VILLANOVA UNIVERSITY
IGNITE CHANGE. GO NOVA.

CSC 1052 – Algorithms & Data Structures II: Recursion

Professor Henry Carter
Spring 2017

Recap

- Stacks provide a LIFO ordered data structure
- Implementation tradeoffs between arrays and linked lists typically involve exchanging speed and memory management
- Example applications include:
  - Saving state
  - Processing nested data
  - Backtracking

Recursion

- Solving a problem by solving smaller versions of the same problem
- In programming: methods calling themselves
- Examples:
  - Russian dolls
  - File systems
  - Math functions

Recursive Definitions

- Define an operation using itself
- Simple to develop and understand
- Hard to measure
- Example: Factorial

Factorial

- How would you write it?

- A recursive definition

Input Constraints

- Recursive definitions must include an initial condition
- Input values outside of the initial condition are typically invalid or cause errors
- Your recursive algorithms must also respect these initial conditions

From definitions to algorithms

- Given these definitions of the function, write an algorithm for solving given n
- Mathematical definition only defines the structure, not the steps to solve
- We'll write your first recursive algorithm in Java

Three Pieces

- Every Recursive algorithm needs three pieces:
  - A base case
  - A check for the base case
  - A recursive case

Three Pieces

- Factorial:
  - Base case
  - Check
  - Recursive case

Factorial in Java

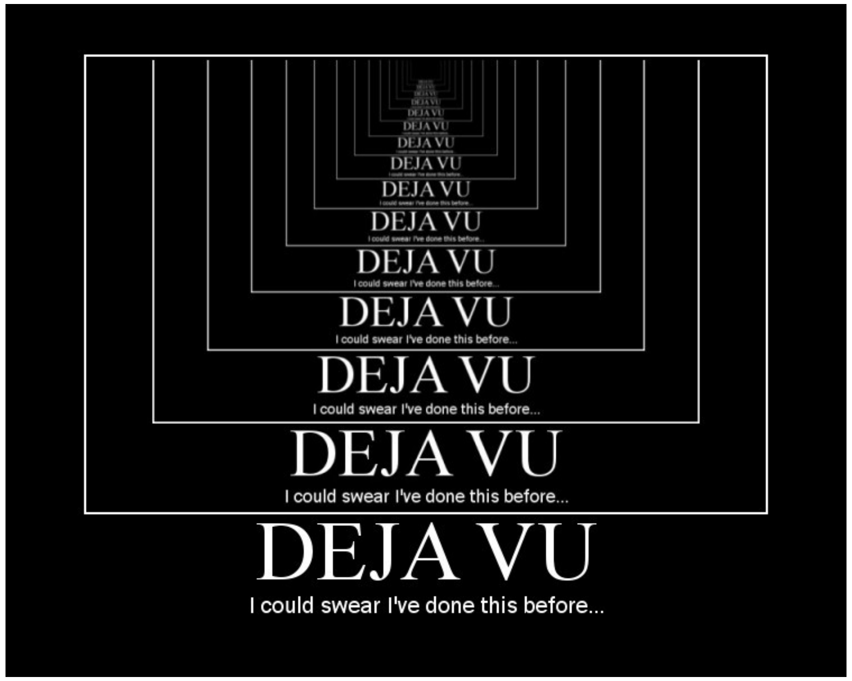
Back to the board: non-recursive

- Can we implement the same thing without recursion?

- Which is better?

The call stack

- Methods invoked have state that is stored on a system call stack
- More method calls == more memory usage
- This is the hidden cost of recursion!



Write your own!

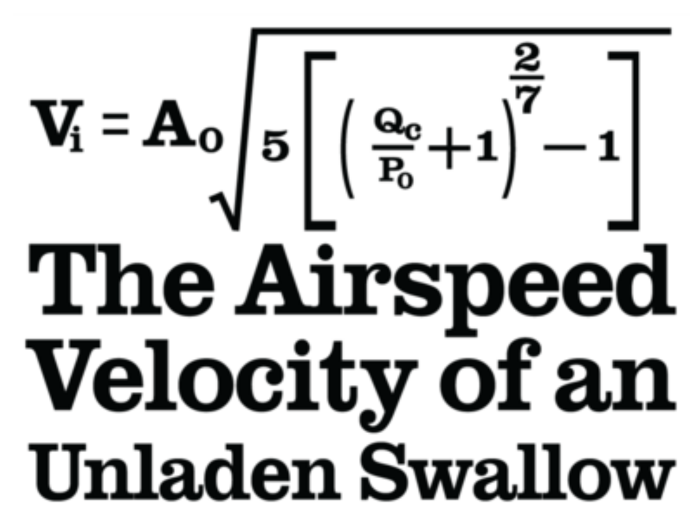
- Multiplication of two integers
- Start with a recursive definition
- Write a recursive algorithm

Verify the algorithm

- Ensuring the recursive algorithm is correct for all inputs seems daunting
- Three pieces:
  - Base case check
  - Base case operation
  - Recursive operation
- Three questions help to verify these pieces are correct

Three Questions

- Is the base case correct?
- Does each recursive call lead towards the base case?
- Assuming the smaller versions are correct, is the next recursive version correct?
- (For those who know inductive proofs, this closely matches that proof style)

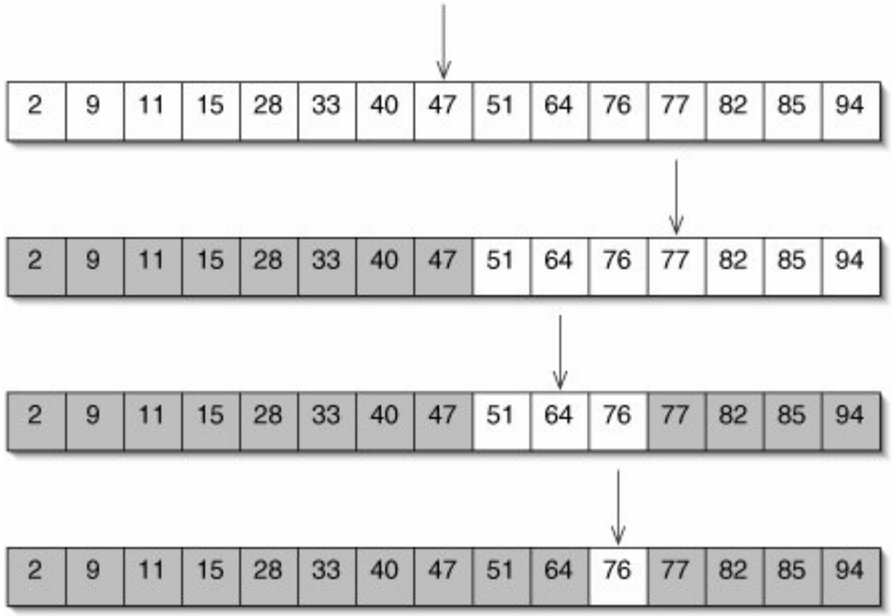


Common Mistakes

- Infinite recursion
- Inefficient recursion
- Incorrect control logic

Example Algorithm

- Recall: Binary search
- Highly amenable to a recursive definition
    - Why?
- Implemented on an array



Processing Arrays Recursively

- Recursive parameters:
- Base case:
- What scope must we declare the array in?

Binary Search Logic

- Check the middle at every step
- Three options:
  - Found?
  - Less?
  - Greater?
- Assume the list is already sorted

Binary Search Example

target: 20

first=0 last=7

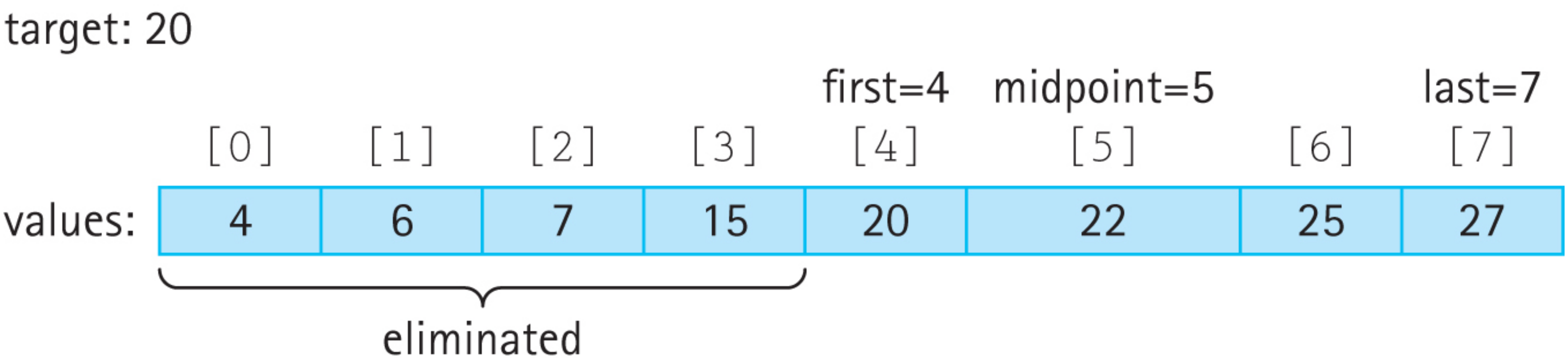
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
values:	4	6	7	15	20	22	25	27

Binary Search Example

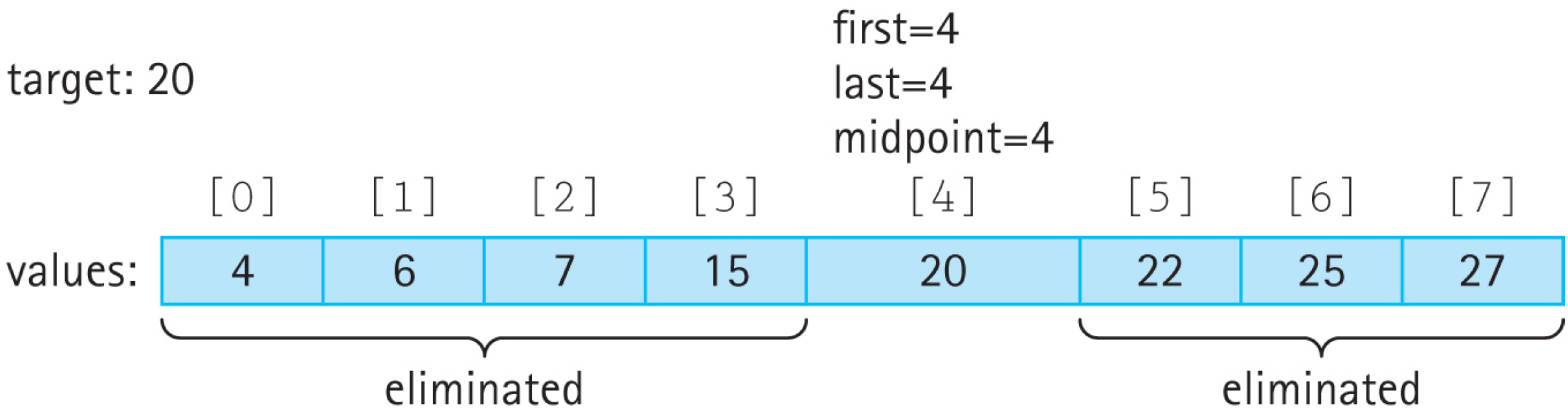
target: 20

	first=0			midpoint=3				last=7
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
values:	4	6	7	15	20	22	25	27

Binary Search Example



Binary Search Example



Binary Search Java

```
boolean binarySearch(int target, int first, int last)
// Precondition: first and last are legal indices of values
//
// If target is contained in values[first,last] return true
// otherwise return false.
{
   int midpoint = (first + last) / 2;
   if (first > last)
      return false;
   else
      if (target == values[midpoint])
         return true;
      else
      if (target > values[midpoint])
         return binarySearch(target, midpoint + 1, last);
      else
         return binarySearch(target, first, midpoint - 1);
}
```

Verification: Three Questions

- Base case(s)?

- Smaller calls?

- General case?

Verified

Binary Search Java

```
boolean binarySearch(int target, int first, int last)
// Precondition: first and last are legal indices of values
//
// If target is contained in values[first,last] return true
// otherwise return false.
{
   int midpoint = (first + last) / 2;
   if (first > last)
      return false;
   else
      if (target == values[midpoint])
         return true;
      else
      if (target > values[midpoint])
         return binarySearch(target, midpoint + 1, last);
      else
         return binarySearch(target, first, midpoint - 1);
}
```

Analysis

- Input size: n
- Input size change at each iteration:
- Order of growth for comparison operation?

Recap

- Recursion involves defining a solution based on smaller versions of the same solution
- Three components:
  - Base case
  - Check
  - Recursive case
- Three questions are needed to verify the correctness of your algorithm
- Binary search is a very efficient search algorithm with a simple recursive definition

Next Time...

- Dale, Joyce, Weems Chapter 3.4
  - Remember, you need to read it BEFORE you come to class!
- Check the course webpage for practice problems
- Peer Tutors
  - http://www.csc.villanova.edu/help/