Algorithmic Analysis

6.1 Algorithm Analysis

The goal of algorithmic analysis is to determine how the running time behaves as \( n \) gets large. The value \( n \) is usually the size of the structure or the number of elements it has. For example, traversing an array takes time proportional to \( n \) time.

We want to measure either time or space requirements of an algorithm. Time is the number of atomic operations executed. We cannot count everything; we just want an estimate. So, depending on the situation, one might count: arithmetic operations (usually assume addition and multiplication atomic, but not for large integer calculations); comparisons; procedure calls; or assignment statements. Ideally, pick one which simple to count but mirrors the true running time.

6.2 Order Notation

We define big-O:

\[
f(n) \text{ is } O(g(n)) \text{ if the growth of } f(n) \text{ is at most the growth of } g(n).
\]

So \( 5n \) is \( O(n^2) \) but \( n^2 \) is not \( O(5n) \). Note that constants do not matter; saying \( f \) is \( O(\sqrt{n}) \) is the same thing as saying \( f \) is \( O(\sqrt{22n}) \).

The order (or growth rate) of a function is the simplest smallest function that it is \( O \) of. It ignores coefficients and everything except the dominant term.

Example. Some would say \( f(n) = 2n^2 + 3n + 1 \) is \( O(n^3) \) and \( O(n^2) \). But its order is \( n^2 \).

Terminology: The notation \( O(1) \) means constant-time. Linear means proportional to \( n \). Quadratic means \( O(n^2) \). Sublinear means that the ratio \( f(n)/n \) tends to 0 as \( n \to \infty \) (sometimes written \( o(n) \)).

Long Arithmetic. Long addition of two \( n \)-digit numbers is linear. Long multiplication of two \( n \)-digit numbers is quadratic.

(Check!)
6.3 Combining Functions

- **ADD.** If \( T_1(n) \) is \( O(f(n)) \) and \( T_2(n) \) is \( O(g(n)) \), then \( T_1(n) + T_2(n) \) is \( \max(O(f(n)), O(g(n))) \).
  That is, when you add, the larger order takes over.

- **MULTIPLY.** If \( T_1(n) \) is \( O(f(n)) \) and \( T_2(n) \) is \( O(g(n)) \), then \( T_1(n) \times T_2(n) \) is \( O(f(n) \times g(n)) \).

  \[ \text{Example.} \ (n^4 + n) \times (3n^3 - 5) + 6n^6 \text{ has order } n^7 \]

6.4 Logarithms

The **log base 2** of a number is how many times you need to multiply 2 together to get that number. That is, \( \log n = L \iff 2^L = n \). Unless otherwise specified, computer science log is always base 2. So it gives the **number of bits**. The function \( \log n \) grows forever, but it grows (much) slower than any power of \( n \).

  \[ \text{Example.} \ Binary \ search \ takes \ O(\log n) \ time. \]

6.5 Loops and Consecutiveness

- **Loop:** How many times \( \times \) average case of loop

- **Consecutive blocks:** this is the sum and hence the maximum

  \[ \text{Primality Testing. The algorithm is} \]
  \[
  \text{for(int } y=2; \ y<N; \ y++) \\
  \quad \text{if( } N/y==0 \ ) \\
  \quad \quad \text{return false;} \\
  \quad \text{return true;} \\
  \]
  \[ \text{This takes } O(\sqrt{N}) \text{ time if the number is not prime, since then the smallest factor is at most } \sqrt{N}. \text{ But if the number is prime, then it takes } O(N) \text{ time. And, if we write the input as a } B \text{-bit number, this is } O(2^{B/2}) \text{ time.} \]
  \[ \text{(Can one do better?)} \]

Note that array access is assumed to take constant time.
Example. A sequence of positive integers is a \textit{radio sequence} if two integers the same value are at least that many places apart. Meaning, two 1s cannot be consecutive; two 2s must have at least 2 integers between them; etc. Here is a test of this: this method is \textit{quadratic}.

\begin{verbatim}
for(int x=0; x<len; x++)
    for(int y=x+1; y<len; y++)
        if(array[x]==array[y] && y-x<=array[x])
            return false;
return true;
\end{verbatim}