Intro to Computer Science

Previous

- Recursion

Next

- Sorting
- Backtracking

Readings	
Gaddis	• Chapter 12

Recursion (is simple!)

- Essentially two aspects to a recursive function
  1. A base case
  2. A set of rules that reduce all other cases toward the base case
- The "difficult" part is learning to *think* this way

Appreciating recursion

- To appreciate recursion you must separate the *physical* implementation from the *conceptual* algorithm
- Recursion is an incredible tool to solve problems
  - Irrespective of whether you use it to implement those solutions!
- **Be patient**: recursion takes time to master

Sorting

- Sorting (along with searching) are big deals in computer science
  - One of the most extensively researched subjects in the field
- There are dozens of methods to do so
- Many of these methods benefit from recursion ☺

Selection sort: in words

The long

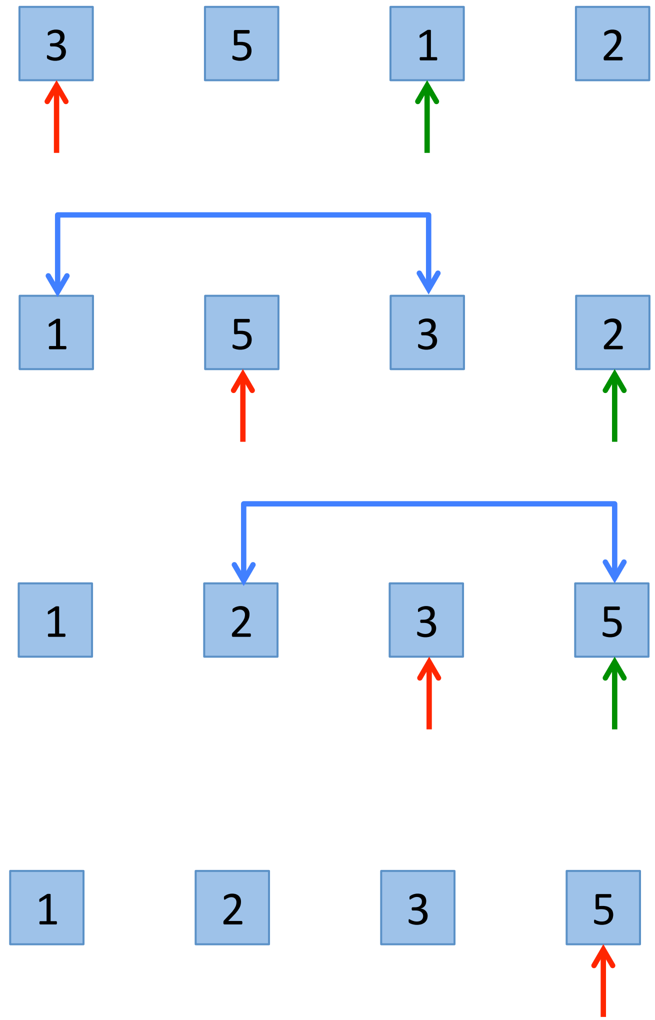
- Start a “pointer” at the head of the list
- Swap the smallest element in the list with the pointer
- Increment the pointer
- Repeat

The short

- Move the smallest elements toward the head

Selection sort: in pictures

1. Start with the head
2. Find the smallest numbers in the tail
3. Swap!
4. Update the pointer



Complexity

Recall our earlier discussion on complexity:

- A measure of the amount of work an algorithm has to do to solve a problem
  - Given an input of size *n*, the number of steps required to solve the problem, with respect to *n*

Data type	Complexity
List	Directly proportional to the size of the list
Binary tree	Logarithmically proportional to the size of the list (reduced our search space by half)

Selection sort: runtime

- Sequential sort loops over the entire list
  1. As it updates the head
  2. To find the minimum element in the tail
- The catch: the minimum element search happens within the list walk:

```
for head_pointer in list:
    for i in tail(list):
```

- For *n* elements, we do approximately *n* comparisons
  - *Quadratic performance*

This is bad

Merge sort: in words

The long

- Split the list into smaller sublists
- Sort the sublists
- Merge the sublists back into the original list

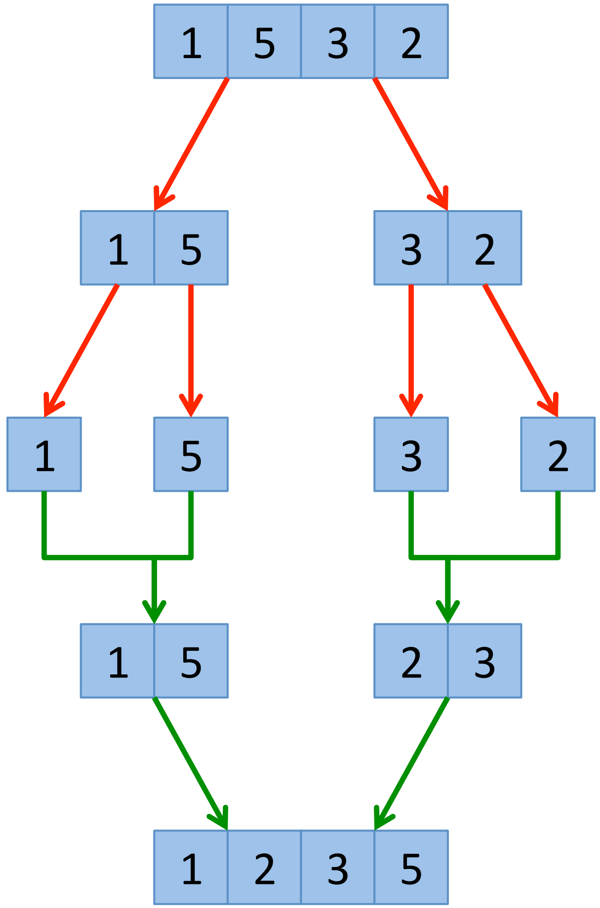
The short

- *Divide and conquer*

You know what that means: recursion!

Merge sort: in pictures

- Recursively split the list into sublists
- Single elements are the base case
- Merge the elements in order until we've recreated the original list



The merge

- Takes two lists, outputs a single sorted list
- Relatively simple, especially given that its input lists are already sorted
  - Complexity proportional to the combined size of the lists

Merge sort: runtime

- Merging the data takes times proportional to the number of items (linear)
- The binary tree means there's about half the number of merges as there are elements
  - *linearithmic time*

A battle of sorts

Backtracking

- Imagine you have a problem with a bunch of potential solutions – which one do you take?
- You could
  - Enumerate all potential solutions, or
  - Incrementally try them until you find the right one
- Backtracking is the latter
- When combined with recursion, it's amazing

The algorithm

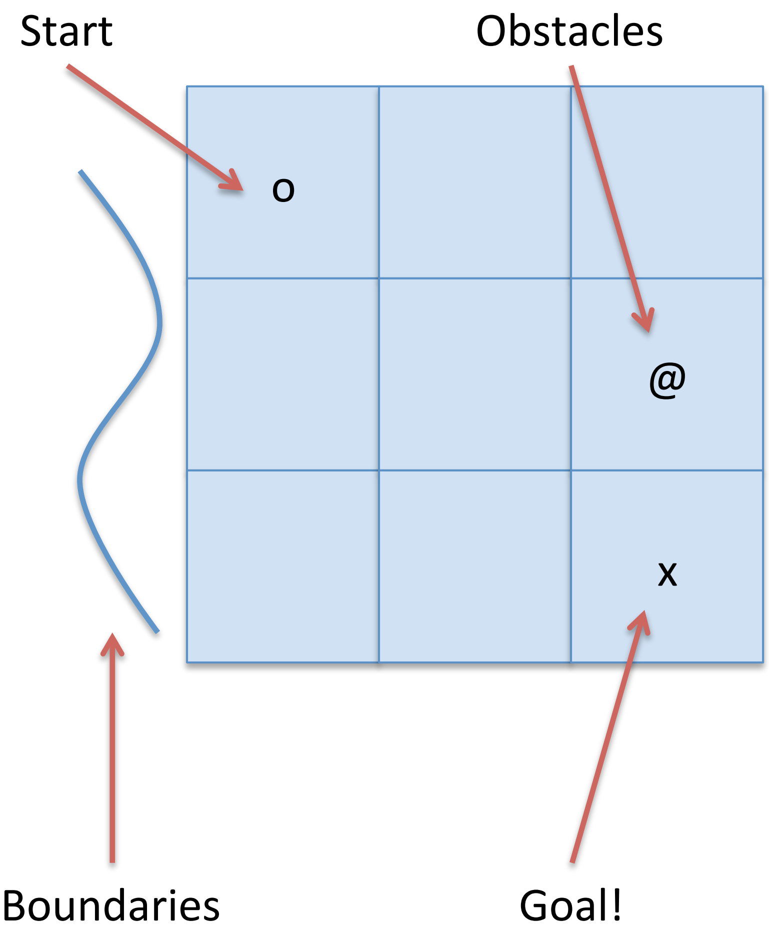
- Backtracking has a general algorithmic form:
  - accept: have we solved the problem?
  - reject: can we say that this is definitely not a solution?
  - move: step toward a potential solution
  - recurse

Solving a maze

- From the start of the board, there are several potential solutions
- The naïve approach is to try every one, from the start of the board
- Backtracking allows us to incrementally try potentials
  - *And remember our work*

The setup

- Given a "board" ☺
- From a given starting position (x_1, y_1), find a path to the solution position (x_2, y_2)



Pruning search space

- Consider all possible paths from start to finish
- Naïve method searches all paths
  – Learns nothing from mistakes
- Backtracking effectively reduces search space by *pruning* paths known to be unproductive
- Essentially performs a *depth-first* search

The algorithm

Given a board, and a (current) position:

- out-of-bounds? false
- goal? true
- obstacle? false
- *recurse* with an update position
  - One of our neighbors