Q1 Give an algorithm (pseudo code, with explanation) to compute 2^{2^n} in linear time, assuming multiplication of arbitrary size integers takes unit time. What is the bit-complexity if multiplications do not take unit time, but are a function of the bit-length.

Solution:

Algorithm to compute 2^{2^n} in linear time.

```
mult(n):
    a = 1 shifted to the left n bits
    return 1 shifted to the left a bits
```

The bit complexity if multiplication does not take unit time would be 2^{2n}.

Q2 Consider the problem of computing $N! = 1 \cdot 2 \cdot 3 \cdots N$

(a) If N is an n-bit number, how many bits long is $N!$ in $O()$ notation (give the tightest bound)?

Solution:

Since N is an n-bit number. We assume that $N \times N$ will be $\Theta(N^2)$ time complexity. However, since it is a factorial, the value multiplied will begin to get smaller. Since there is a decrease in the value of N being multiplied, then the total running time for $N!$ will be $\Theta(nlogn)$

(b) Give an algorithm to compute $N!$ and analyze its running time.

Solution:

```
nfactorial(n):
    int result = 0;
    for (int i = 2; i < n; i++){
        result *= i;
    }
    return result;
```

The for loop will run n times, multiplication $n \times n$ would be n^2 runtime, but since our multiplication is by an increasing value of n, the first numbers leading up to n are

negligeable until reaching to n. Instead of the multiplication being n, it can be considered as $logn$, taking a total of $nlogn$ running time to complete.