# Discrete Mathematics: Lecture 15

- Last time:
  - Chap 5.3: Recursive definitions and structural induction
- Today:
  - Chap 5.4: Recursive algorithms
  - Course review

# Review of last time

- Recursively defined functions
- A property of Fibonacci numbers and its application in analysis of time complexity of the Euclidean algorithm: whenever $n \geq 3$, $f_n > \alpha^{n-2}$, where $\alpha = (1 + \sqrt{5})/2$
- Recursively defined sets and structure
- Recursively defined functions on recursively defined sets
- Structural induction

# Chap 4.4: Recursive algorithms

- Definition: An algorithm is called recursive if it solves a problem by reducing it to an instance of the same problem with smaller input.
- Examples
  - Computing $n!$ (how the algorithm works)
  - Modular exponentiation
  - Computing $gcd(a, b)$
  - Binary search

# Proving recursive algorithms correct

- Mathematical induction can be used to prove that a recursive algorithm is correct, that is, it produces the desired output for all possible input values.
- Example: prove the recursive algorithm for modular exponentiation is correct.

# Recursion and iteration

- We can devise a recursive algorithm to evaluate a recursively defined function.
- We can also start with function values at base cases and successively apply the recursive definition to evaluate function values at larger integers. Such a procedure is called iterative.
- Often, an iterative approach requires much less computation than the recursive approach.
- Example: computing Fibonacci numbers
  linear versus exponential number of additions
- It is sometimes preferable to use a recursive procedure even if it is less efficient than an iterative procedure, since it is easier to write a recursive procedure

# An recursive sorting algorithm: The merge sort

procedure mergesort($L = a_1, a_2, \ldots, a_n$)
if $n > 1$ then
    $m := \lfloor n/2 \rfloor$
    $L_1 := a_1, a_2, \ldots, a_m$
    $L_2 := a_{m+1}, a_{m+2}, \ldots, a_n$
    $L := merge(mergesort(L_1), mergesort(L_2))$

procedure merge($L_1, L_2$ : sorted lists)
$L :=$ empty list
while both $L_1$ and $L_2$ are nonempty
    remove the smaller one of the first elements of $L_1$ and $L_2$
        from its list and put it at the end of $L$
    if one list is empty then append the other list to $L$

Example: merge 2,3,5,6 and 1,4

# Time complexity of the merge sort

- Lemma: Two sorted lists with $m$ elements and $n$ elements can be merged into a sorted list using $\leq m + n - 1$ comparisons.
- Theorem: The number of comparisons needed to merge sort a list with $n$ elements is $O(n \log n)$.

# Course review: Propositional logic

- Propositional connectives
  - negation, conjunction, disjunction, implication, bi-implication
  - converse (逆命题), contrapositive, and inverse (否命题) of implication
  - truth table
- tautology, contradiction, contingency; valid, satisfiable, consistent
- Propositional equivalences
  - proof using truth table
  - proof using basic equivalences
- Translating sentences to logical formulas

# Predicate logic

- Predicates: propositional functions
- Domains and quantifiers
- Logical equivalences
  - negating quantified expressions
- Translating sentences to quantified formulas

# Rules of inference

- Proofs
  - arguments: a sequence of statements
  - valid arguments
- Rules of inference
  - modus ponens, modus tollens, hypothetical syllogism, disjunctive syllogism, addition, simplification, conjunction
  - universal instantiation, universal generalization, existential instantiation (must use a constant symbol not used before), existential generalization
  - using rules to construct valid arguments

# Sets and functions

- Basic concepts of sets
  - empty set, subset, proper subset, power set, Cartesian product
- Set operations
  - union, disjoint, intersection, difference, complement
  - proving set identities: using basic set identities, using definition
- Basic concepts of functions
  - injection, surjection, bijection
  - inverse functions, composition of functions

# Algorithms and integers

- Complexity of algorithms
  - measured in terms of the number of main operations
  - Big-O/$\Omega$/$\Theta$ notation
  - $(log\, n)^c$, $n^{d_1}$, $n^{d_2}$, $b_1^n$, $b_2^n$, $n!$, where $c > 0$, $d_2 > d_1 > 0$, $b_2 > b_1 > 1$
- Algorithms for integers
  - Base $b$ representation of integers
  - Euclidean algorithm

# Basics of number theory

- divisibility and modular arithmetic
- express $gcd(a, b)$ as a linear combination of $a$ and $b$
- solve linear congruence $ax \equiv b \pmod{m}$
- using Chinese reminder theorem to solve the system $x \equiv a_i \pmod{m_i}$, $i = 1, \ldots, n$
  - $m_1, \ldots, m_n$ have to be pairwise relatively prime
  - otherwise, get another system so that $lcm(m_1, \ldots, m_n) = m'_1 \cdot \ldots \cdot m'_n$
- Format's little theorem

# Induction and recursion

- simple induction, strong induction, well-ordering property
- recursive definitions, structural induction, recursive algorithms