Analyzing Algorithms

We begin with some definitions.

Problem  A specification of an infinite set of inputs and corresponding outputs.

Note that in practice it can be difficult to distinguish infinite from large finite. For example, there are interesting questions we can ask about games like chess and Rubik’s cube.

Algorithm  A well defined computational procedure to go from any input to the corresponding output.

For our purposes — algorithms are specified using pseudo-code.

Analyze an Algorithm  Measure time and space used by the algorithms as a function of input size.

Measured not by running the program but by using an abstract model of computation.

Models of Computation

• specify the elementary computations out of which algorithms are built

• specify measure of time, space, input size

Bottom line: the model should reflect (but greatly simplify) reality.
1 Models of Computing

Our first model is general purpose, and can do anything a real computer can.

I. Pseudo-code

(a) Each line takes 1 time step.

Cautions:

• Some pseudo-code lines are too powerful. E.g., initializing an array is 1 line but has cost $n$ where $n$ is the length of the array.
• Integers can grow too large.

Example:

```plaintext
function Fibonacci(n)
    i, j := 0, 1
    for k = 1 to n do
        i, j := j, i + j
    return j
```

0, 1, 1, 2, 3, 5, 8, 13, 22, 35, 57, ... , 12200160415121876738, 19740274219868223167, ...

Fibonacci(93) < $2^{64}$, Fibonacci(94) > $2^{64}$

But the numbers grow so quickly that $n$ steps does not reflect reality: $n = 94$ causes an overflow using a 64-bit machine words, e.g., of type `unsigned long`. 
Example of “expression swell”

```maple
| \^/ | Maple 2020 (APPLE UNIVERSAL OSX)
| \_/ | Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2020
| \\ MAPLE / | All rights reserved. Maple is a trademark of
| \-|\_/ \_/> Waterlo Maple Inc.
| | Type ? for help.
| \> with(combinat):
| > fibonacci(7000):
| > (3664348305037232832276358967281604921857154393417598962627069872072801145996| |
| | 14526152053044740885086342851333977208014386960985843763728990950560338| |
| | 251079604581881276176484396309788275689930688063233914962445779252106554| |
| | 96624507469829546295160700987649783441511835995330307627790877434593918| |
| | 172439090198052759766331155561303319415384846587511336793498980790278348| |
| | 569811790271945996685562735304733743410753082978863630291190842638975525| |
| | 2823713762514625139073770774797947760248229483843646633681833549215123475| |
| | 05854827154728090873839417589043465226408479182330772693288661083451144| |
| | 27890779695990005117224442643471755382948548185636387620265469851156271937| |
| | 7096542631053943527028083010342485050544328989553516895529167136603624| |
| | 447915876432378032795204011886532527884702888478000351372675123176634292| |
| | 601143933394828014521369041992958201987034328371275386503307791744101108| |
| | 9802284149249691037048733860547306635658221088845805285220556917015235685| |
| | 0628426912688415908336529818984996274562358921065055713479149834483505| |
| | 47947756232111875546791699814341126089145748332466833264377501892470569| |
| | 0019182274575145613483922681596385654759233269442642416195266808813417| |
| | 3728034947587727832333884505984394140372235755587500123029133557948506487| |
| | 84308559343577303219076693667107597304311558020944644082678038946442563894| |
| | 2201877393180553647515377317433628216988894513327185167202072760588365| |
| | 918458452981222907609113064343011443630526238542031405945821004219543194| |
| | 7096493318081320875| |
| > quit
| memory used=1.0MB, alloc=8.3MB, time=0.05
```
(b) Count bit cost.

size of a positive integer \( a = \# \text{ bits in binary representation} \)

\[ = \lfloor \log_2 a \rfloor + 1 \]
\[ \in \Theta(\log a) \]

Example: \( c = 100! \)

\[ = 9332621544394415268169923885626670049071596826438162\]
\[ 146859296389521759999322991560894146397615651828\]
\[ 625369792082722375825118521091686400000000000000\]
\[ 0000000000 \]

- \( c \) is an example of a multi-precision integer.
- \( c \) can be stored an array of type \texttt{unsigned long} of length \([ (\log_2 c) / 64 ] + 1 = 9\).
- Computer algebra systems like Maple work with such multi-precision integers.

Let \( a \) and \( b \) be two multi-precision integers. Computing \( a \times b \) takes
- \( O((\log a)(\log b)) \) using the school method.
- \( O((\log a)(\log b)^{0.59}) \) steps using a more efficient multiplication.
  [we’ll see this algorithm later]

Our next models are more formal than pseudo-code.
II. RAM = Random Access Machine

- abstracts assembly language
- "random access" means we can access memory location $i$ in 1 time step (not like a tape or Turing machine)

How to charge for "size" of a memory location?
- unit cost (like (a) above) is too powerful
- bit cost (like (b) above) is too weak

Good compromise: word RAM - each memory location holds one word
- assume # bits in word is $\Theta(\log n)$, $n =$ input size.

Comments
- May seem a bit weird if hardware changes to fit the input!
- But consider, for example, if the input is an array $A[1 \ldots n]$. Then we expect/want an index $i \in [1 \ldots n]$ to fit in a word.
Other general purpose models:

III. Circuit family – abstracts hardware circuitry

IV. Turing machine – abstract human computer working with pencil and paper
   Note: time to access memory location \( i \) is proportional to \( i \).

Special purpose or “structured” models of computing:

- comparison-based model for sorting: \( \Omega(n \log n) \) lower bound
- arithmetic model

Our model of computation: the word RAM

\[ \equiv \]

pseudo-code and try to be realistic about what are elementary operations
2 Running Time of an Algorithm

Running time depends on input.

$$T_A(I) := \text{running-time of algorithm } A \text{ on input } I$$

We expect running-time to increase as size of input increases.

Simplify by expressing run-time as a function of input size.

[Note: model of computations must say how to count input size.]

For a given size $n$, there are various inputs.

How do we combine running-times to one number?

Worst case running-time

[the standard unless otherwise specified]

$$T_A(n) = \max\{T_A(I) \mid I \text{ an input of size } n\}$$

[leave off $A$ if it’s understood]
Why worst-case?
- We want an absolute guarantee.
- Alternative of average case is hard to analyze and depends on assumptions about input distribution (uniform?)

---

[This might come up later in the course (but ignore for now).]

When an algorithm uses random numbers we use expected run-time.

\[ T_A^E(I) := \text{expected run-time on input } I \]

(depends on random numbers used by \( A \))

Still use worst case over inputs

\[ T_A^E(n) := \max\{T_A^E(I) \mid I \text{ input of size } n\} \]
3 Asymptotic Analysis of Algorithms

Recall: $T(n) :=$ worst-case run-time of algorithm as a function of input size

We want $T(n)$ to be

- simple to express, e.g., $n^2$
- machine independent, thus ignore multiplicative factors (one machine might be twice as fast), thus ignore lower order terms

\[ \text{Definition Big Oh notation} \]

Let $f(n), g(n)$ be functions from $\mathbb{N}$ to $\mathbb{R}_{\geq 0}$.

Then $f(n)$ is $O(g(n))$ if $\exists$ positive constants $c$ and $n_0$ such that

\[ f(n) \leq cg(n) \text{ for all } n > n_0. \]

(we say $f$ is bounded by a constant times $g$ for $n$ sufficiently large — this is what asymptotic means)

\[ \text{Notation: } f(n) \in O(g(n)) \]
Big Oh gives an upper bound.

Examples:

- \( T(n) = 5n^2 + 3n + 25 \) is \( O(n^2) \)
- \( 10^{100}n \) is \( O(n) \)
- \( \log n \) is \( O(n) \) but \( n \) is not \( O(\log n) \)
- \( 2^{n+1} \) is \( O(2^n) \)
- \( (n + 1)! \) is not \( O(n!) \)
4 Analyzing Algorithms, continued

Properties of big Oh:

- max rule: $O(f(n) + g(n))$ is $O(\max\{f(n), g(n)\})$

- transitivity: $f(n) \in O(g(n))$, $g(n) \in O(h(n))$
  \[ \Rightarrow f(n) \in O(h(n)) \]

Further Definition:

- $f(n)$ is $\Omega(g(n))$ “big Omega” if $\exists$ positive constants $c, n_0$ s.t.
  \[ f(n) \geq cg(n) \forall n \geq n_0. \]

- $f(n)$ is $\Theta(g(n))$ “big Theta” if $f(n)$ is $O(g(n))$ and $\Omega(g(n))$
  – this gives an exact (asymptotic) bound

- $f(n)$ is $o(g(n))$ “little oh” if for any constant $c > 0$, there exists a positive constant $n_0$ such that
  \[ f(n) \leq cg(n) \forall n \geq n_0. \]

Equivalently

\[ \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \quad (\text{for } f, g : \mathbb{N} \to \mathbb{R}_{>0}) \]
Comparing Algorithms using Asymptotic Notation

Suppose the worst case run-time of

- algorithm $A$ is $O(n^2)$
- algorithm $B$ is $O(n \log n)$

Question: Which is better? Answer: We can’t know!

This is like $x \leq 5, y \leq 10$, which is smaller?

To compare algorithms, we need tight bounds, e.g.,

$$\Theta(n^2) \text{ vs } \Theta(n \log n)$$
• $O$ is like $\leq$
• $o$ is like $<$
• $\Theta$ is like $=$

One difference: we can compare 2 number

$x \leq y$ or $y \leq x$

But there are functions $f, g$ s.t. $f(n)$ is not $O(g(n))$ and $g(n)$ is not $O(f(n))$. Find some.

Challenge Can you find such functions using just $+,*,$ exponentiation, log?

• not allowed to use sin, $[\cdot], \lfloor\cdot\rfloor$
• not allowed to say $f(n) = \begin{cases} -n & \text{even} \\ -n & \text{odd} \end{cases}$

Ref. G.H. Hardy. See Concrete Mathematics by Graham, Knuth, Patashnik.
Typical run-times and how they compare

\[ \log n \]  - binary search
\[ n \]  - find max
\[ n \log n \]  - sorting
\[ n^2 \]  - insertion sort
\[ n^3 \]  - multiplying two \( n \times n \) matrices
\[ 2^n \]  - try all subsets
\[ n! \]  - try all ordering of a set (e.g., travelling salesman)

More: \( \sqrt{n} \), \( \log \log n \), \( (\log n)^2 \)

Ordering, where \( f(n) \ll g(n) \) means \( f(n) \in o(g(n)) \)

\[ 1 \ll \log \log n \ll \log n \ll (\log n)^2 \ll \sqrt{n} \ll n \ll n \log n \ll n^2 \ll n^3 \ll 2^n \ll n! \]

Also

- \( n^a \in o(n^b) \) for \( b > a > 0 \)
- \( (\log n)^a \in o(n^b) \) for \( b > 0 \)
- \( n^a \in o(2^n) \)
Sometimes we will analyze an algorithm’s run-time in terms of several parameters.

E.g. 1. graph with $n$ vertices and $m$ edges
   $m$ is $O(n^2)$ but $O(nm)$ can be better than $O(n^2)$

E.g. 2. input and output size
   e.g. Jarvis’ March $O(nh)$, $n =$ input size, $h =$ output size

Definition $f, g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$

$f(n, m)$ is $O(g(n, m))$ if $\exists$ constants $c, n_0, m_0$ s.t.

$f(n, m) \leq cg(n, m)$ for all $n \geq n_0, m \geq m_0$
Summary We analyze algorithms by analyzing the asymptotic rate of growth of the worst case run-time.

Does it really matter?

<table>
<thead>
<tr>
<th>Size of largest problem solvable in one hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>present computer</td>
</tr>
<tr>
<td>$n$</td>
</tr>
<tr>
<td>$n^2$</td>
</tr>
<tr>
<td>$n^3$</td>
</tr>
<tr>
<td>$n^5$</td>
</tr>
<tr>
<td>$2^n$</td>
</tr>
</tbody>
</table>