbf{4}, \mathbf{5}, 2\}$
- $\{\mathbf{1}, \mathbf{3}, \mathbf{4}, \mathbf{5}, 2\} \Rightarrow \{\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}\}$

Algorithms – Greedy Algorithms

Task: Find a solution that minimizes or maximizes some parameter.

★ Moodle Quiz !

Applications:

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

Greedy Algorithms select the locally optimal choice at each step, the goal is global optimization.

Not guaranteed to find the overall optimal solution….must check after a solution has been found.

Algorithms – Greedy Algorithms

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

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

67 cents

- 1 quarter
  Check how left : 42¢
- 1 quarter
  left with: 17¢
- 1 dime
  left with: 7¢
- 1 nickel
  left: 2¢
- 2 pennies

Algorithms – Greedy Algorithms

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

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

procedure Change$(c_1, c_2, \ldots, c_r)$

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

$n := n - c_i$

Algorithms – 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

Algorithms – Greedy Algorithms

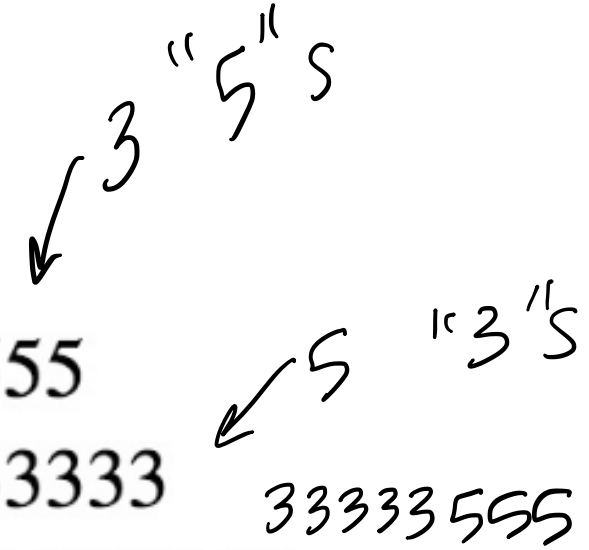
Task: Given a number N, find the largest Decent Number with N digits. If no such number exists, return -1.

A **Decent Number** is a number that includes only 3's and 5's:

1. The number of 3's is divisible by 5
2. The number of 5's is divisible by 3

Examples:

- The largest 3-Digit Decent Number is 555
- The largest 5-Digit Decent Number is 33333
- The largest 8-Digit Decent Number is 55533333



What we've done:

- ❖ Search Algorithms
- ❖ Sorting Algorithms
- ❖ Greedy Algorithms
- ❖ PB&J algorithms

Next:

Complexity