The complete cheat sheets from both midterms are at the back.

Sorting

Pseudocode for MergeSort:

```
int[] mergeSort(int[] sortme) {
  if (sortme.length == 1) {
    return sortme;
  }

  // Copy (but do not sort or otherwise re-order) the two halves.
  int[] lefts = copyLeftHalf(sortme);
  int[] rights = copyRightHalf(sortme);

  lefts = mergeSort(lefts);
  rights = mergeSort(rights);

  // merge takes in two sorted arrays and merges them into a
  // single sorted array.
  return merge(lefts, rights);
}
```

Graphs

One possible implementation of a graph (An “adjacency list”):

```
// One node, and its neighbors.
class GraphNode {
  // Label of this node.
  public String myLabel;
  // Adjacent nodes.
  public ArrayList<GraphNode> myNeighbors;
  // myNeighbors.get(0) has label myEdgeLabels.get(0).
  public ArrayList<String> myEdgeLabels;

  /* Constructor, getters, setters, methods etc. elided. */
}

// The graph itself.
class Graph {
  public ArrayList<GraphNode> myNodes;
  /* Constructor, getters, setters, methods etc. elided. */
}
```

Another possible implementation of a graph (an “adjacency matrix”):

```
class Graph {
  // If myMatrix[i][j] == true, there’s an edge from node i to node j.
  // NOTE THAT this does not necessarily mean there’s an edge from j to i!
  public boolean[][] myMatrix;
  // Instance variables for storing labels elided, as are constructors,
  // methods, etc.
}
```

x-first search

The following code implements breadth-first search. If the Queue is replaced by a Stack, it implements depth-first search. If the Queue is replaced by a Priority Queue, it implements informed search (informed by whatever the ordering on the nodes is).

```
void BFS(GraphNode n) {
  Queue<GraphNode> q = new Queue<GraphNode>();
  Set<GraphNode> visited = new HashSet<GraphNode>();
  q.add(n);
  visited.add(n);

  while (q.size() > 0) {
    GraphNode n = q.poll();

    // "Iterate through the neighbors." May look different depending on
    // how your graph is stored.
    for (GraphNode n2 : n.getNeighbors()) {
      if (!visited.contains(n2)) {
        q.add(n2);
        visited.add(n2);
      }
    }
  }
}
```