In [1]:

```
# Set up environment with correct dependencies
using Pkg
Pkg.activate(".")
Pkg.instantiate()
```

```
 Activating environment at `~/GitHub/MathSys/teaching/MA934/MA934-slid
es/Project.toml`
```

In [2]:

```
using Plots
using LaTeXStrings
pyplot()
# Set default fonts for all plots
fnt = Plots.font("DejaVu Sans", 8.0)
default(titlefont=fnt, guidefont=fnt, tickfont=fnt, legendfont=fnt)
```

MA934

Divide and conquer algorithms

Estimating the computational complexity of numerical computations

Big O notation

Given functions, $f(x)$ and $g(x)$, we say that $f(x) = \mathcal{O}(g(x))$ if there exist x_0 and $K > 0$ such that

$$|f(x)| \leq K\, g(x)$$

for all $x > x_0$.
Used to describe an upper bound on the growth of a function, $f(x)$, as its argument, x, tends to ∞.

FLOPS and computational complexity

FLOPS : Floating point operations per second

The *Computational complexity* of an algorithm is the rate at which the total number of operations (additions, multiplications, comparisons) required to solve a problem of size n grows with n.

We denote this number of operations by $F(n)$.

The more efficient the algorithm, the slower the $F(n)$ grows with n.

Example : standard matrix multiplication

Consider calculating $C = A\,B$ where A and B are $n \times n$ matrices.
Using the standard algorithm, the $i\ j$ entry of C is:

$$c_{i\,j} = \sum_{k=1}^{n} a_{i\,k}\, b_{k\,j}.$$

Needs n multiplications and $n-1$ additions, giving $2n-1$ ops in total. Since there are n^2 entries, the total number of ops required is $n^2\ (2n-1)$.

Thus matrix multiplication has computational complexity $F(n) = \mathcal{O}(n^3)$.

Aside : Benchmarking in Julia

The BenchmarkTools.jl package provides useful tools for measuring execution times and other benchmarks. Let's check matrix multiplication:

In [3]:

```
using BenchmarkTools
times = Float64[]; sizes = [2^i for i in 1:8]
for n in sizes
    A = rand(n,n); B = rand(n,n)
    bm = @benchmark ($A)*($B)
    push!(times, median(bm.times))
    println(n, " ", median(bm.times))
end
```

```
2 82.28247422680413
4 288.0388692579505
8 419.61650485436894
16 1292.55
32 5966.142857142857
64 28770.5
128 248308.0
256 1.464875e6
```

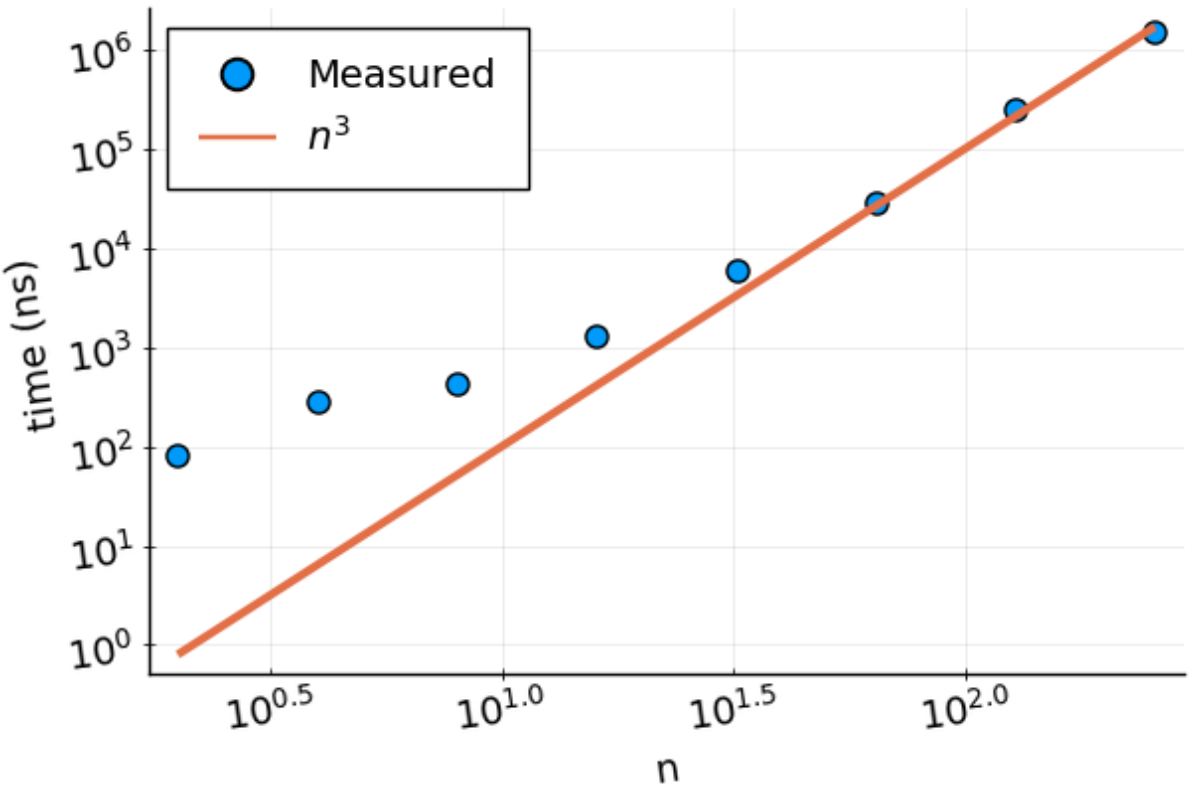
Aside: Benchmarking in Julia

Let's plot the results:

In [4]:

```
scatter(sizes, times, yscale=:log10, xscale=:log10,
    label="Measured",ylabel="time (ns)", xlabel="n", markersize=8)
plot!(sizes, 0.1*sizes.^3.0, label = L"n^3", linewidth=3)
```

Out[4]:



A recursive approach to matrix multiplication

Assuming that $n = 2^m$, divide $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ into smaller matrices of size $\frac{n}{2} \times \frac{n}{2}$,

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{pmatrix} \quad \mathbf{C} = \begin{pmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{pmatrix}$$

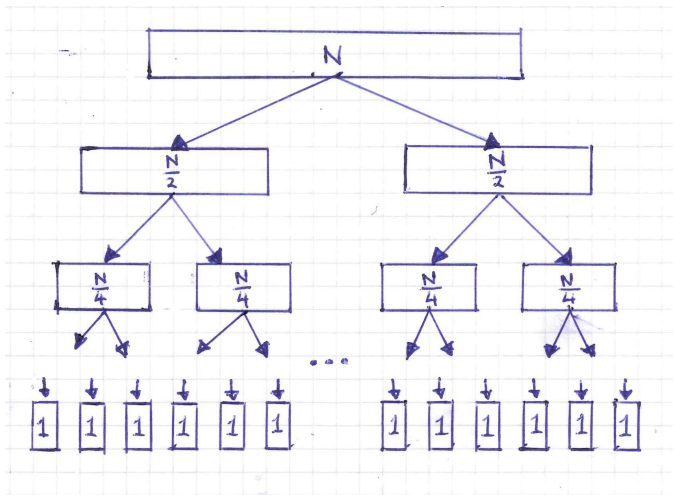
The matrices $C_{i\,j}$ are

$$\mathbf{C}_{11} = \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21}$$
$$\mathbf{C}_{12} = \mathbf{A}_{11}\mathbf{B}_{21} + \mathbf{A}_{12}\mathbf{B}_{22}$$
$$\mathbf{C}_{21} = \mathbf{A}_{21}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{21}$$
$$\mathbf{C}_{22} = \mathbf{A}_{21}\mathbf{B}_{21} + \mathbf{A}_{22}\mathbf{B}_{22}.$$

Divide and conquer strategy

Multiplication of $n \times n$ matrices can be expressed as multiplication (and addition) of $\frac{n}{2} \times \frac{n}{2}$ matrices.
Recursion leads to trivial 1×1 matrices.

This is an example of what is called a "divide and conquer" strategy.



Divide-and-conquer: break a problem into smaller subproblems of the same type, recursively solve the subproblems, and finally combine the solutions.

Computational complexity of recursive matrix multiplication

Is the recursive strategy better or worse than the iterative one?

Let $F(n)$ be the ops required to multiply 2 matrices of size $n \times n$.
We must have

$$F(n) = 8\,F\left(\frac{n}{2}\right) + 4\left(\frac{n}{2}\right)^2,$$

since we have to do 8 multiplications and 4 additions of size $\frac{n}{2}$.

When the recursion reaches $n = 1$, we have $F(1) = 1$.

Solving for $F(n)$

How do we solve such equations?
Trick: let

$$n = 2^p$$
$$a_p = F(2^p).$$

This gives a linear first order recursion relation

$$a_p = 8\,a_{p-1} + 2^{2p}$$

with $a_0 = 1$.

The solution is

$$a_p = 2^{2p}\left(2^{p+1} - 1\right).$$

Returning to the original variables: $p = \log_2 n$.

$$F(n) = n^2\,(2n - 1).$$

In this case, the divide and conquer strategy gives the same answer as the iterative strategy ... but this is not always the case.

The "Master Theorem"

Let a and b be integers greater than or equal to 1 (usually $b = 2$). Let c and d be positive and real.

Given a recurrence relation of the form

$$F(n) = \begin{cases} a\,F(n/b) + n^c & \text{if } n > 1 \\ d & \text{if } n = 1, \end{cases}$$

then if n is a power of b, there are three cases:

1. if $\log_b(a) < c$ then $F(n) = O(n^c)$
2. if $\log_b(a) = c$ then $F(n) = O(n^c \log(n))$
3. if $\log_b(a) > c$ then $F(n) = O(n^{\log_b(a)})$.

Matrix multiplication again: Strassen's algorithm

An alternative (very non-obvious) divide and conquer strategy is:

$$\begin{aligned} \mathbf{C}_{11} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 & \mathbf{C}_{12} &= \mathbf{M}_3 + \mathbf{M}_5 \\ \mathbf{C}_{21} &= \mathbf{M}_2 + \mathbf{M}_4 & \mathbf{C}_{22} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6, \end{aligned}$$

where $\mathbf{M}_1$ to $\mathbf{M}_7$ are

$$\begin{aligned} \mathbf{M}_1 &= (\mathbf{A}_{11} + \mathbf{A}_{22})\,(\mathbf{B}_{11} + \mathbf{B}_{22}) & \mathbf{M}_2 &= (\mathbf{A}_{21} + \mathbf{A}_{22})\,\mathbf{B}_{11} \\ \mathbf{M}_3 &= \mathbf{A}_{11}\,(\mathbf{B}_{12} - \mathbf{B}_{22}) & \mathbf{M}_4 &= \mathbf{A}_{22}\,(\mathbf{B}_{21} + \mathbf{B}_1) \\ \mathbf{M}_5 &= (\mathbf{A}_{11} + \mathbf{A}_{12})\,\mathbf{B}_{22} & \mathbf{M}_6 &= (\mathbf{A}_{21} - \mathbf{A}_{11})\,(\mathbf{B}_{11} + \mathbf{B}_{12}) \\ \mathbf{M}_7 &= (\mathbf{A}_{12} - \mathbf{A}_{22})\,(\mathbf{B}_{21} + \mathbf{B}_{22}) & . \end{aligned}$$

Computational complexity of Strassen multiplication

These crazy formulae were dreamt up by Strassen (1969). The key point is that there are only 7 multiplications rather than 8. We have:

$$F(n) = 7\,F\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2$$

, with $F(1) = 1$. The solution is

$$F(n) = 7n^{\log_2(7)} - 6n^2 = \mathcal{O}(n^{\log_2(7)}).$$

Check that this is consistent with the master theorem.

Note that $\log_2(7) \approx 2.807 < 3$. Amazing!

Solving linear recursion relations

We don't need to memorise the master theorem: linear recursion relations are very similar to linear differential equations.

1st order case:

$$a_n = b_1\,a_{n-1} + f(n) \quad \text{with initial condition } a_1 = A_1.$$

2nd order case:

$$a_n = b_1\,a_{n-1} + b_2\,a_{n-2} + f(n) \quad \text{with initial conditions } a_1 = A_1,\ a_2 = A_2$$

Solution procedure is the same in both cases:

Solving linear recursion relations

1. Find the *general solution* of the homogeneous equation,
$$a_n = b_1\, a_{n-1} \quad \text{or} \quad a_n = b_1\, a_{n-1} + b_2\, a_{n-2}$$
with the ansatz $a_n = x^n$.
The solution will usually be of the form
$$a_n = C_1\, x^n \ \text{ or } \ a_n = C_1\, x_1^n + C_2\, x_2^n,$$
where the C's are arbitrary constants.
2. Find a *particular solution*, α_n, of the inhomogeneous equation.
3. The full solution is
$$a_n = C_1\, x^n + \alpha_n \ \text{ or } \ a_n = C_1\, x_1^n + C_2\, x_2^n + \alpha_n,$$
Use the initial conditions to evaluate the constants, C.

Solving linear recursion relations

The tricky part is step 2 - need to "guess" a particular solution.

A good guess is often a term, $\text{const} \times f(n)$, proportional to $f(n)$ (or sometimes a polynomial in $f(n)$).

As for resonant ODEs, it can happen that the solution of the homogeneous equation is already proportional to $f(n)$

In this case a good guess for the particular solution is $\text{const} \times n\, f(n)$.

See the excellent online notes available **here (https://www.tutorialspoint.com/discrete_mathematics/discrete_mathematics_recurrence_relation.htm)**.