http://www.fluffycat.com/Java-Design-Patterns/

http://stackoverflow.com/questions/350404/how-do-the-proxy-decorator-adaptor-and-bridge-patterns-differ

Proxy, Decorator, Adapter, and Bridge are all variations on "wrapping" a class. But their uses are different.

* **Proxy** could be used when you want to lazy-instantiate an object, or hide the fact that you're calling a remote service, or control access to the object.
* **Decorator** is also called "Smart Proxy." This is used when you want to add functionality to an object, but not by extending that object's type. This allows you to do so at runtime.
* **Adapter** is used when you have an abstract interface, and you want to map that interface to another object which has similar functional role, but a different interface.
* **Bridge** is very similar to Adapter, but we call it Bridge when you define both the abstract interface and the underlying implementation. I.e. you're not adapting to some legacy or third-party code, you're the designer of all the code but you need to be able to swap out different implementations.
* **Facade** is a higher-level (read: simpler) interface to a subsystem of one or more classes. Think of Facade as a sort of container for other objects, as opposed to simply a wrapper.

**Sorting algorithm:**

Comparison sort: O(nlogn) Merge, Quick, Heap.

Non-Comparison Sort: O(n) Counting, Bucket, Bubble

Search: BFS, DFS

**Trees**:

BST, Two children nodes. Can be skewed.

Red-Black, Two children nodes. self-balancing

AVL, Two children nodes. more balanced than red-black

B-tree, multiple children. .mongo DB. unlike self-balancing tree, optimized of read/write large data, used in databases and file systems.

B+ tree, CoruchDB, all data are stored in leaf. only keys are stored in interior nodes.

Trie (prefix tree): good for store dictionary., phone book. start typing and it immediately finds all strings that began with your string

<http://stackoverflow.com/questions/623892/where-do-i-find-a-standard-trie-based-map-implementation-in-java>

<http://code.google.com/p/google-collections/issues/detail?id=5>

Suffix tree: store a word. Good for search (longest) substring.

Generalized Suffix tree: store a set of words.

Heap: itself does not require all elements sorted. Can be used for sorting. Takes O(NlgN).

Hashmap: (non-synchronized, can contain null, recommended) . elements not sorted. Must have hash function. Find by O(lgN). worst case O(N)

Hashtable: (synchronized, cannot conain null, obsolete)

TreeMap: sorted elements. binary search O(lgN). java implementation is based upon red-black tree.

LinkedHashMap: maintain input order. Hashmap does not maintain input order. Hibernate uses this internally for <map .. order-by=”xxx”>

LinkedHashSet: maintain input order. Set does not maintain input order. Hibernate uses this internally for <set .. order-by=”xxx”>

Tree diameter: <http://www.cs.duke.edu/courses/spring00/cps100/assign/trees/diameter.html>

LCA problem: <http://www.leetcode.com/2011/07/lowest-common-ancestor-of-a-binary-tree-part-i.html>

BFS: using a queue

DFS: using a stack

*function closure(c, c++,* ***not*** *java)*

**Structural (modular):**

**OOP:**

**Declarative:**

What to do, no side effect

**Logic:**

Prolog, SQL

**Functional:**

*lambda calculus*

*function closure*

Scala, Scheme

**Imperative:**

How to do, side effect

**Procedural:**

C, C++, Pascal