Reminders

- Quiz today
- Homework 5 is due today
- Homework 6 is released
  - Due Thursday

Process

1. Express statement as “for all $n \geq b$, P(n)” for some b
2. Write “Basis Step” and show P(b) is true
3. Write “Inductive Step”
4. “Assume P(k) is true for arbitrary k” (hypothesis)
5. “Prove P(k) → P(k+1)” being explicit about P(k+1)
6. “This completes the inductive step”
7. Conclude the proof with “By mathematical induction, P(n) is true for all n ≥ b”

Examples

- Prove that if I have a sequence of dominos lined up correctly, I can topple all of them.
- If *n* is a positive integer, then
$$1 + 2 + \ldots + n = n(n+1)/2$$
- Prove $2^n < n!$ for $n \geq 4$

Strong Induction

- Similar to regular mathematical induction
- Sometimes, we can't complete inductive step
- Instead, inductive hypothesis shows:
  - If P(j) is true for all positive integers not exceeding *k*, then P(k+1) is true
  - ie: Assuming P(j) is true j = 1, 2, ..., k

Example

- All positive integers ≥ 2 can be written as product of primes

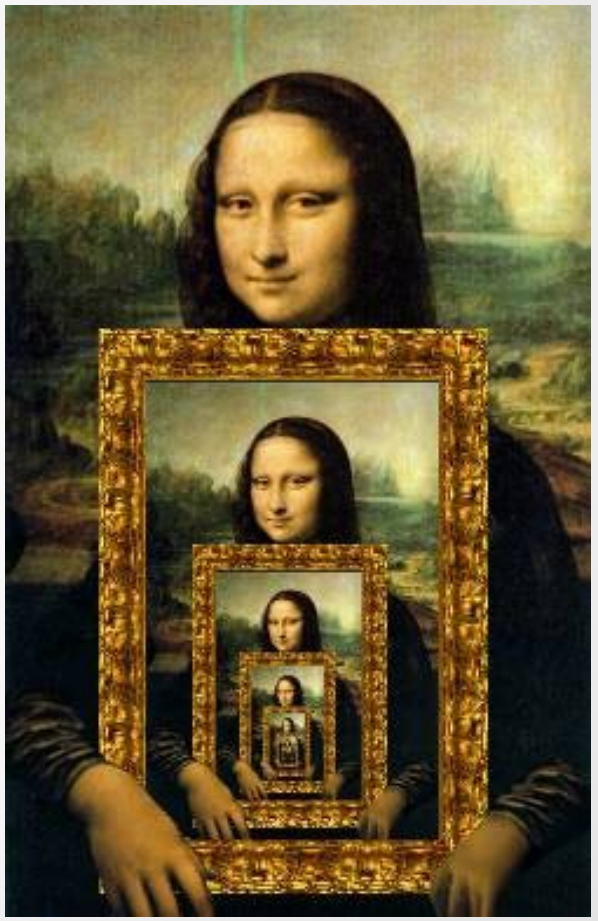
- Prove every amount of postage 12 cents or more can be formed using just 4-cent and 5-cent stamps

CMSC 203: Lecture 14

Recursive Algorithms

Recursion

- Define an object in terms of itself
- For example, defining a sequence by specifying how terms are found from previous terms
- Can use induction to prove results are correct



When Can I Use It?

- If the reduction of the problem on one input to a smaller input is still valid
- Solution to original problem can be found with sequence of reductions
- Some initial case must be established

Examples

- Computing $n!$

  procedure *factorial* (n: nonnegative integer)

  if *n* = 0 **then return** 1

  else return *n* * *factorial* (*n* – 1)

- Computing a^n

  procedure *power* (*a* : nonzero real, *n* : nonnegative int)

  if *n* = 0 **then return** 1

  else return *a* * *power*(*a*, *n* - 1)

Iteration

- **Iterative Procedure** - Use recursive definition successively to find values of function at larger integers
- May require much less computation than recursion

procedure *fibonacci* (*n* : nonnegative integer)

if *n* = 0 **then return** 0

else if *n* = 1 **then return** 1

else return *fibonacci* (n – 1) + *fibonacci* (n – 2)

Merge Sort

- Iteratively split list into two sublists of equal length
  - Or have one sublist with one more element
- Continue until each sublist has one element
- Successively merge pairs of lists
- O(n log n)

Quick Sort

- Take first element (a_1) and form two sublists
  - One contains elements less than a_1
  - Second contains elements greater than a_1
- Place a_1 at the end of the first sublist
- Repeat recursively until all sublists contain 1 item
- Combine sublists of 1 item in order they occur
- O(n log n) in practice