Midterm Review

Topics
- Design
- Java Syntax
- Contracts
- Complexity
- Vectors
- Sorting
- Induction
- Recursion
- Linked lists
- Binary Representation

Design
- Describe in English
  - Nouns -> State, group into classes
  - Verbs -> Computation, create methods on classes
  - Adjectives -> Interfaces for similar classes
- Caller decides return value, function implementer decides arguments
- Avoid duplicate state, unless the performance advantage is significant
- Avoid duplicate code: use helper methods and abstract classes

Java & Syntax
- Objects (state = members, computation = methods)
- static, final
- Inheritance
  - public, private, protected
  - Interfaces
  - Abstract classes
  - Derived (sub) classes
  - super
  - Generics

Contracts
- Document all assumptions
- Pre & post-conditions
- Class invariants
- Assertions vs. [input] errors

Vector
- Amortization trick for O(1) append
- Bounds for common methods, e.g.,
  - get/set: O(1)
  - add: O(1) expected amortized
  - add to front: O(n)
  - removeAt: O(1)
**Sorting**
- Be able to recognize and describe algorithms
- Insertion sort
  - Using each element forward (or backward). Easy on lists and arrays
  - \( O(n^2) \), low constants
- Merge sort
  - Recursive split and then merge. Ping-pong arrays, easy on lists
  - \( O(n \log n) \)
- Quick sort
  - Recursive partition and then swap. Reasonable on arrays
  - \( O(n \log n) \) expected, \( O(n^2) \) worst
- Comparable elements to be sorted
- Comparator objects

* A lot of caveats on this bound

**Complexity**
- Definition of asymptotic upper bound: \( f(x) \) is \( O(g(x)) \)
- Identify “trivially” \( O(1), O(n \log(n)), O(n), O(n^2), O(n^3) \) algorithms
- Expected and worst case bounds
- Expected amortized bounds

**Induction**
- Structure an inductive proof
  - Base case (e.g., let \( n = 1 \))
  - Inductive step (e.g., assume true for \( n = k \), prove for \( n = k + 1 \))
  - Full proof
- Relationship to recursion

**Recursion**
- Linked-list applications
- Exhaustive enumeration (e.g., subset sum) application
- Iteration -> Recursion
- Recursion -> Iteration using an explicit stack
- How compilers/interpreters evaluate recursion using the built-in stack
- Be aware of the space cost of the stack

**Linked List**
- Trivial singly-linked list with only a head
- *“Common”* singly-linked list with head, tail, and count
- Doubly-linked list
- Lists with dummy nodes
- Bounds for common methods under each variant, e.g., for common:
  - \( \text{get}() : O(n) \)
  - \( \text{add} : O(1) \)
  - \( \text{add to front} : O(1) \)
  - \( \text{removeAt} : O(n) \), but \( O(1) \) during iteration

**Binary Numbers**
- Decimal <-> binary conversion
- \( n \) bits = \( 2^n \) unique representable values
- Bitwise operators: \& | ~ ^ >> <<
- Use of bit masks
- Common identities and tricks:
  - \( x << 2 = x \cdot 2 \)
  - \( x >> 1 = x / 2 \)
  - \( 1 << n = 2^n \)
  - \( 1 \land x = x \)
  - \( x \lor (x+1) = x + 1 \)
  - \( (x >> n) \land y \) = read \( n \) of \( x \)