Recursion

Michael Ernst
CSE 190p
University of Washington

To seal: moisten flap, fold over, and seal
Three recursive algorithms

- Sorting
- GCD (greatest common divisor)
- Exponentiation

Used in cryptography, which protects information and communication
Sorting a list

• Python’s `sorted` function returns a sorted version of a list.
  \[
  \text{sorted}([4, 1, 5, 2, 7]) \implies [1, 2, 4, 5, 7]
  \]

• How could you implement `sorted`?

• Idea ("quicksort", invented in 1960):
  – Choose an arbitrary element (the “pivot”)
  – Collect the smaller items and put them on its left
  – Collect the larger items and put them on its right
First version of quicksort

```python
def quicksort(thelist):
    pivot = thelist[0]
    smaller = [elt for elt in thelist if elt < pivot]
    larger = [elt for elt in thelist if elt > pivot]
    return smaller + [pivot] + larger

print quicksort([4, 1, 5, 2, 7])
```

There are three problems with this definition

Write a test case for each problem
def quicksort(thelist):
    if len(thelist) < 2:
        return thelist
    pivot = thelist[0]
    smaller = [elt for elt in thelist if elt < pivot]
    larger = [elt for elt in thelist if elt > pivot]
    return quicksort(smaller) + [pivot] + quicksort(larger)

How can we fix the problem with duplicate elements?
Handling duplicate pivot items

def quicksort(thelist):
    if len(thelist) < 2:
        return thelist
    pivot = thelist[0]
    smaller = [elt for elt in thelist if elt < pivot]
    pivots = [elt for elt in thelist if elt == pivot]
    larger = [elt for elt in thelist if elt > pivot]
    return quicksort(smaller) + pivots + quicksort(larger)

def quicksort(thelist):
    if len(thelist) < 2:
        return thelist
    pivot = thelist[0]
    smaller = [elt for elt in thelist[1:] if elt <= pivot]
    larger = [elt for elt in thelist if elt > pivot]
    return quicksort(smaller) + [pivot] + quicksort(larger)
GCD (greatest common divisor)

gcd(a, b) = largest integer that divides both a and b
- gcd(4, 8) = 4
- gcd(15, 25) = 5
- gcd(16, 35) = 1

How can we compute GCD?
Euclid’s method for computing GCD
(circa 300 BC, still commonly used!)

\[
gcd(a, b) = \begin{cases} 
a & \text{if } b = 0 \\ 
gcd(b, a) & \text{if } a < b \\ 
gcd(a-b, b) & \text{otherwise} \\ 
\end{cases}
\]
Python code for Euclid’s algorithm

def gcd(a, b):
    if b == 0:
        return a
    if a < b:
        return gcd(b, a)
    return gcd(a-b, b)
Exponentiation

Goal: Perform exponentiation, using only addition, subtraction, multiplication, and division. (Example: $3^4$)

def exp(base, exponent):
    """Exponent is a non-negative integer"""
    if exponent == 0:
        return 1
    return base * exp(base, exponent - 1)

Example:
exp(3, 4)
3 * exp(3, 3)
3 * (3 * exp(3, 2))
3 * (3 * (3 * exp(3, 1)))
3 * (3 * (3 * (3 * exp(3, 0))))
3 * (3 * (3 * (3 * 1)))
Faster exponentiation

Suppose the exponent is even. Then, \( \text{base}^{\text{exponent}} = (\text{base} \times \text{base})^{\text{exponent}/2} \)

Examples: 3\(^4\) = 9\(^2\)  \quad 9\(^2\) = 81\(^1\)  \quad 5\(^{12}\) = 25\(^6\)  \quad 25\(^6\) = 625\(^3\)

New implementation:

```python
def exp(base, exponent):
    """Exponent is a non-negative integer""
    if exponent == 0:
        return 1
    if exponent % 2 == 0:
        return exp(base*base, exponent/2)
    return base * exp(base, exponent - 1)
```
Comparing the two algorithms

<table>
<thead>
<tr>
<th>Original algorithm: 12 multiplications</th>
<th>Fast algorithm: 5 multiplications</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5^{12}$</td>
<td>$5^{12}$</td>
</tr>
<tr>
<td>$5 \times 5^{11}$</td>
<td>$(5 \times 5)^6$</td>
</tr>
<tr>
<td>$5 \times 5 \times 5^{10}$</td>
<td>$25^6$</td>
</tr>
<tr>
<td>$5 \times 5 \times 5 \times 5^9$</td>
<td>$(25 \times 25)^3$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5^0$</td>
<td>$625^3$</td>
</tr>
<tr>
<td>$5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 1$</td>
<td></td>
</tr>
<tr>
<td>$5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5$</td>
<td></td>
</tr>
<tr>
<td>$5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 25$</td>
<td></td>
</tr>
<tr>
<td>$5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 125$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$244140625$</td>
<td>$625 \times 625^2$</td>
</tr>
<tr>
<td></td>
<td>$625 \times 625 \times 625^1$</td>
</tr>
<tr>
<td></td>
<td>$625 \times 625 \times 625 \times 625^0$</td>
</tr>
<tr>
<td></td>
<td>$625 \times 625 \times 625 \times 1$</td>
</tr>
<tr>
<td></td>
<td>$625 \times 625 \times 625$</td>
</tr>
<tr>
<td></td>
<td>$625 \times 390625$</td>
</tr>
<tr>
<td></td>
<td>$244140625$</td>
</tr>
</tbody>
</table>

**Speed matters:** In cryptography, exponentiation is done with 600-digit numbers.
Recursion: base and inductive cases

• Recursion expresses the essence of divide and conquer
  – Solve a smaller subproblem, use the answer to solve the original problem

• A recursive algorithm always has:
  – a base case (no recursive call)
  – an inductive or recursive case (has a recursive call)

• What happens if you leave out the base case?
• What happens if you leave out the inductive case?
Recursion vs. iteration

• Any recursive algorithm can be re-implemented as a loop instead
  – This is an “iterative” expression of the algorithm
• Any loop can be implemented as recursion instead

• Sometimes recursion is clearer and simpler
  – Mostly for data structures with a recursive structure
• Sometimes iteration is clearer and simpler
More examples of recursion

• List algorithms: recursively process all but the first element of the list, or half of the list
• Map algorithms: search for an item in part of a map (or any other spatial representation)
• Numeric algorithms: Process a smaller value