Lab 3A

1. Use the Master Formula to find an explicit formula for the running time T(n) of a certain algorithm, given that T(n) satisfies the following recurrence relation:

```
T(n) = T(n/2) + n; T(1) = 1
```

2. *Primes.* Create a Java method with the following signature and return type:

```
int isPrime(int n)
```

   (Remember that a number *p* is *prime* it is > 1 and its only divisors are 1 and *p* itself.) Calculated in terms of input *value* (rather than *size*), make sure that the running time of your method is o(*n*) (little-oh). Then give the running time of your algorithm in terms of *size* of input. (See the slides for a formula for converting your "value" running time to "size" running time.)

   T(n) = _________ (in terms of value)

   T(b) = ___________ (in terms of size)

3. Devise an o(*n*) algorithm (calculated in terms of the *value* of *n*) that solves the following problem: Given positive integers *m, n*, return true if gcd(m,n) has the form $2^k * p$ for some nonnegative integer *k* and some prime *p*. Demonstrate that your algorithm really runs in o(*n*) time.

4. *Sorting.* The following pseudo-code describes a recursive algorithm for sorting a list.

```
Algorithm recSort(S)
        Input list L with n elements
        Output list L sorted
  if L.size() > 1 then
        (L_1, L_2) ← partition(L, n/2)
        recSort(L_1)
        recSort(L_2)
        L ← merge(L_1, L_2)
  return L
```

   The partition step of the algorithm places every element in position ≤ n/2 into a list L_1 and the elements to the right of position n/2 into a second list L_2. As with BinarySearch, this can be accomplished more efficiently by using indices instead of creating new lists each time.

   Do the following:

   A. Prove the algorithm is correct.

B. Use the version of the Master Formula shown below to prove correctness. Remember that merging can be done in O(n) time.

C. Implement this algorithm in Java, and write it so that it can be run in the Sorting Environment. Then run it and compare running times with the other sorting algorithms we have studied. Is it faster than LibrarySort?

Theorem. Suppose $T(n)$ satifies

$$T(1) = d; \qquad T(n) \le aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + cn^k \qquad \text{for } n > 1,$$

where k is a non-negative integer and a, b, c, d are constants with $a > 0, b > 1, c > 0, d \ge 0$. Then

$$T(n) = \begin{cases} O(n^k) & \text{if } a < b^k \\ O(n^k \log n) & \text{if } a = b^k \\ O\left(n^{\log_b a}\right) & \text{if } a > b^k \end{cases}$$