Running Time and Elementary Operations

Richard Hua

Semester 2, 2021

THE UNIVERSITY OF AUCKLAND NEW ZEALAND

Running time

What we have so far: There are three main characteristics of an algorithm designed to solve a given problem.
Domain of definition: The set of legal inputs.
Correctness: The algorithm gives correct output for each legal input. This depends on the problem we are trying to solve and can be tricky to prove.
Resource usage: We focus on time efficiency here using our measure of running time.

Resource usage

- This depends on the input, and on the implementation (hardware, programmer skill, compiler/interpreter, language, etc.)
- It usually grows as the input size grows.
- There is a trade-off between resources (e.g., time vs space, FASTFIB, SLOWFIB).
- Running time is usually more important than space use.

THE UNIVERSITY OF AUCKLAND NEW ZEALAND

Running time

Now we have the basic tools we need.
We can describe the running time of an algorithm using a function $f(n)$ where n is the size of the input, e.g. $f : \mathbb{N} \to \mathbb{N}$ or $f : \mathbb{N} \to \mathbb{R}^+$.
Note that by nature of what we are studying here, we can safely assume that $f(n) \geqslant 0$ for all n.

Measuring input sizes and counting elementary operations

Can be tricky.
Can affect our analysis.

Example

Assuming that $F(n)$ has the order of n decimal digits (which is true), and adding two n digits number requires an amount of work that is in order of n, then the total amount of work done by FASTFIB is of order $1 + 2 + \cdots + n$ which is in order of n^2.
Consider what happens if we use the number of bits we need to represent n ($m = \lfloor \lg n \rfloor + 1$) as the measurement for the input size. So clearly, $2^{m-1} \leqslant n < 2^m$. So the total amount of work done would be in order of 2^m.

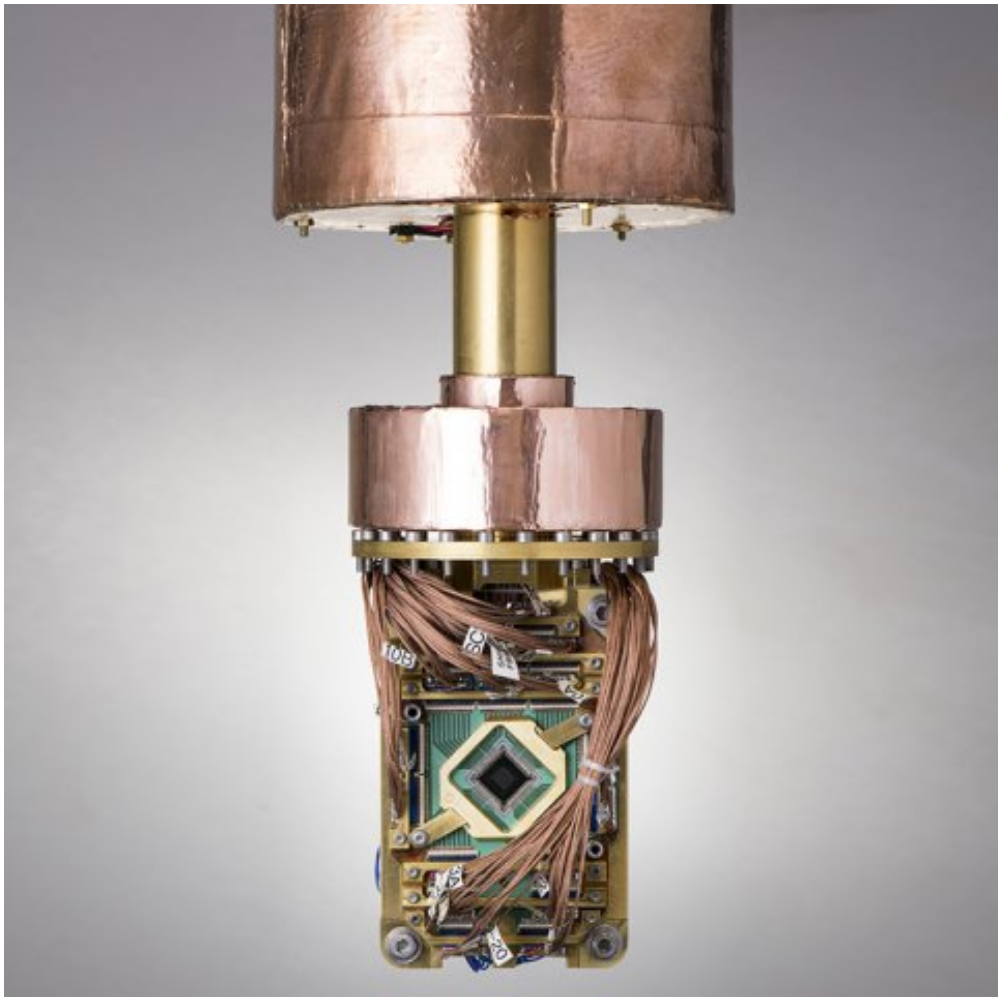
How we measure the input size and what we consider elementary operations must be specified clearly in algorithm analysis.

More limitations on our framework

There are computational models where this theoretical framework doesn't work at all.
Example: Adiabatic Quantum Computing (AQC).
`https://www.theguardian.com/technology/2016/may/22/age-of-quantum-computing-d-wave`,
`https://www.nature.com/articles/541447b`

More limitations on our framework

More limitations on our framework



Running time

Common functions as running times:

- Constant functions: $f(x) = c$ for some $c \in \mathbb{R}$, e.g. $f(n) = 5$, $f(n) = \sqrt{2}$.
- Linear functions: $f(x) = ax + b$ for some $a, b \in \mathbb{R}$, e.g. $f(n) = 3n + 5$.
- Polynomials: $f(x) = a_k x^k + \cdots + a_2 x^2 + a_1 x + a_0$ where $a_i \in \mathbb{R}$, e.g. $f(n) = \sqrt{3}n^4 + 5n + 3$.
- Exponential functions: $f(x) = a^x$ for some $a \in \mathbb{R}$, e.g. $f(n) = 3^n$.
- Logarithmic functions $f(x) = \log_a(x)$ for some $a \in \mathbb{R}$, e.g. $f(n) = 3 \lg n$.
- Linearithmic functions: $f(x) = cx \log(x)$ for some $c \in \mathbb{R}$, e.g. $f(n) = 3n \ln(n)$.
- Factorial functions: $f(x) = x! = 2 \cdot 3 \cdot 4 \cdot \cdots \cdot n$.

Running time

It seems natural to call running times by their respective function name, e.g. FASTFIB is a linear algorithm since it does a linear number elementary operations (in terms of n). Note that it's not always so simple since actual running times can be a combination of these types of functions, e.g. $f(n) = 1.5^n + 2n^2 + 3$.
We introduce the notion of **growth rate** here.

Definition

Let $f(n)$ and $g(n)$ be two functions. We say that $f(n)$ **grows faster** than $g(n)$, denoted by $f(n) >> g(n)$, if $\lim_{n\to\infty} \frac{|f(n)|}{|g(n)|} = \infty$.

Since we only work with non-negative functions, we can omit the absolute value sign. We will call a running time function by the name of the term with the fastest growth rate. e.g. an algorithm with running time $f(n) = 3n^3 + 10n^2 + \sqrt{3}n \lg 3$ is called a cubic algorithm.

Growth rate comparisons

Running time		Input size			
Function	Notation	10	100	1000	10^7
Constant	1	1	1	1	1
Logarithmic	$\log_{10} n$	1	2	3	7
Linear	n	10	100	1000	10^7
Linearithmic	$n \log_{10} n$	10	200	3000	7×10^7
Quadratic	n^2	100	10000	10^6	10^{14}
Exponential	2^n	1024	1.277×10^{30}	1.07×10^{301}	9.05×10^{3010299}

Note that there are about 3×10^{18} nanoseconds in a century. What does it tell us about the growth rate of these function? The exponential function grows way faster than all the other examples.

Estimating actual running times

By definition, the running time of an algorithm is the number of elementary operations it requires.
i.e. Not actual clock-time.
But what about actual programs?
From a practical point of view, we do sometimes would like to know how much clock-time a given program will need to execute.
Especially if the program is computing something important, we may wish to know how much time is needed before we get the answer.
E.g. What's the meaning of life, the universe and everything?
Which we now know the answer is 42.
The idea of elementary operations can obviously be applied to actual programs.

Estimating actual running times

Example

Say a particular program does exactly $f(n) = n^2$ number of one type of elementary operations. If we know how much clock-time is required to execute this elementary operation once, and assume there is no delay in between the operations, then we can estimate how much clock-time is needed for a given n.

Conclusion: It can be done under some assumptions.
Even if the program uses more than one type of elementary operations, we can use the slowest elementary operation to at least give us an 'upper bound' on the actual time needed.

Exercises

Exercise

A quadratic algorithm with running time $T(n) = cn^2$ *uses* 500 *elementary operations for processing* 10 *data items. How many will it use for processing* 1000 *data items?*

Substitute $n = 10$ in $T(n)$ and solve for c, we get $c = 500/10^2 = 5$. Now substitute $n = 1000$ in the formula and we get $T(1000) = 5 \times 1000^2 = 5 \times 10^6$.

Exercises

Exercise

A quadratic program runs for 1 *second on a problem of size* 200. *If you have* 12 *days, estimate the size of the largest input that can be solved within this time.*

We can use the same method we did from the last example. i.e. Start by assuming the program does $f(n) = cn^2$ and solve for c. Or think in a slightly different way, what happens to the running time if the input size is scaled by a factor k? Let's take a look at the ratio $f(kn)/f(n)$. If $f(n) = cn^2$ then $\frac{f(kn)}{f(n)} = \frac{c(kn)^2}{n^2} = k^2$. So we expect the running time to be scaled by a factor of k^2. Now 12 days is approximately 10^6 seconds and so the computing time has being scaled by a factor of 10^6. We would expect the input to be scaled by a factor of $\sqrt{10^6} = 1000$ and so we can process an input size of 200×1000.

Exercises

Exercise

A certain linearithmic algorithm (i.e. $T(n) = cn \log n$*) uses* 200 *elementary operations to process an input of size* 1000*. What is the number of elementary operations it will use if an input of size* 3000 *is given?*

Note that the base of the log is not specified. If it's base 10, then the calculation is easy and we can use the same method as before. Let $T(n) = cn \log_{10} n$, $T(100) = c(1000) \log_{10} 1000 = 200$ so $c = 200/3000 = 1/15$ and we proceed. At the end we get $T(3000) \approx 695$.

But can we make this assumption in this case?

THE UNIVERSITY OF AUCKLAND NEW ZEALAND

Exercises

Exercise

A certain linearithmic algorithm (i.e. $T(n) = cn \log n$*) uses* 200 *elementary operations to process an input of size* 1000*. What is the number of elementary operations it will use if an input of size* 3000 *is given?*

Recall the change of log base formula, $\log_a(x) = \frac{\log_b(x)}{\log_b(a)}$.
Note that the term on the bottom is a constant. And so if we did change the log base, it does look like we would need a different constant factor c.

Exercises

Exercise

A certain linearithmic algorithm (i.e. $T(n) = cn \log n$*) uses* 200 *elementary operations to process an input of size* 1000*. What is the number of elementary operations it will use if an input of size* 3000 *is given?*

Let's look at the factor of change instead. Let $k = 1000$ so $3000 = 3k$ (i.e. input size has been scaled by a factor of 3). What does it do to the number of operations?

$\frac{c(3k)\log 3k}{ck \log k} = \frac{3 \log 3k}{\log k} = \frac{3(\log 3 + \log k)}{\log k} = \frac{3 \log 3}{\log k} + \frac{3 \log k}{\log k} =$
$3 \log 3 / \log k + 3 = 3 \log 3 / \log 1000 + 3.$

Everything seems to be constant except $\log 3 / \log 1000$.

Exercises

Exercise

A certain linearithmic algorithm (i.e. $T(n) = cn \log n$) uses 200 *elementary operations to process an input of size* 1000*. What is the number of elementary operations it will use if an input of size* 3000 *is given?*

Let's show that the value $\log 3 / \log 1000$ is independent of the log base. Suppose $\frac{\log_x 3}{\log_x 1000} = r$ for some base x. What is $\frac{\log_y 3}{\log_y 1000} = ?$.

If we apply the change of log base formula to $\frac{\log_y 3}{\log_y 1000}$, we get

$$\frac{\log_y 3}{\log_y 1000} = \frac{\frac{\log_x 3}{\log_x y}}{\frac{\log x1000}{\log_x y}} = \frac{\log_x 3}{\log_x 1000} = r.$$

Exercises

Exercise

A certain linearithmic algorithm (i.e. $T(n) = cn \log n$*) uses* 200 *elementary operations to process an input of size* 1000*. What is the number of elementary operations it will use if an input of size* 3000 *is given?*

How is this useful?

It shows the rate of change is independent of the log base, and so it means our original calculation where we assumed it was base 10 would give us the correct answer.

THE UNIVERSITY OF AUCKLAND NEW ZEALAND

Pseudo-code running time

Now we have seen how we can estimate how the running time of a given algorithm behaves when n changes (in some cases).
What if we don't know what $T(n)$ is?

Running time of pseudo-code

function FINDMAX(array $a[0, \cdots, n-1]$)
    $k \leftarrow 0$
    for $j \leftarrow 1$ to $n-1$ **do**
        if $a[k] < a[j]$ **then**
            $k = j$
    return k

Always important to ask, what are we counting exactly?
E.g., do we count incrementing j or are we only counting what's inside the main loop body?
Need to specify.

It can get complicated...

Harder example:

for $i = 1;\ i < n;\ i \leftarrow 2i$ **do**
    for $j = 1;\ j < n;\ j \leftarrow 2j$ **do**
        if $j = 2i$ **then**
            for $k = 0;\ k < n;\ k \leftarrow k + 1$ **do**
                constant number of operations
        else
            for $k = 1;\ k < n;\ k \leftarrow 3k$ **do**
                constant number of operations

Running time of pseudo-code

General strategy:

- Running time of disjoint blocks adds.
- Running time of nested loops with non-interacting variables multiplies.
- If the structure is more complicated, find out how many iterations each loop has first, and then analyze how the loops/condition checks interact with each other to decide.

THE UNIVERSITY OF AUCKLAND NEW ZEALAND