What is the effect in the time required to solve a problem when you double the size of the input from n to $2n$, assuming that the number of milliseconds the algorithm uses to solve the problem with input size n is each of these function? [Express your answer in the simplest form possible, either as a ratio or a difference. Your answer may be a function of n or a constant.]

a) $\log\log n$ **b)** $\log n$ **c)** $100n$

d) $n\log n$ **e)** n^2 **f)** n^3

g) 2^n

In each case we want to compare the function evaluated at $2n$ to the function evaluated at n. The most desirable form of the comparison (subtraction or division) will vary.

a) Notice that

$$\log\log 2n - \log\log n = \log\frac{\log 2 + \log n}{\log n} = \log\frac{1+\log n}{\log n}.$$

If n is large, the fraction in this expression is approximately equal to 1, and therefore the expression is approximately equal to 0. In other words, hardly any extra time is required. For example, in going from $n = 1024$ to $n = 2048$, the number of extra milliseconds is $\log 11/10 \approx 0.14$.

b) Here we have $\log 2n - \log n = \log\frac{2n}{n} = \log 2 = 1$. One extra millisecond is required, independent of n.

c) This time it makes more sense to use a ratio comparison, rather than a difference comparison. Because $100(2n)/(100n) = 2$, we conclude that twice as much time is needed for the larger problem.

d) The controlling factor here is n, rather than $\log n$, so again we look at the ratio:

$$\frac{2n\log(2n)}{n\log n} = 2\cdot\frac{1+\log n}{\log n}$$

For large n, the final fraction is approximately 1, so we can say that the time required for $2n$ is a bit more than twice what it is for n.

e) Because $(2n)^2/n^2 = 4$, we see that four times as much time is required for the larger problem.

f) Because $(3n)^2/n^2 = 9$, we see that nine times as much time is required for the larger problem.

g) The relevant ratio is $2^{2n}/2^n$, which equals 2^n. If n is large, then this is a huge number. For example, in going from $n = 10$ to $n = 20$, the number of milliseconds increases over 1000-fold.

An algorithm is called **optimal** for the solution of a problem with respect to a specified operation if there is no algorithm for solving this problem using fewer operations.

a) Show that Algorithm 1 in Section 3.1 is an optimal algorithm with respect to the number of comparisons of integers. [*Note:* Comparisons used for bookkeeping in the loop are not of concern here.]

b) Is the linear search algorithm optimal with respect to the number of comparisons of integers (not including comparisons used for bookkeeping in the loop)?

```
procedure max(a_1, a_2, ...., a_n: integers)
   max := a_1
   for i := 2 to n
      if max < a_i then max := a_i
   return max{max is the largest element}
```

a) In order to find the maximum element of a list of n elements, we need to make at least $n-1$ comparisons, one to rule out each of the other elements. Since Algorithm 1 in Section 3.1 used just this number (not counting bookkeeping), it is optimal.

b) Linear search is not optimal, since we found that binary search was more efficient. This assumes that we can be given the list already sorted into increasing order.