COMP 250 Midterm exam:
When: Wednesday Feb 22nd, 6:00pm - 7:30pm
Where:
• Last name starting with A to L: McIntyre room 522
• Last name starting with M to Z: McIntyre room 504
What to bring:
• 0) A couple of pencils (not pen) to fill the scantron sheets
• 1) Your student ID card
• 2) A 1-page double side crib sheet (8.5 inches x 11 inches)
• 3) Your brain and energy!

Java programming
• Everything we’ve covered in class could be on the exam
  – Methods
  – parameter passing
  – object oriented programming

Recursive algorithms
• To write a recursive algorithm:
  – Find how the problem can be broken up in one or more smaller problems of the same nature
  – Remember the base case!
• Usually, better running times are obtained when the size of the subproblems are approximately equal
  – \text{power}(a,n) = a \times \text{power}(a,n) \Rightarrow O(n)
  – \text{power}(a,n) = (\text{power}(a,n/2))^2 \Rightarrow O(\log n)
• Fibonacci, BinarySearch, Power, Integer multiplication, MergeSort

Induction proofs
• To prove that a proposition \( P(n) \) holds for all \( n \geq a \):
  – \text{Base case:}
    Prove that \( P(a) \) holds
  – \text{Induction step on} \( n \):
    Induction Hypothesis: Assume \( P(n) \) holds
    Prove that I.H. implies that \( P(n+1) \) holds

Proof of correctness of iterative algorithms
Using \textbf{Loop invariants:}
1. Initialization: It is true prior to the first iteration of the loop.
2. Maintenance: If it is true before an iteration of the loop, it remains true before the next iteration.
3. Termination: When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.
Running time

- Primitive operations
  - Running time is constant, indep. of problem size
    - Assigning a value to a variable
    - Calling a method, returning from a method
    - Arithmetic operations, comparisons
    - Indexing into an array
    - Following object reference
-条件
- Running time \( \equiv \) Number of primitive operations
- Loops: Sum the running time of each iteration
- \( \text{findMin}, \text{selectionSort}, \text{insertionSort} \)

Recurrences

- For recursive algorithms, we express the running time \( T(n) \) for an input of size \( n \) as a function of \( T(a) \) for some \( a < n \)
- Example:
  - Binary search: \( T(n) = T(n/2) + a \)
  - RecursiveFastMult.: \( T(n) = 3 \ T(n/2) + c \ n + d \)

Solving recurrences

- Solving recurrence \( \equiv \) give explicit formula for \( T(n) \)
- Substitution method:
  - Replace occurrences of \( T() \) by their value
  - Repeat until pattern emerges
- Prove by induction that guess is correct

Big-Oh notation

- \( f(n) \ is \ O(g(n)) \) iff there exist constants \( c \) and \( N \) such that \( f(n) \leq c \ g(n) \) for all \( n \geq N \)
- Proving that \( f(n) \ of \ O(g(n)) \), or \( f(n) \ is \ not \ O(g(n)) \)
- Tricks to establish if \( f(n) \ is \ O(g(n)) \):
  - Hierarchy of big-Oh classes, Simplification rules
  - Test of the limit of the ratio
- NEW! Theta notation
  - \( f(n) \ is \ \Theta(g(n)) \) iff \( f(n) \ is \ O(g(n)) \) and \( g(n) \ is \ \Theta(f(n)) \)
  - \( f(n) \ and \ g(n) \ grow \ equally \ fast \ (within \ a \ multiplicative \ constant) \)
- NEW! Omega notation
  - \( f(n) \ is \ \Omega(g(n)) \) iff \( g(n) \ is \ O(f(n)) \)

List Abstract Data Type

- Operations supported
- Two possible implementations
  - Array-based
  - Linked lists
- Advantages and disadvantages of each implementations

Linked lists

- Implementation (node: value, next)
- Basic operations
  - getFirst(), get(n), getLast()
  - removeFirst(), removeLast(), remove(o)
  - addFirst(o), addLast(o), add(o)
  - empty(), size()
- Advantages and disadvantages over arrays
Stacks

- Last-in First-out data structure
- Operations: push(o), pop(), top()
- Applications:
  - Back button on browser
  - checking parentheses
  - evaluating expressions (homework #3)

Queues

- Queue:
  - First-in First-out data structure
  - Operations: enqueue, dequeue, front
- Doubly-ended queue:
  - Elements can be accessed, added, or removed at both ends
- Rotating array implementation