

Algorithm Analysis

3.1 Order Notation

We define big-O:

$f(n)$ is $O(g(n))$ if the growth of $f(n)$ is at most the growth of $g(n)$.

The **order** of a function is the simplest smallest function that it is O of. It ignores coefficients and everything except the dominant term. Terminology: **Linear** means proportional to n . **Sublinear** means that the ratio $f(n)/n$ tends to 0 as $n \rightarrow \infty$.

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

3.2 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

3.3 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 **number of bits**. $\log n$ grows forever, but it grows slower than any power of n .

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

3.4 Algorithm Analysis

The goal of order 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 $O(n)$ 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 is the *correct order* but *simple* to count.

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

(Check!)

3.5 Loops and Consecutiveness

- **Loop**: How many times \times average case of loop
- **Consecutive blocks**: this is the sum and hence the maximum

|| Primality Testing. The obvious algorithm is

```
bool isPrime(int N)
{
    int F=2;
    while(N%F!=0)
        F++;
    return F==N;
}
```

|| 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?)

Example. A sequence of positive integers is a *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 numbers in between them; etc. Here is a test of this: this method is *quadratic*, meaning $O(n^2)$.

```
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;
```