CMSC132 Practice Questions Answers

Problem 1 Sorting

A. What is a comparison sort?
   Answer: a sort where the only operation allowed on keys is to determine the relative order of two of them.

B. When is a sorting algorithm not a comparison sort? Given an example of a sorting algorithm that isn’t a comparison sort.
   Answer: a sort that uses other properties, such as the digits in a number. Examples include counting sort, radix sort and bucket sort.

C. What is a stable sort?
   Answer: one that doesn’t reorder elements that have equal keys

D. What is an in-place sort?
   Answer: one that doesn’t need more than a constant amount of additional space to sort the elements

E. What is an external sort?
   Answer: a sort that works with data that isn’t all read into memory at once (e.g., resides on disk).

F. What is the average case complexity of sorting a random array using
   a. bubble sort: O(n^2)
   b. heap sort: O(n log n)
   c. quick sort: O(n log n)

G. What is the worst case complexity of sorting using
   a. selection/exchange sort: O(n^2)
   b. tree sort: questions was perhaps ambiguous: with a balanced tree implementation, it is O(n log n)
   with a non-balanced tree, it is O(n^2)
   c. heap sort: O(n log n)
   d. quick sort: : O(n^2)

H. Are the typical implementations of these sorts stable?
   a. bubble sort: yes
   b. quick sort: no
   c. merge sort: yes

I. Are the typical implementations of these sorts performed in place?
   a. selection/exchange sort: yes
   b. tree sort: no
   c. merge sort: no
Problem 2 Recursion

For this problem, you need to implement a recursive solution to the nqueens problem. The nqueens problem is the problem of, given an N by N chess board, find all placements of N queens such that no queen can attack any other queen.

You are provided with a method that determines if two queens can attack each other:

```java
static boolean canAttack(int r1, int c1, int r2, int c2) {
    return r1 == r2 || c1 == c2 || r1 + c1 == r2 + c2
        || r1 - c1 == r2 - c2;
}
```

For this problem, you need to define a method nqueen(int n) that will print out all placements of n queens on a nxn board (board positions numbered 0..n-1):

```java
static void nqueen(int n) {
    // Your implementation goes here
}
```

You should use recursion to implement nqueen (e.g., your nqueens method should invoke an auxiliary function that is recursive).

a) What is the base case for the recursive method(s) you give in part (b)?

Answer: no more queens to be placed.

b) Provide your implementation of nqueen(int n) and any auxiliary functions you need to define:

```java
static void nqueen(int n) {
    nqueens(0, new int[n]);
}

static void nqueens(int nextCol, int[] row) {
    if (nextCol == row.length) {
        System.out.println(Arrays.toString(row));
        return;
    }
    // Check for all positions to place a queen in column nextCol
    for (int r = 0; r < row.length; r++) {
        // can we place a queen on row r of column nextCol?
        for (int c = 0; c < nextCol; c++)
            if (canAttack(row[c], c, r, nextCol))
                continue check;
        row[nextCol] = r;
        nqueens(nextCol + 1, row);
    }
}
```