COMP20010: Maths and Complexity Review Points

April 23, 2010

1 Big-O

1.1 Points

We want to know how the run-time[1] of an algorithm. The size of the input is N. Let $g(N)$ be some function of N. We say that the runtime is $O(g(N))$,

1. The meaning: If we look for big enough n, the runtime is less than a function proportional to $g(N)$.

2. Rule of thumb, find the biggest term, replace the constant multiplier (coefficient) with 1.

   Example:

$$\begin{aligned} 3N^2 + \frac{1}{5}N^5 + 57 + 19N + 0.001N^3 \\ \cancel{3N^2} + \boxed{\tfrac{1}{5}N^5} + \cancel{57} + \cancel{19N} + \cancel{0.001N^3} \\ \tfrac{1}{5}N^5 \\ O(N^5). \end{aligned}$$

3. Big-O is an <u>upper bound</u>. We want to find the best approximation to the run-time. If we cannot find a good approximation, we find a function which is definitely bigger for large enough N.

1.2 Loops within algorithms

1. Loops from 1 to N of statements which do not grow with N are $O(N)$.

[1] or other resources, such as memory space

Example:

```
for i = 1 to N do
  array[i] ← i + 1
end for
```

has runtime $O(N)$

Example:

```
for i = 1 to N^2 do
  print''Hello World''
end for
```

has runtime $O(N^2)$, because the loop runs for N^2 iterations.

2. Sequences of loops add.

Example:

```
for i = 1 to N do
  i ← i + 1
end for
printi
for i = 1 to N^2 do
  print''Hello World''
end for
```

has runtime $O(N) + O(N^2)$, which is $O(N^2)$.

3. Loops within loops multiply

Example:

```
for i = 1 to 2N do
  for j = 1 to N do
    print i + j
  end for
end for
```

is order $O(N^2)$.

2 Exponents and Logarithms

2.1 Exponential Growth

The exponential is repeated multiplication,

$$\begin{aligned}
\underbrace{2 \times 2 \times \cdots \times 2}_{N} &= 2^N \\
\underbrace{10 \times 10 \times \cdots \times 10}_{N} &= 10^N \\
\underbrace{e \times e \times \cdots \times e}_{N} &= e^N
\end{aligned}$$

Exponential growth B^N grows faster than any power of N if $B > 1$. B is the "base". You are familiar with this for $B = 10$:

$N=$	0	1	2	3	4	5	6	7	8
10^N	1	10	100	1000	10,000	100,000	1 million	10 million	100 million

And for $B = 2$:

$N=$	0	1	2	3	4	5	6	7	8
2^N	1	2	4	8	16	32	64	128	256

2.2 Logarithms

Logarithms are the inverse of exponentials. So, exponential is an answer to the question,

$$10^3 = ?.$$

What is the answer to the question,

$$10^? = 1000.$$

The answer is $\log_{10}(1000)$. In general, $\log_B x$ is the number which number which B must be raised to to get x. Since the exponential grows very fast (faster than any power), the logarithm grows very slowly (slower than any power).

Inverting the tables above shows how slowly logs grow. Base 10:

$N=$	1	10	100	1000	10,000	100,000	1 million	10 million	100 million
$\log_{10} N$	0	1	2	3	4	5	6	7	8

And base 2:

$N=$	1	2	4	8	16	32	64	128	256
$\log_2 N$	0	1	2	3	4	5	6	7	8

Note, often the base is assumed, or stated in words. Computer scientists often use base 2. More common are base 10, often written log, base $e = 2.71828183\ldots$, usually written ln.

2.3 Rules of logarithms

These are true for any base.

1. $\log(ab) = \log a + \log b$
2. $\log a^N = N \log a$

2.4 Logs in computer science

This illustrates the basic use of logs.

Question: Suppose you have a collection of N objects. You divide that collection in half and choose one of the halves. You divide that sub-collection in half and choose one of the halves. You divide that sub-sub-collection in half and choose of the halves. Etc. How many times can you do this before you are left with only one object?

Answer: $\log_2 N$. (Convince yourself of this using powers of N as a power of 2, e.g. 128.)

Binary search on a sorted list and divide-and-conquor algorithms use this idea.

2.5 Log-log plots

Suppose we believe that the runtime proportional to a power of N,

$$t = CN^p$$

but we don't know the power p or the constant C. How to find? One way is to plot $\log t$ versus $\log N$ in any base. From the rules above,

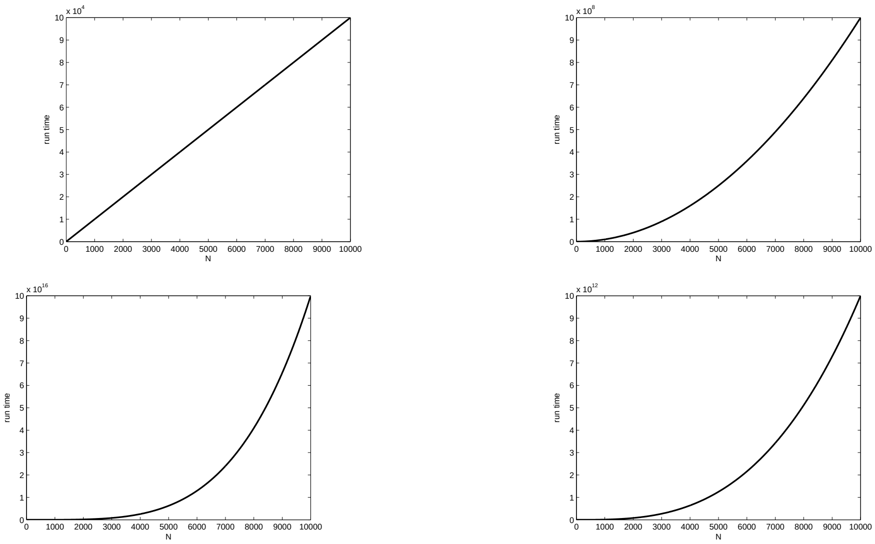
$$\log t = \log C + p \log N,$$

So, this will produce a straight line in the log-log plot, where the slope is the power, and the intercept is $\log C$. To summarize, if runtime is a power of N, then when plotted using a log-log plot,

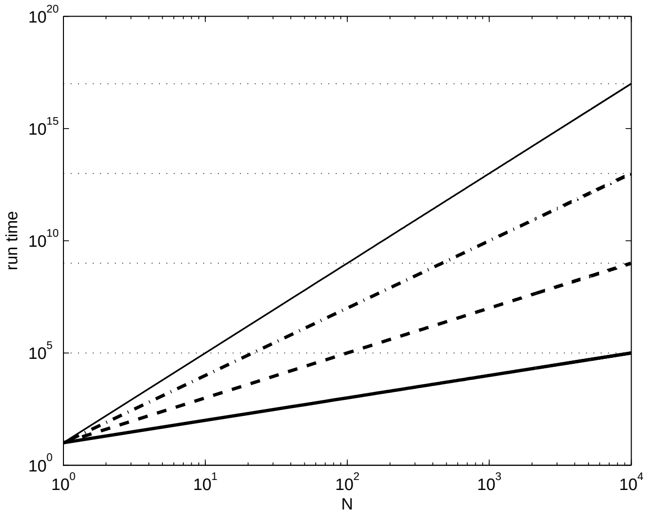
1. The plot will be a straight line,
2. The slope of the line is the power p,
3. The intercept of the line is the log of the constant of proportionality (i.e. the constant multiplying factor).

If it is approximately a powerlaw for large N, the above will hold for large N.

Here is an example. The following shows run time for different powers of N; clockwise from upper left: $10N$, $10N^2$, $10N^3$ and $10N^4$.



Now we plot as log-log plot: $10N$ (thick solid line), $10N^2$ (dashed line), $10N^3$ (dot-dashed line) and $10N^4$ (thin solid line). The horizontal dotted lines are at 10^5, 10^9, 10^{13} and 10^{17}.

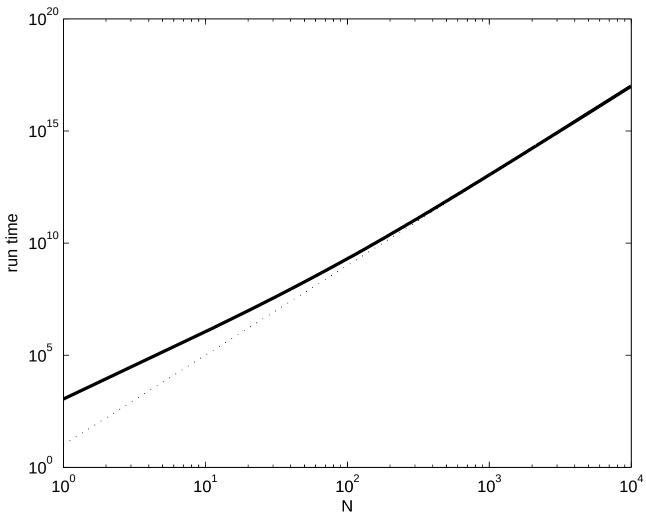


Notice that the axis are still labelled with N and run time, but on a log scale. We can take the logs (using base 10) in our heads. Let us compute the formula for the thin solid line. The intercept is at $10^1 = 10$. The run-time ranges from 10^1 to 10^{17} which on the log scale is from 1 to 17. The values of N range from 10^0 to 10^4 which on the log scale is 0 to 4. The slope is rise over run, e.g.

$$\frac{17-1}{4-0} = 4.$$

You should be able to convince yourself that the other lines also follow the appropriate power laws.

Often the data does not follow a powerlaw exactly, but does for large N. Here is an example,



The solid line is the data, and the dotted line is an approximation to the asymptote the data is approaching for large N. We see that for as N gets large, it approaches a power law. (Can you estimate the power?)

2.6 Semilog Plots

Suppose we believe that the runtime is exponential,

$$t = CB^N,$$

but we don't know the base B and we don't know the constant C. How to find these? One way is to plot $\log t$ versus N, called a semi-log plot. From the rules above,

$$\log t = \log C + N \log B.$$

This will produce a straight line in the semi-log plot, with the slope being $\log B$ and the intercept being $\log C$.