1. Program Readability

**Preconditions**
- A precondition tells when (under what circumstances) a method can be called:

```java
public Ratio(int top, int bottom)
/* Initialize the receiver to be the fraction
whose numerator is the top and whose
denominator is the bottom.
The bottom cannot be zero.
pre: bottom != 0 */
```

**Postconditions**
- A postcondition tells what a method must do:

```java
public Ratio(int top, int bottom)
/* Initialize the receiver to be the fraction whose numerator is the top and whose
denominator is the bottom.
pre: bottom != 0
post: constructs a ratio equivalent to top/bottom */
```

2. Interface

- There are three aspects for any class:
  - Protocol: a description of the resources that the class provides.
  - Implementation: the code that describes the private state of instances and implements the computational resources of the class.
  - Use: code in a using class causes new instances of the used class to be created and invokes computations on these instances.

A Java interface provides a mechanism to formally specify the protocol

3. Java’s Exception

- When we write a piece of code that we know might produce an exception and we want to catch (handle) the exception, we encapsulate that code in a try block.
- If no exception is thrown by code within the try block (or the methods that are called within the try block), the code executes normally.
- If an exception arises in the try block the execution of the try block terminates execution immediately and a catch-clause is sought to handle the exception. Either
  1. An appropriate catch clause is found, in which case it is executed; or
  2. The exception is propagated to the calling method
Full Semantics of try-catch-finally

- If an expression in the try block throws an exception:
  - the rest of the code in the try block is skipped.
  - If the exception "matches" the first catch clause
    • the code in its catch block is run
    • the code in the finally clause is run.
    • the rest of the catch clauses are ignored.
    • the rest of the code in the method is run.
  - If the exception does not "match" the first catch clause, similar actions are taken for the first "matching" catch clause.

Possible Execution Paths (1)

1. **No exception occurs**
   1. Execute the try block
   2. Execute the finally clause
   3. Execute the rest of the method

Possible Execution Paths (2)

1. **Exception occurs and is caught**
   1. Execute the try block until the first exception occurs
   2. Execute the first catch clause that matches the exception
   3. Execute the finally clause
   4. Execute the rest of the method

Possible Execution Paths (3)

3. **Exception occurs and is not caught**
   1. Execute the try block until the first exception occurs
   2. Execute the finally clause
   3. Propagate the exception to the calling method

4. **Vector and its implementation**
   - Array
   - Vector
   - How to use an array to implement a container with unlimited size, like vectors
   - Time and space complexity of our implementations
5 Time and Space Complexity

- The time and space requirements of a method can be determined by analysis.
- Since these values depend on the data size, we often describe them as a function of the data size (referred to as n).
- These functions are called respectively, the time complexity, time(n), and space complexity, space(n), of the algorithm.

Big-O definition

- If an algorithm has time complexity order \( n^2 \) we write \( \text{time}(n) = O(n^2) \)
- The technical definition of big-O is:
  \[
  \text{time}(n) \text{ is } O(g(n)) \text{ if and only if } \lim_{n \to \infty} \frac{\text{time}(n)}{g(n)} = c, \text{ where } c \text{ is a constant}
  \]
- In English, this means \( \text{time}(n) \) grows no faster than the function \( g(n) \)
- Two Important rules:
  - make \( g(n) \) as small as possible
  - \( g(n) \) never contains unnecessary constants or terms

6. Recursive Methods

- Recursion occurs when a method calls itself, either directly or indirectly.
- For recursion to terminate, two conditions must be met:
  - there must be one or more simple cases that do not make recursive calls.
  - the recursive call must somehow be simpler than the original call so that they lead to the base case.

To write a recursive function

- One needs to transform the given problem into a same problem with a smaller size such that the solution can be obtained based on the solution to the smaller problem.
- One needs to identify a boundary problem with a simple solution (i.e., it can be solved without recursion.)

Common complexity orders

<table>
<thead>
<tr>
<th>NAME</th>
<th>Big-O</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>logarithmic</td>
<td>( O(\log(n)) )</td>
</tr>
<tr>
<td>linear</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>quadratic</td>
<td>( O(n^2) )</td>
</tr>
<tr>
<td>cubic</td>
<td>( O(n^3) )</td>
</tr>
<tr>
<td>exponential</td>
<td>( O(2^n) )</td>
</tr>
</tbody>
</table>

7. Format of the midterms

- Understanding java programs
  - Debug
  - Find out outputs
  - Specify the pre and post conditions
- Programming
  - To implement a given interface
  - To write a recursive functions for solving a given simple program
  - To deal with an exception
Format of the midterms (continued)

- Analyze a given algorithm/function
  - Time complexity
  - Space complexity