Admin

- Fifth quiz available from Monday
  - Sixth quiz available next Monday

- Sudoku assignment
  - Cut-off date is **15th March 4pm**
  - No more help with Sudoku assignment in Virtual Contact Hours from now on
  - Book me for office hours if you need help

- Primes assignment
  - Worksheet made **available next Monday at 11am**
  - Only involves programming tasks and submission of single js file
  - Next week's VCH devoted to this assignment
  - Deadline **15th March 4pm**
  - Cut-off date **29th March 4pm**

“Big O” notation recipe

1) Treat all non-zero constants as 1
2) Include in brackets only the **fastest growing part as *n* increases**

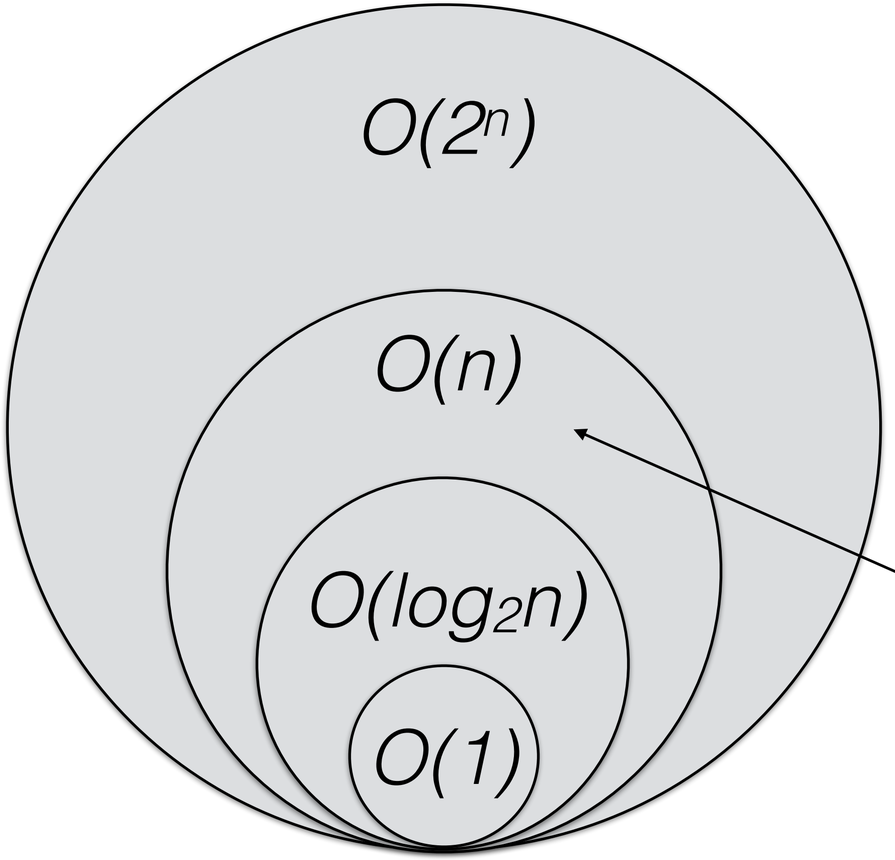
$$O(1) < O(\log_2 n) < O(n) < O(n^2) < O(n^k) \; (k>2) < O(2^n) < O(2^{2n})$$

"Big O" really says: function will grow **at most as fast** as the thing in the brackets

e.g. $f(n) = 3n + 2$ will be in $O(n)$, **AND** also in $O(2^n)$
BUT not in $O(\log_2 n)$

Whatever function you have will belong in a class and then many more

Set inclusions



f(n) = 3n + 2

WEB

1. Connect to **www.wooclap.com/PS4CS6**
2. You can participate

Bases

What about $O(log_3 n)$?

It doesn't matter which base you choose as long as it is larger than 1

e.g. $O(log_2 n) = O(log_3 n)$

Why?

What about $O(10^n)$ instead of $O(2^n)$?

Discuss this during the **Review Seminar**

$O(log_3 n)$

Rewrite into base 2 using change of base formula

$log_2 n = log_3 n \,/\, log_3 2$

$log_3 n = log_3 2 \; log_2 n$

$log_3 n$ is just $log_2 n$ multiplied by $log_3 2$ (constant)

We don't need to worry about this constant

$O(log_3 n) = O(log_2 n)$

We use O(log n) for this reason

$O(10^n)$

Rewrite into base 2, i.e. $10^n = 2^m$, find m

$$\log_2 10^n = \log_2 2^m$$
$$n\log_2 10 = m\log_2 2$$
$$m = n\log_2 10 \text{ (using change of base)}$$

$$10^n = (2^n)^c \qquad c = \log_2 10 > 3$$

We **cannot** ignore this constant, cf. n and n^3

Different base can result in faster growth!

$$O(1) = O(1^n) < O(k^n) < \mathbf{O(2^n) < O(10^n)}$$

$1<k<2$

This was pretty mathematical

1) What made the most sense to you
2) What made the least sense
3) When do constants matter?
4) Can you think of a “Big O” class not mentioned yet?

This was pretty mathematical

1) What made the most sense to you
2) What made the least sense
3) When do constants matter?

When they are a power e.g. $O(n^2)$ versus $O(n)$

4) Can you think of a “Big O” class not mentioned yet?

My favourite: $O(\log^2 n)$

```
function sumOfFactorials(n) {
    if (n===0) {
        return 1;
    }
    var a = 1;
    for (var i = 1; i <= n; i++) {
        var b = 1;
        for (var j = 1; j <= i; j++) {
            b = b * j;
        }
        a = a + b;
    }
    return a;
}
```

How many operations (in “Big O” notation) in *n* are required in a RAM implementation?

```
function sumOfFactorials(n) {
    if (n===0) {
        return 1;
    }
    var a = 1;
    for (var i = 1; i <= n; i++) {
        var b = 1;
        for (var j = 1; j <= i; j++) {
            b = b * j;
        }
        a = a + b;
    }
    return a;
}
```

How many operations (in “Big O” notation) in *n* are required in a RAM implementation?

$O(n^2)$

```
function factorialPlusSum(n) {
    if (n===0) {
        return 1;
    }
    var a = 1;
    for (var i = 1; i <= n; i++){
        a = a * j;
    }
    var b = 0;
    for (var i = 1; i <= n; i++){
        b = b + j;
    }
    return a + b;
}
```

How many operations (in “Big O” notation) in *n* are required in a RAM implementation?

```
function factorialPlusSum(n) {
    if (n===0) {
        return 1;
    }
    var a = 1;
    for (var i = 1; i <= n; i++){
        a = a * j;
    }
    var b = 0;
    for (var i = 1; i <= n; i++){
        b = b + j;
    }
    return a + b;
}
```

How many operations (in “Big O” notation) in *n* are required in a RAM implementation?

O(n)