Section 2.2. Complexity of Algorithms

Note. In analyzing a problem, we are interested in the amount of time and memory that it takes to run the algorithm on a computer. We will only consider the time and do so by counting the number of operations (comparisons, additions, multiplications, divisions, or other basic operations) used in executing the program.

Note. On pages 106–108, the text shows that the complexity of algorithms 1, 2, 3 of Section 2.1 are $O(n)$, $O(n)$, and $O(\log n)$, respectively (in a "worst case" [versus average] analysis).

Definition. We use big-oh notation in describing the Complexity of Algorithms. We use the following terminology:

Complexity	**Terminology**
$O(1)$	constant complexity
$O(\log n)$	logarithmic complexity
$O(n)$	linear complexity
$O(n \log n)$	$n \log n$ complexity
$O(n^b)$, $b \in \mathbb{Z}^+$	polynomial complexity
$O(b^n)$, $b > 1$	exponential complexity
$O(n!)$	factorial complexity

Definition. An algorithm with worst-case complexity that takes polynomial time is *tractable*. If the worst-case complexity takes longer than polynomial time is intractable.

Definition. Problems for which a solution can be checked in polynomial time is in the *class NP* The class of *NP-complete problems* is a class of famous problems such that if there is a polynomial time worst-case solution of one, then all can be solved in polynomial time. So far, no such solution is known.

Solution. We have:

```
procedure exp2k(x: real number, k: positive integer)
      i := 1, P := x
while (i ≤ k)
      P := P ∗ P
      i = i + 1
{ x^(2^n) is output as P }
```

This requires $2k$ operations ($3k$ if we consider the "`i ≤ k`" comparison). So the algorithm is $O(k)$. Multiplying x by itself is $O(2^k)$, so the above algorithm is better.

Example. Page 112 Numbers 8 and 12.

Revised: 4/1/2019