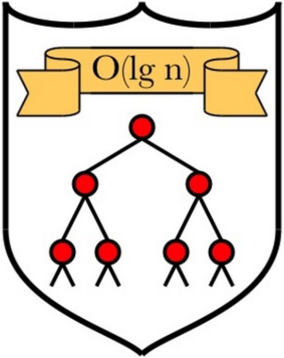
# Discrete Mathematics

## MATH1064, Lecture 16

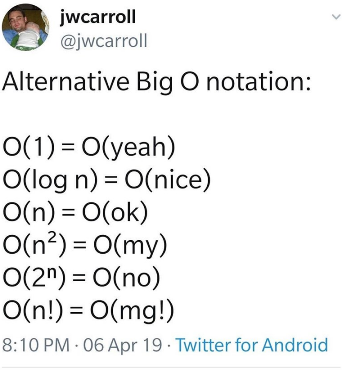
Jonathan Spreer



# Extra exercises for Lecture 16

Section 3.2: Problems 1–10

Section 3.3: Problems 1–4



# Number of operations: Best-case, worst-case, average-case

Search integers $a_1, \ldots, a_n$ for presence of integer called key.

```
procedure LinearSearch(key, n: integer; a_1, a_2, a_3, ..., a_n: integers)
begin
  i := 1                              {initializes the counter}
  while (i ≤ n and key ≠ a_i) do
    i := i + 1
  if i ≤ n then location := i         {successful search}
  else location := 0                  {unsuccessful search}
end {location is the subscript of the first array entry that equals key;
    location is 0 if key is not found}
```

Complexity $f(n) =$ number of elements examined until key is found

What is the best-case scenario? If key $= a_1$, then $f(n) = 1$.

Observation: $f(n)$ and number of steps within a constant multiple.
So best-case complexity of algorithm is $O(1)$.

What is the worst-case scenario?

If key $= a_n$, or key not in list, then $f(n) = n$.
So worst-case complexity of algorithm is $O(n)$ (in fact, it is in $\Theta(n)$).

What is the average-case scenario?

Assume *key* is in the list and equal to $a_k$ with probability $p = 1/n$.
Average number of elements examined is:

$$f(n) = 1 \cdot p + 2 \cdot p + \ldots + n \cdot p = p(\sum_{k=1}^{n} k) = \frac{1}{n} \cdot \frac{n(n+1)}{2} = (n+1)/2.$$

Average-case complexity of the algorithm is also $O(n)$ (in fact, $\Theta(n)$).

What happens if *key* is not necessarily in the list?

# Number of operations: Euclidean algorithm

```
function Euclid (a, b);
Input  : Two nonnegative integers a and b
Output: gcd(a, b)
if b = 0 then
  | return a;
else
  | return Euclid(b, a mod b);
end
```

A call of `Euclid` has

- 2 operations if $b = 0$, and
- 3 operations if $b > 0$.

How many steps are necessary until $b = 0$?

# Number of operations: Euclidean algorithm

```
function Euclid (a, b);
Input  : Two nonnegative integers a and b
Output: gcd(a, b)
if b = 0 then
  | return a;
else
  | return Euclid(b, a mod b);
end
```

Arguments passed to `Euclid`:

- $(a, b)$, $a \geq b$, $a \neq 0$;
- $(b, r_1)$ with $r_1 = a \mod b$,
  $b > r_1$ by the Quotient Remainder Theorem (Lecture 12);
- $(r_1, r_2)$ with $r_2 = b \mod r_1$;
- $(r_2, r_3)$ with $r_3 = r_1 \mod r_2$;
- etc.

# Number of operations: Euclidean algorithm

```
function Euclid (a, b);
Input  : Two nonnegative integers a and b
Output: gcd(a, b)
if b = 0 then
  | return a;
else
  | return Euclid(b, a mod b);
end
```

Idea: look at first argument every second step:

- $r_i = k \cdot r_{i+1} + r_{i+2}$, $k \in \mathbb{N}$
- We have $r_i > r_{i+1} > r_{i+2}$ and hence
- $k \geq 1$
- $r_i = k \cdot r_{i+1} + r_{i+2} \geq r_{i+1} + r_{i+2} > 2r_{i+2}$
- $r_{i+2} < r_i/2$

Size of arguments decrease by at least half every second step.

# Number of operations: Euclidean algorithm

```
function Euclid (a, b);
Input  : Two nonnegative integers a and b
Output: gcd(a, b)
if b = 0 then
|   return a;
else
|   return Euclid(b, a  mod b);
end
```

After at most $2\log_2(a)$ calls of `Euclid`, the arguments must be

$$\gcd(a,b) \quad \text{and} \quad 0.$$

Running time: $3(2\log_2(a)) - 1 \in O(\log(a))$.

# What to make of theoretical bounds?

Suppose algorithm $A$ has time-complexity $f(n) \in O(n)$, and algorithm $B$ has time-complexity $g(n) \in O(n^2)$.

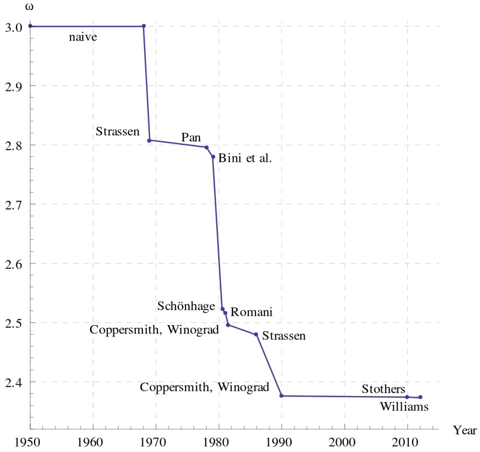
Suppose $f(n) = 1000n$ and $g(n) = n^2$.

Question: Is it always better to use algorithm A?

If the size of the input is not bigger than 1000, then $B$ is faster!

# What to make of theoretical bounds?

Matrix multiplication is in $O(n^{\omega})$ for $\omega =$




Note: matrix multiplication is in $\Omega(n^2)$.

# $O$-notation from a more practical angle

- For a simplified (mathematically less rigorous) explanation of $O$-notation, search:
  "Big-O notation in 5 minutes – The basics" on YouTube

- See https://www.bigocheatsheet.com/ for a list of running times of famous algorithms

Warning: The comments on this video suggest that this teaches you "all you need". But the level of this video is below the mathematical standard of this course.

# P vs NP

A decision problem is a yes/no question, for which we wish to find an algorithm.

Examples:

- Input: $k \in \mathbb{N}$.
  Question: Is $k$ prime?

- Input: Logical statement forms $P, Q$.
  Question: Is $P \equiv Q$?

P vs NP is about which decision problems you can solve quickly, and which problems are inherently difficult.

# What do we mean by "quickly"?

Let $n$ measure the size of the input:

- Input: Logical statement forms $P, Q$.
  Question: Is $P \equiv Q$?
  Input size: $n =$ number of symbols used in $P$ and $Q$

- Input: $k \in \mathbb{N}$.
  Question: Is $k$ prime?
  Input size: $n =$ number of digits in $k$

An algorithm is considered fast if its running time is bounded by a polynomial.

Fast running times: $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^5)$, $O(n^{100})$

Slow running times: $O(C^n)$, $C > 1$, $O(n!)$, $O(e^{e^n})$

## The classes P and NP

A decision problem is in the class P if you can solve it quickly.

- Input: $k, \ell \in \mathbb{N}$. Question: Are $k$ and $\ell$ coprime?
  Solution: Euclidean algorithm!

A decision problem is in the class NP if, when the answer is "yes", I can give you information that lets you verify my solution quickly.

- Input: $k \in \mathbb{N}$. Question: Is $k$ composite?
  Information: I give you the prime factorisation of $k$.

We know how to "quickly" test whether $k$ is compositite.

# P vs NP

The question P vs NP asks: are P and NP the same?

That is, if a "yes" solution is fast to verify, does that mean the problem must be fast to solve?

The hardest problems in NP are called NP-complete:
a fast solution to any one of these would give a fast solution to every problem in NP!

Some NP-complete problems:

- Input: A logical statement form $P$. Is $P$ satisfiable?
- Input: A set $S \subseteq \mathbb{Z}$. Does $S$ have a subset whose sum is 0?

Many people think that P $\neq$ NP… but some people do not!

Have a look at https://www.win.tue.nl/~gwoegi/P-versus-NP.htm

# True or false?

$$\forall a \in \mathbb{N}, \quad a^n \in O(n!)$$