

---

# Reductions, Self-Similarity, and Recursion

*Relations between problems*

---

Notes for CSC 100 - The Beauty and Joy of Computing  
The University of North Carolina at Greensboro

---

# Reminders

---

Blown to Bits: Chapter 4 discussion over the next week

Homework 3:

- Read assignment carefully to make sure you understand it
- Get started! Goal: At least watch the video by Friday

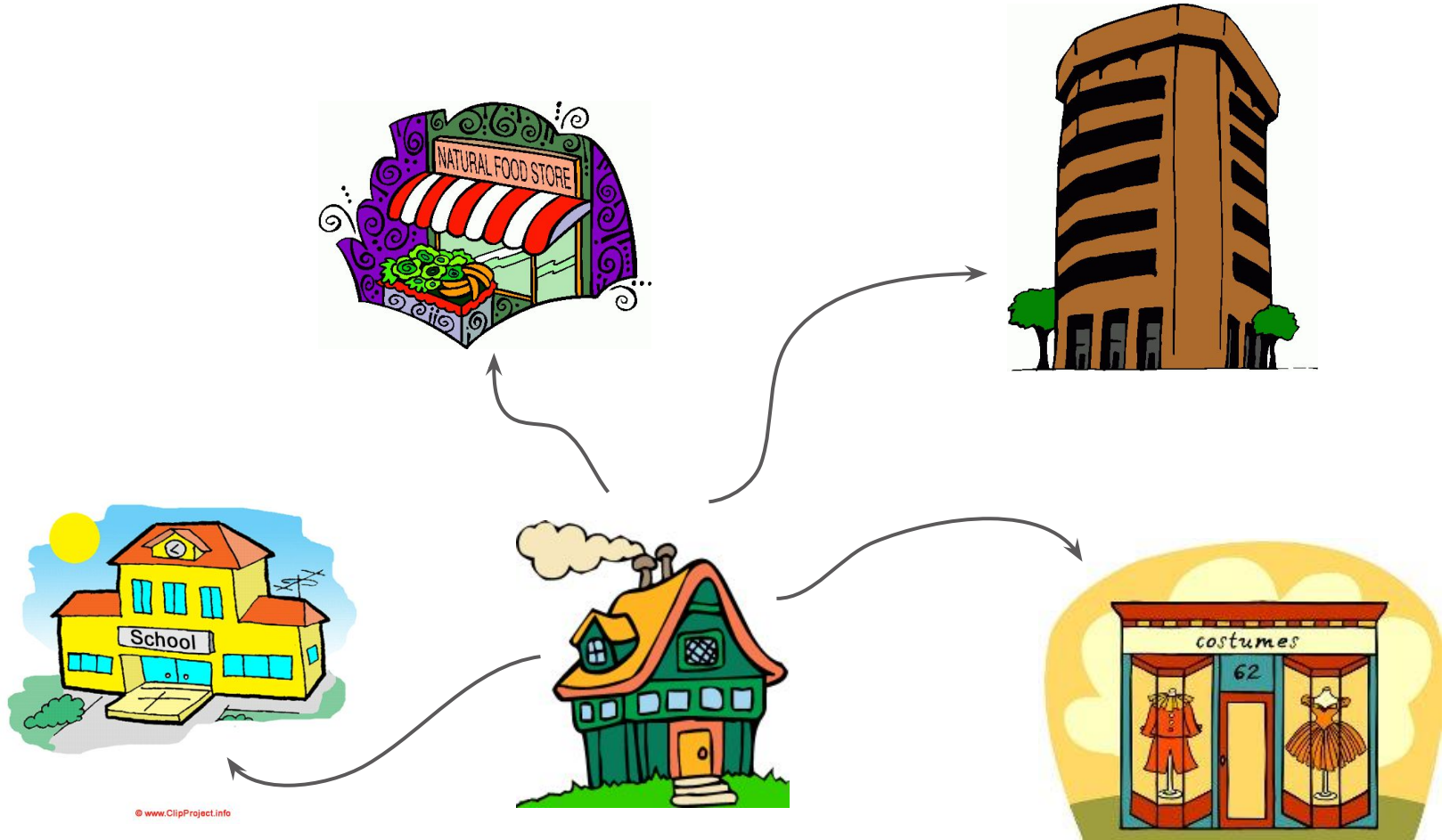
Projects: Think about ideas, talk to other students, ...

*We will discuss more about project ideas / teams on Wednesday*

---

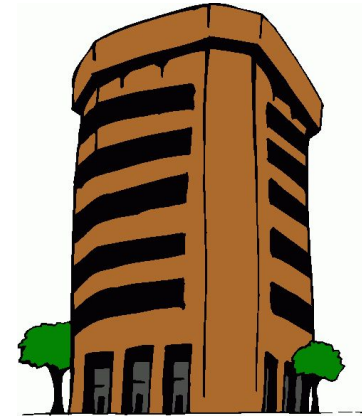
# Getting to places from my house...

---



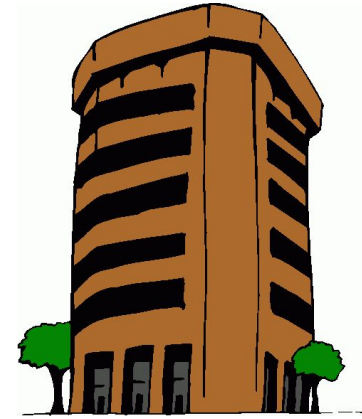
# Now I buy a new house!

---



# Get anywhere by first going to old house

---



# Things to notice...

---

- I can go anywhere from my new house by
- 1. Going to my old house
  - 2. Going to my destination from there
- What I want to do...*
- What I know how to do...*
-

# Things to notice...

---

I can go anywhere from my new house by

1. Going to my old house
2. Going to my destination from there

*What I want to do...*

*What I know how to do...*

Terminology: I have reduced the problem of traveling from my new house to the problem of traveling from my old house.

Important points:

- Solution is easy to produce (often easier than direct solution)
  - Solution is easy and compact to describe (especially with abstraction!)
  - Solution may not be the most efficient to execute
-

# Things to notice...

---

- I can go anywhere from my new house by
- 1. Going to my old house
  - 2. Going to my destination from there
- What I want to do...*
- What I know how to do...*
- 

Question: Is a reduction a property of problems or algorithms?

---



# Things to notice...

---

I can go anywhere from my new house by

1. Going to my old house
2. Going to my destination from there

*Problem*

*Problem*

Reductions are between problems

- The reduction operation is an algorithm
  - Abstraction: We don't care how the "known algorithm" works!
-

# The Basics

---

A reduction is using the solution of one problem (problem A) to solve another problem (problem B).

We say "problem B is reduced to problem A".

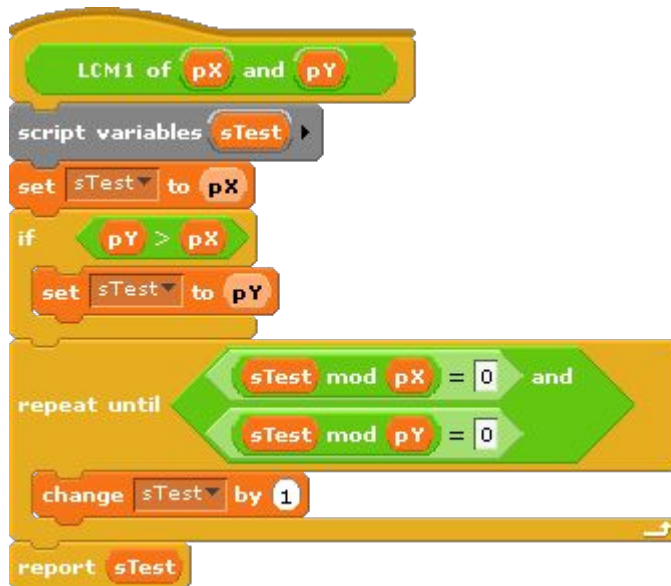
Reductions are a fundamental "big idea" in computer science

- Lots of types of reductions - you could spend a lifetime studying this!
  - Our reductions use a small amount of work in addition to a constant number of calls to problem A.
    - As a result, can say problem B is not much harder than problem A
    - True even if we don't know the most efficient way to solve problem A!
-

# An example from Mathematics

---

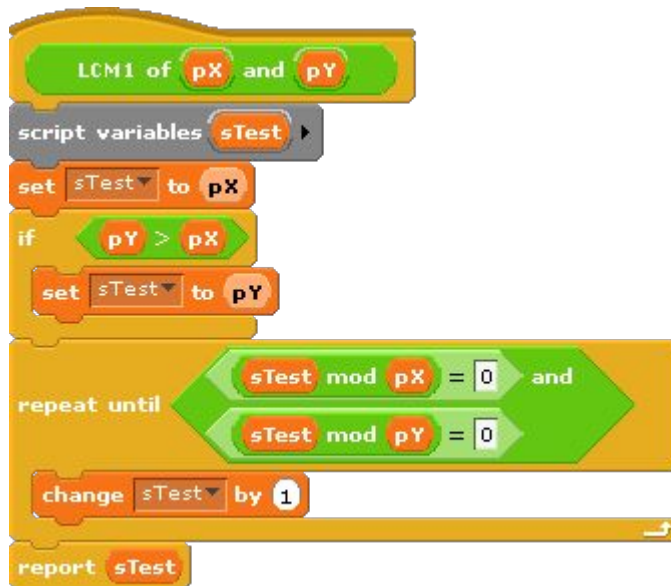
To find least common multiple (LCM):



# An example from Mathematics

---

To find least common multiple (LCM):



But if you already have GCD

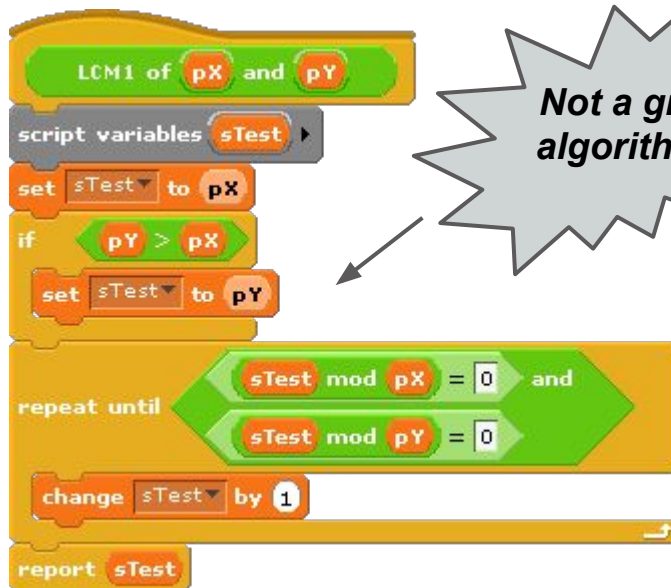


What have we done? We have reduced the problem of computing LCM to the problem of computing GCD.

---

# An example from Mathematics

To find least common multiple (LCM):



```
LCM1 of pX and pY
script variables sTest
set sTest to pX
if pY > pX
  set sTest to pY
repeat until (sTest mod pX = 0 and sTest mod pY = 0)
  change sTest by 1
report sTest
```

Not a great algorithm...

But if you already have GCD



```
LCM2 of pX and pY
report pX * pY / GCD of pX and pY
```

What have we done? We have reduced the problem of computing LCM to the problem of computing GCD.

So: LCM is no harder computationally than GCD. And remember... Euclid's algorithm is a very efficient GCD algorithm!

# Similarity and Self-Similarity

---

Reducing LCM to GCD identifies similarities between the two problems.

Many problems are structured so that solutions are "self-similar"  
- large solutions contain solutions to smaller versions of the same problem!

Example: Think about adding up the numbers in an  $n$ -element list. Adding up the first  $n-1$  elements is a smaller version of the same problem!

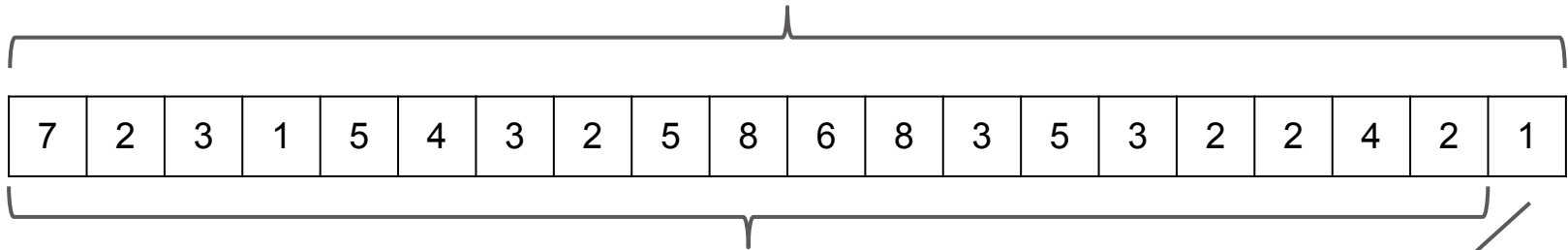
An algorithm can solve a large problem by breaking it down to smaller versions of the *same* problem - this is called recursion.

---

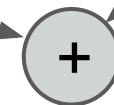
# Example: Adding up a list

---

Sum of 20 items ( = 76)



Sum of 19 items ( = 75)

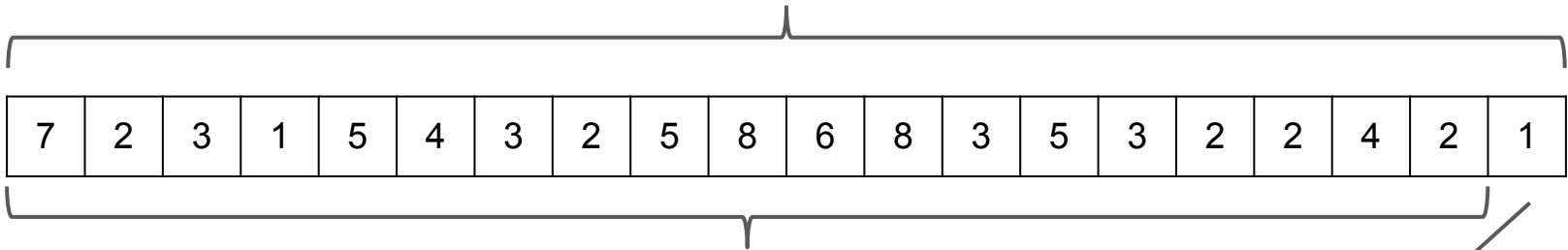


76

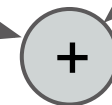
# Example: Adding up a list

---

Sum of 20 items (= 76)



Sum of 19 items (= 75)



76

```
def sum_of_first(data, size):  
    if (size == 0):  
        return 0  
  
    subProblem = sum_of_first(data, size-1)  
    return subProblem + data[size-1]
```



# Breaking it down

---

```
def sum_of_first(data, size):  
    if (size == 0):  
        return 0
```

Base case: Handling smallest case directly

```
    subProblem = sum_of_first(data, size-1)  
    return subProblem + data[size-1]
```

Recursive case: Solving a smaller version of the same problem.

Constant amount of work to use answer from subproblem to compute answer to overall problem.

---

# Breaking it down

---

## Workhorse Function

```
def sum_of_first(data, size):  
    if (size == 0):  
        return 0
```

Base case: Handling smallest case directly

```
    subProblem = sum_of_first(data, size-1)  
    return subProblem + data[size-1]
```

Recursive case: Solving a smaller version of the same problem.

Constant amount of work to use answer from subproblem to compute answer to overall problem.

## Driver Function

```
def sum_of(data):  
    return sum_of_first(data, len(data))
```

Driver function: sets up first call to recursion

---

# Another example: Sorting

---

"Selection sort" from algorithms lab:

```
def sort(data):  
    for left in range(len(data), 1, -1):  
        maxPos = max_pos_from_first(data, left)  
        swap(data, maxPos, left-1)
```

# Another example: Sorting

---

"Selection sort" from algorithms lab:

```
def sort(data):
    for left in range(len(data), 1, -1):
        maxPos = max_pos_from_first(data, left)
        swap(data, maxPos, left-1)
```

Recursive version:

```
def recursive_sort(data, size):
    if (size > 1):
        maxPos = max_pos_from_first(data, size)
        swap(data, maxPos, size-1)
        recursive_sort(data, size-1)
```

Base case: One item - nothing to do!

Setting up recursion: Swap max item to last position

Recursion: Sort all the rest

Note the elegance of the recursive description: “If there’s something to sort, put the largest item at the end and then sort the rest.”

---

# Summary

---

Finding relations between problems can simplify solutions:

- Sometimes relations between different problems (reductions)
- Sometimes relation to smaller version of the same problem (recursion)

What you should know:

- Recognize reductions and recursion
- Understand the basic principles

We will explore this more in a lab!

---