CS 240 — Data Structures and Data Management

Kevin James

Spring 2014

---

1 Algorithms

An **algorithm** is a step-by-step process for carrying out a set of operations given an arbitrary problem instance. An algorithm **solves** a problem if, for every instance of the problem, the algorithm finds a valid solution in finite time.

A **program** is an implementation of an algorithm using a specified programming language.

For each problem we can have several algorithms and for each algorithm we can have several programs (implementations).

In practice, given a problem:

1. Design an algorithm.
2. Assess the correctness of that algorithm.
3. If the algorithm is acceptable, implement it. Otherwise, return to step 1.

When determining the efficiency of algorithms, we tend to be primarily concerned with either the runtime or the memory requirements. In this course, we will focus mostly on the runtime.

To perform runtime analysis, we may simply implement the algorithm and use some method to determine the end-to-end time of the program. Unfortunately, this approach has many variables: test system, programming language, programmer skill, compiler choice, input selection, ... This, of course, makes manual implementation a bad approach.

An idealized implementation uses a **Random Access Machine (RAM)** model. RAM systems have constant time access to memory locations and constant time primitive operations, thus the running time is determinable (as the number of memory operations plus the number of primitive operations).

We can also generally use **order notation** to compare multiple algorithms. For the most part, we compare assuming n is very large, since for small values of n the runtime will be miniscule regardless of algorithm.

We denote the runtime of a function as $T(f(x))$, for example: $T(3 \times 4)$ may be equal to $0.8ns = 8\text{ops}$. The return value is the number of operations required in the worst-case scenario.

Example: given $T_A(n) = 1000000n + 2000000000$ and $T_B(n) = 0.01n^2$, which is 'better'? For $n < 100000000$, algorithm B is better. Since we only care about large inputs, though, we say A is better overall.

Example 1.1. *Prove that* $2010n^2 + 1388 = \mathbb{O}(n^3)$.

Proof. $\forall c > 0$, $2010n^2 + 1388 \leq cn^3$
$n > 1388 \implies 2010n^2 + 1388 \leq 2011n^2 \leq cn^3$
$2011n^2 \leq cn^3 \iff 2011 \leq cn$
$n > \frac{2011}{c} = n_0$ □

Definition 1.1. *$f(n) = \mathbb{O}(g(n))$ if there exists a positive real number c and an integer $n_0 > 0$ such that $\forall n \geq n_0$, $f(n) \leq cg(n)$.*

More concretely, we can say that $f(n) = \mathbb{O}(af(n))$ and $a'f(n) = \mathbb{O}(f(n))$. It's also worth noting that order notation is transitive (e.g. $f(n) = \mathbb{O}(g(n))$ and $g(n) = \mathbb{O}(h(n))$ implies $f(n) = \mathbb{O}(h(n))$).

We use five different symbols to denote order notation:

- o denotes a function *always less* than a given order
- $\mathbb{O}$ denotes a function *less than or equal* to a given order
- Θ denotes a function *exactly equal* to a given order
- Ω denotes a function *greater than or equal* to a given order
- ω denotes a function *always greater* than a given order

Example 1.2. *For the psuedo-function*

```
function(n):
    sum = 0
    for i=1 to n:
        for j=i to n:
            sum = sum + (i-j)^2
            sum = sum^2
    return sum
```

we find the order equation

$$
\begin{aligned}
&= \Theta(1) + \sum_{i=1}^{n}\sum_{j=i}^{n}\Theta(1) + \Theta(1) \\
&= \Theta(1)\sum_{i=1}^{n}\sum_{j=1}^{n}1 \\
&= \Theta(1)\sum_{i=1}^{n}(n-i+1) \\
&= \Theta(1)\Big(\sum_{i=1}^{n}n - \sum_{i=1}^{n}i + \sum_{i=1}^{n}1\Big) \\
&= \Theta(1)\big(n^2 + i^n + n\big) \\
&= \Theta(n^2) + \Theta(i^n) + \Theta(n)
\end{aligned}
$$

Example 1.3. *For the psuedo-function*

```
function(A,n):
    max = 0
    for i=1 to n:
        for j=i to n:
        sum = 0
        for k=1 to j:
         sum = A[k]
         if sum > max:
         max = sum
    return max
```

we find the order equation

$$
\begin{aligned}
&= \sum_{i=1}^{n}\sum_{j=i}^{n}\big(1 + \sum_{k=i}^{j}c\big) \\
&= \sum_{i=1}^{n}\sum_{j=i}^{n}c(j-i+1) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n-i+1}j \\
&= \sum_{i=1}^{n}\Theta(n-i+1) \\
&= \sum_{i=1}^{n}\Theta(i)
\end{aligned}
$$

Example 1.4. *For the psuedo-function*

```
function(n):
    sum = 0
    for i=1 to n:
     j = i
        while j >= 1:
         sum = sum + i/j
         j = j/2
    return sum
```

we find the order equation

$$\sum_{i=1}^{n}\sum_{j=1}^{\log_2 i} c = \sum_{i=1}^{n}(c\log_2 i)$$
$$= c\big(\log 1 + \log 2 + \log 3 + \cdots + \log n\big)$$

all n of our terms are below $\log n$ *half of our n terms are above* $\frac{n}{2}$

$$= \mathbb{O}(n\log n) \qquad = \Omega(\frac{n}{2}\log\frac{n}{2})$$
$$= \Omega(n\log n)$$
$$= \Theta(n\log n)$$