Christian Rodriguez
Algorithms HW5

1. Prove that the incorrect sorting algorithm below runs in O(n*lg(n)) time.

```
Input: data: an array of integers to sort
Input: n: the number of values in data
Output: a permutation of data such that
data[1] <= data[2] <= ... <= data[n]
1 Algorithm: BadSort
2 foreach i = n - 1 to 1 step -1 do     //O(n)
3   foreach j = 1 to n - i step i do    //O(?)
4     if data[j] > data[j + i] then     //O(1)
5       Swap data[j] and data[j + i]    //O(1)
6     end
7   end
8 end
9 return data                           //O(1)
```

Solution:

Need to find what the Big-Oh of the nested for loop is.

Let x=# of iterations

x: 1, 2, 3, ..., x
j: 1, 1+i, i+2i, ..., 1+(x-1)i

Since j must not go over n-i iterations,

$1+(x-1)i \leq n-i$

And through simplification we can find:

$1+xi-i \leq n-i$
$1+xi \leq n$
$xi \leq n-1$
$x \leq (n-1)/i$

So the time for the nested for loop to go through a single iteration is $x \leq (n-1)/i$.

This can be represented as a summation in the following manner:

$i=n-1$ to 1, $n/i = n/(n-1)+n/(n-2)+...+n/1$

This summation can also be represented in reverse in the following manner:

$i=1$ to $n-1$, $n/i = n/(n-1)+n/(n-2)+...+n/(n-1)$

And then adding/subtracting n/n-n/n you get the following:

$i=1$ to n, $n/i = [n/(n-1)+n/(n-2)+...+n/(n-1)+n/n]-n/n$

Since n/n will always be 1, and we are not concerned with the Big-Oh of constants, we remove O(n/n)=O(1)

We can factor out the n out of the summation, leaving us with:

i=1 to n, 1/i = 1/(n-1)+1/(n-2)+...+1/(n-1)+1/n

By taking the Big-Oh of this summation, we get the following:

O(lg(n))

And, since we still have the n factored out of the summation, we have:

O(n)*O(lg(n)) = O(n*lg(n))

Which is our final answer!