

Project 1

Khalid Hourani

February 14, 2017

1 Analysis Tools

- The Constant Function

A constant function is any function of the form

$$f(n) = c$$

where c is some constant. Typically, we will have $c = 1$, since any constant function $g(n) = c$ can be written as

$$g(n) = c \cdot f(n)$$

where f is another constant function. The constant function frequently describes the number of steps required for basic operations, like assigning a value to a variable, adding two integers, or accessing an array index.

- The Logarithm Function

A logarithm function is any function of the form

$$f(n) = \log_b(n)$$

Defined by

$$\log_b(x) = y \text{ if and only if } b^y = x$$

Typically, while mathematicians write \log to mean $\ln = \log_e$ and engineers write \log to mean \log_{10} , computer scientists write \log to mean \log_2 . However, any base- b logarithm can be converted to base- a by the formula

$$\log_b(n) = \frac{\log_a(n)}{\log_a(b)}$$

so the choice of base has little impact in practice. The following four properties of logarithms are notable:

1. $\log_b(mn) = \log_b(m) + \log_b(n)$
2. $\log_b(m/n) = \log_b(m) - \log_b(n)$
3. $\log_b(m^n) = n \log_b(m)$
4. $b^{\log_b(m)} = m$

The logarithm function frequently shows up in algorithm analysis in examples like the binary search, which, on a search of n elements, takes at most $\log(n)$ operations.

- The Linear Function

A linear function is any function of the form

$$f(n) = cn$$

This function frequently occurs when we have to perform an operation on each element in a set. For example, a linear search on n elements takes at most n comparison operations.

- The N-Log-N Function

The N-Log-N function is a function of the form

$$f(n) = n \log(n)$$

This function grows faster than the logarithm function and the linear function, but slower than the quadratic function. This function tends to show up when optimizing a function that runs in quadratic time.

- The Quadratic Function

A quadratic function is any polynomial

$$f(n) = an^2 + bn + c$$

However, we typically focus on the quadratic

$$f(n) = n^2$$

since $an^2 + bn + c \leq (a + b + c)n^2$. This function tends to show up when nesting loops. For example, multiplying two numbers using the elementary algorithm requires quadratic time, since you must iterate through each digit of the first number while also iterating through each digit of the second.

- The Cubic Function and Other Polynomials

The cubic function is any polynomial

$$f(n) = an^3 + bn^2 + cn + d$$

Though, as in the above example, we typically consider

$$f(n) = n^3$$

A polynomial function of degree m is a function of the form

$$f(n) = a_0n^m + a_1n^{m-1} + \dots + a_{m-1}n + a_m$$

Similarly, we typically consider the polynomial

$$f(n) = n^m$$

These functions show up in cases where there are multiple nested loops. In the case of four nested loops, for example, the function would have a running time of approximately n^4 .

- The Exponential Function

An exponential function is any function of the form

$$f(n) = b^n$$

The exponential function can be defined for non-negative integer values recursively as

$$b^n = \begin{cases} 1 & n = 0 \\ b \cdot b^{n-1} & n > 0 \end{cases}$$

This definition can be extended to all integers by setting

$$b^{-n} = \frac{1}{b^n}$$

and to all rational numbers by

$$b^{\frac{m}{n}} = \sqrt[n]{b^m}$$

Finally, we can extend the definition to all real numbers by defining

$$b^r = \lim_{n \rightarrow \infty} b^{r_n}$$

where r_n is any sequence of rationals such that $r_n \rightarrow r$. An equivalent definition is found by setting

$$\ln(x) = \int_1^x \frac{1}{x} dx$$

and defining e^x as the inverse function of $\ln x$. We then set

$$b^x = e^{x \ln b}$$

The following three properties are notable:

1. $b^m \cdot b^n = b^{m+n}$
2. $\frac{b^m}{b^n} = b^{m-n}$
3. $(b^m)^n = b^{mn}$

The exponential function frequently occurs when the something is repeatedly doubled.

2 Asymptotes

- Asymptotic Notation

Asymptotic Notation is the use of the following terminology to describe the behavior of an algorithm:

1. $O(n)$ - **Big O Notation**
2. $\Omega(n)$ - **Big Omega Notation**
3. $\Theta(n)$ - **Big Theta Notation**

We say a function $f(n)$ is $O(g(n))$, written

$$\begin{aligned} f(n) &\text{ is } O(g(n)) \text{ or} \\ f(n) &= O(g(n)) \text{ or} \\ f(n) &\in O(g(n)) \end{aligned}$$

if and only if there exists a real constant $c > 0$ and an integer constant $m \geq 1$ such that

$$f(n) \leq cg(n) \text{ for all } n \geq m$$

For example, the function $f(n) = n^2 + 4n + 9$ is $O(n^3)$, since

$$n^2 + 4n + 9 \leq 1n^3 \text{ for } n \geq 4$$

Similarly, we say that $f(n)$ is $\Omega(g(n))$, written

$$\begin{aligned} f(n) &\text{ is } \Omega(g(n)) \text{ or} \\ f(n) &= \Omega(g(n)) \text{ or} \\ f(n) &\in \Omega(g(n)) \end{aligned}$$

if and only if there exists a real constant $c > 0$ and an integer constant $m \geq 1$ such that

$$f(n) \geq cg(n) \text{ for all } n \geq m$$

For example, the function $f(n) = n^2 + 4n + 9$ is $\Omega(n)$, since

$$n^2 + 4n + 9 \geq 1n \text{ for } n \geq 1$$

Finally, we say that f is $\Theta(g(n))$, written

$$\begin{aligned} f(n) &\text{ is } \Theta(g(n)) \text{ or} \\ f(n) &= \Theta(g(n)) \text{ or} \\ f(n) &\in \Theta(g(n)) \end{aligned}$$

if and only if there exist real constants $c' > 0$ and $c'' > 0$ and an integer constant $m \geq 1$ such that

$$c'g(n) \leq f(n) \leq c''g(n) \text{ for all } n \geq m$$

Equivalently, $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

For example, the function $f(n) = n^2 + 4n + 9$ is $\Theta(n^2)$, since

$$n^2 \leq n^2 + 4n + 9 \leq 14n^2 \text{ for } n \geq 1$$

Formally, we distinguish between Big O, Big Omega, and Big Theta notation, but in practice, especially among computer scientists, Big O and Big Theta are used interchangeably, with the context usually distinguishing whether we have a “tight” bound or simply an upper bound.

- Asymptotic Analysis

Asymptotic Analysis is the use of the above three definitions to analyze the speed and memory-efficiency of an algorithm. If a function $f(n)$ is $O(g(n))$, then, for sufficiently large n (hence “asymptotic”), we know that f is no slower than g . For example, if a particular algorithm runs in $O(n^2)$ time, then we know that doubling the size of the input will at most **quadruple** the runtime. Consider the following algorithm for calculating a non-negative exponent in C++:

```
int pow(x, n)
{
    if (n == 0)
    {
        return 1;
    }
    else
    {
        return x * pow(x, n - 1);
    }
}
```

This function will perform n operations, and therefore is $O(n)$. However, this function can be improved by performing what is called “binary exponentiation”. To calculate x^9 , we write $9 = 1001_b$ and observe then that

$$x^9 = x^8 x^1$$

This creates the far more efficient algorithm:

```
int pow(x, n)
{
    int result = 1;
    while (exponent > 0)
    {
        if (exponent % 2 == 1)
        {
            result *= base;
        }
        exponent /= 2;
        base *= base;
    }
    return result;
}
```

We see that this version of the function will terminate when the exponent becomes 0, which will occur after integer division by 2 has occurred $\lceil \log(n) \rceil$ times. Thus, this version of the function is $O(\log(n))$.

As a rule of thumb, algorithms which are “faster” asymptotically are preferred. So an $O(n)$ algorithm is preferable to an $O(n^2)$ algorithm. This isn’t *always* true, however, as an algorithm

with an $O(10^{1000}n)$ runtime, while certainly faster in long-term than an $O(n^2)$ algorithm, is hardly preferable. In fact, the $O(10^{1000}n)$ runtime algorithm will not see such results until $n > 10^{1000}$. In other words, it is important to contextualize these bounds.

3 Queue Operations

The following queue operations:

enqueue(5), enqueue(3), dequeue(), enqueue(2), enqueue(8), dequeue(), dequeue(), enqueue(9), enqueue(1), dequeue(), enqueue(7), enqueue(6), dequeue(), dequeue(), enqueue(4), dequeue(), dequeue()

appear as follows:

