## Algorithms and Their Evaluation

### Search Algorithms:

**Binary Search**

* **Purpose**: Locate parts by partID or name
* **Implementation Details**:
  + Precondition: Collection must be sorted
  + Time Complexity: O(log n)
* **Use Case**: Efficient lookup of specific parts in a sorted inventory

### Sorting Algorithms

**Quicksort**

* **Purpose**: Sort inventory by various attributes (e.g., price, performance metrics)
* **Implementation Details**:
  + Time Complexity: Average case O(n log n), Worst case O(n²)
  + Pivot Selection: Median-of-three for improved performance
* **Use Cases**:
  + Price-based sorting for budget filtering
  + Performance-based sorting for recommendations
  + Name-based sorting for alphabetical listings

### Performance Metrics Algorithms

**computeAdditionalMetrics()**

* **Purpose**: Calculate component performance and price-performance ratio
* **Implementation Details**:
  + Component-specific metric calculations
  + Time Complexity: O(1) per component
* **Use Cases**:
  + Performance evaluation
  + Price-performance calculation
  + Compatibility calculations

**getCompatibilityRate()**

* **Purpose**: Assess component compatibility based on user preferences
* **Implementation Details**:
  + Time Complexity: O(1) per component
* **Use Cases**:
  + Component compatibility checks and PC build recommendations based on it

### 3.4 File I/O Operations

* **Purpose**: Load and save inventory data
* **Implementation Details**:
  + Time Complexity: O(n), where n is the number of items
* **Use Cases**:
  + Initial inventory loading
  + Real-time inventory updates
  + Saving state for persistence

## 4. Data Structure Selection

### 4.1 Main Collection

* **Structure**: vector<ComputerPart\*>
* **Rationale**:
  + Supports dynamic resizing
  + Allows random access for quick lookups
  + Supports polymorphism for various component types
  + Efficient for sorting operations
  + Fast access by index O(1)