Lane Lawley
Algorithms 515, Homework 1

Question 1

Equivalence classes are denoted by vertical alignment. Classes further to the left grow slower, so for any adjacent pair of functions (f, g), f(n) = O(g(n)).

1	$\log\log n$	$\sqrt{\log n}$	$\log n$ $\log n^2$ $\log_{10} n$	$(\log n)^2$	$\sqrt{2}^{\log n}$ $n^{\frac{2}{3}}$	n $2^{\log n}$	$n \log n$	$n^{\frac{3}{2}}$	$n + \frac{n^2}{10^{20}}$ $4^{\log n}$	$n^{\log\log n}$ $(\log n)^{\log n}$	2^n 2^{n+1}	$n2^n$	2^{2n}	$n!$	2^{n^2}

In case this is too small to read, the full file has been submitted as question1.png.

Question 2

$T(n < 4) \leq 2T(\frac{n}{4}) + 1$ as per problem definition

$$2[2T(\frac{n}{16}) + 1] = 4T(\frac{n}{16}) + 2$$

$$4[2T(\frac{n}{64}) + 1] + 2 = 8T(\frac{n}{64}) + 3$$

and so on...

for depth k $T(n > 4) \leq 2^k T(\frac{n}{4^k}) + k$

base case $T(n \leq 4) = 1$ as per problem definition

so, solving for k... $\frac{n}{4^k} = 1 \Rightarrow k = \log_4 n$

$$T(n > 4) \leq 2^{\log_4 n} T(1) + \log_4 n = \sqrt{n} + \log_4 n$$

square root grows faster than log base four

$$T(n > 4) \leq \sqrt{n}$$

so $\sqrt{n}$ is our upper bound!

Question 3

The algorithm, implemented in Java, is attached.

Pseudocode

data = all n items to sort

base = data.length

buckets = data.length

for k = 0 to 2:
    for i = 0 to data.length:
        j = the kth digit of data[i] using base
        put data[i] in buckets[j]

    newdata = empty list

    for i = 0 to buckets.length:
        while buckets[i] is not empty:
            pop the first item off of buckets[i] and add it to the end of newdata

    replace data with newdata

Running Time Estimate

My algorithm is an implementation of radix sort, and performs in O(n) time. This incredible runtime is available by virtue of the non-comparative nature of the radix sort, which depends on the data being represented in a positional numerical system. For each digit, of which there can only be two, the algorithm does a linear pass over the data, grouping them into buckets and using this as their new order. There can only be two digits because I represent the numbers in base N, and the upper bound on the size of any number in the input is N^2 - 1.

Why It Works

The algorithm works, as mentioned above, due to the positional numeral system used to represent the numbers. Since representing a number less than N^2 - 1 in base N only requires two digits, we can sort the numbers first by their least significant digit, then by their most significant digit; this sorting models the actual comparison of numbers, and guarantees that more significant digits are given higher priority, but that all sub-digits are also already sorted. The sort is also stable, because the algorithm uses queues for the buckets needed to store intermediate orderings. This means that all elements in the same bucket are removed in the same order they were added. All operations on these queues take O(1) time.