Chapter A2: Order Analysis

A2.1 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 while a single array access is assumed to take constant 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.

A2.2 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 $o(n)$).

Example. Long Arithmetic Long addition of two $n$-digit numbers is linear. Long multiplication of two $n$-digit numbers is quadratic.

(Check!)
A2.3 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 $O(f(n) \times g(n))$.

**Example.** $(n^4 + n) \times (3n^3 - 5) + 6n^6$ has order $n^7$

For consecutive blocks, the overall running time is their sum and hence the maximum. For loops, the overall running time is how many times the body is executed times the average case of the body. One can get an upper bound for loops by taking an upper bound for these two quantities.

**Example. Primality Testing** The algorithm is

```java
for(int y=2; y<N; y++)
    if( N%y==0 )
        return false;
return true;
```

This takes $O(\sqrt{N})$ time if the number is not prime, since then the smallest factor is at most $\sqrt{N}$. But if the number is prime, then it takes $O(N)$ time. And, if we write the input as a $B$-bit number, this is $O(2^{B/2})$ time. (Can one do better?)

The **log base 2** of a number is how many times you need to multiply 2 together to get that number. That is, $\log n = L$ when $2^L = n$. 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 $n$.

**Example.** Binary search takes $O(\log n)$ time.