Algorithm complexity

A Gentle Introduction

Read more:
http://discrete.gr/complexity/

Victor Ananyev, QFT

Number of instructions is a measure of complexity

```
var M = A[ 0 ];

for ( var i = 0; i < n; ++i ) {
    if ( A[ i ] >= M ) {
        M = A[ i ];
    }
}
```

Var M = A[0]		i = 0		i < n ++i		A[i] >= M M = A[i]
1	+	1	+	n	+	n

$$2n + 2$$

O(n) notation — easy way to represent complexity

Due to lots of assumptions made to estimate number of instructions, the only significant value is an asymptotic behavior.

$$f(x) = O(g(x)).$$

$$|f(x)| \leq M|g(x)| \text{ for all } x \geq x_0 .$$

Examples:

f(n) = n^6 + 3n = O(n^6)

f(n) = 2^n + 12 = O(2^n)

f(n) = 3^n + 2^n = O(3^n)

f(n) = n^n + n = O(n^n)

Example. Searching for element in an array

```php
<?php
    $exists = false;
    for ( $i = 0; $i < n; ++$i ) {
        if ( $A[ $i ] == $value ) {
            $exists = true;
            break;
        }
    }
?>
```

?

Example. Searching for element in an array

```php
<?php
    $exists = false;
    for ( $i = 0; $i < n; ++$i ) {
        if ( $A[ $i ] == $value ) {
            $exists = true;
            break;
        }
    }
?>
```

O(n)

Example. Checking for duplicates

```
bool duplicate = false;
for ( int i = 0; i < n; ++i ) {
    for ( int j = 0; j < n; ++j ) {
        if ( i != j && A[ i ] == A[ j ] ) {
            duplicate = true;
            break;
        }
    }
    if ( duplicate ) {
        break;
    }
}
```

?

Example. Checking for duplicates

```
bool duplicate = false;
for ( int i = 0; i < n; ++i ) {
    for ( int j = 0; j < n; ++j ) {
        if ( i != j && A[ i ] == A[ j ] ) {
            duplicate = true;
            break;
        }
    }
    if ( duplicate ) {
        break;
    }
}
```

O(n^2)

Example. Binary search. Logarithmic complexity

```
def binarySearch( A, n, value ):
    if n = 1:
        if A[ 0 ] = value:
            return true
        else:
            return false
    if value < A[ n / 2 ]:
        return binarySearch( A[ 0...( n / 2 - 1 ) ], n / 2 - 1, value )
    else if value > A[ n / 2 ]:
        return binarySearch( A[ ( n / 2 + 1 )...n ], n / 2 - 1, value )
    else:
        return true
```

O(log(n))

True example. Merge sort

```
def mergeSort( A, n ):
    if n = 1:
        return A # it is already sorted
    middle = floor( n / 2 )
    leftHalf = A[ 1...middle ]
    rightHalf = A[ ( middle + 1 )...n ]
    return merge( mergeSort( leftHalf, middle ), mergeSort( rightHalf, n - middle ) )
```

?

```
def merge( A, B ):
    if empty( A ):
        return B
    if empty( B ):
        return A
    if A[ 0 ] < B[ 0 ]:
        return concat( A[ 0 ], merge( A[ 1...A_n ], B ) )
    else:
        return concat( B[ 0 ], merge( A, B[ 1...B_n ] ) )
```

True example. Merge sort

```
def mergeSort( A, n ):
    if n = 1:
        return A # it is already sorted
    middle = floor( n / 2 )
    leftHalf = A[ 1...middle ]
    rightHalf = A[ ( middle + 1 )...n ]
    return merge( mergeSort( leftHalf, middle ), mergeSort( rightHalf, n - middle ) )
```

$O(n \cdot \log(n))$

```
def merge( A, B ):
    if empty( A ):
        return B
    if empty( B ):
        return A
    if A[ 0 ] < B[ 0 ]:
        return concat( A[ 0 ], merge( A[ 1...A_n ], B ) )
    else:
        return concat( B[ 0 ], merge( A, B[ 1...B_n ] ) )
```

Surprising example. Matrix multiplication

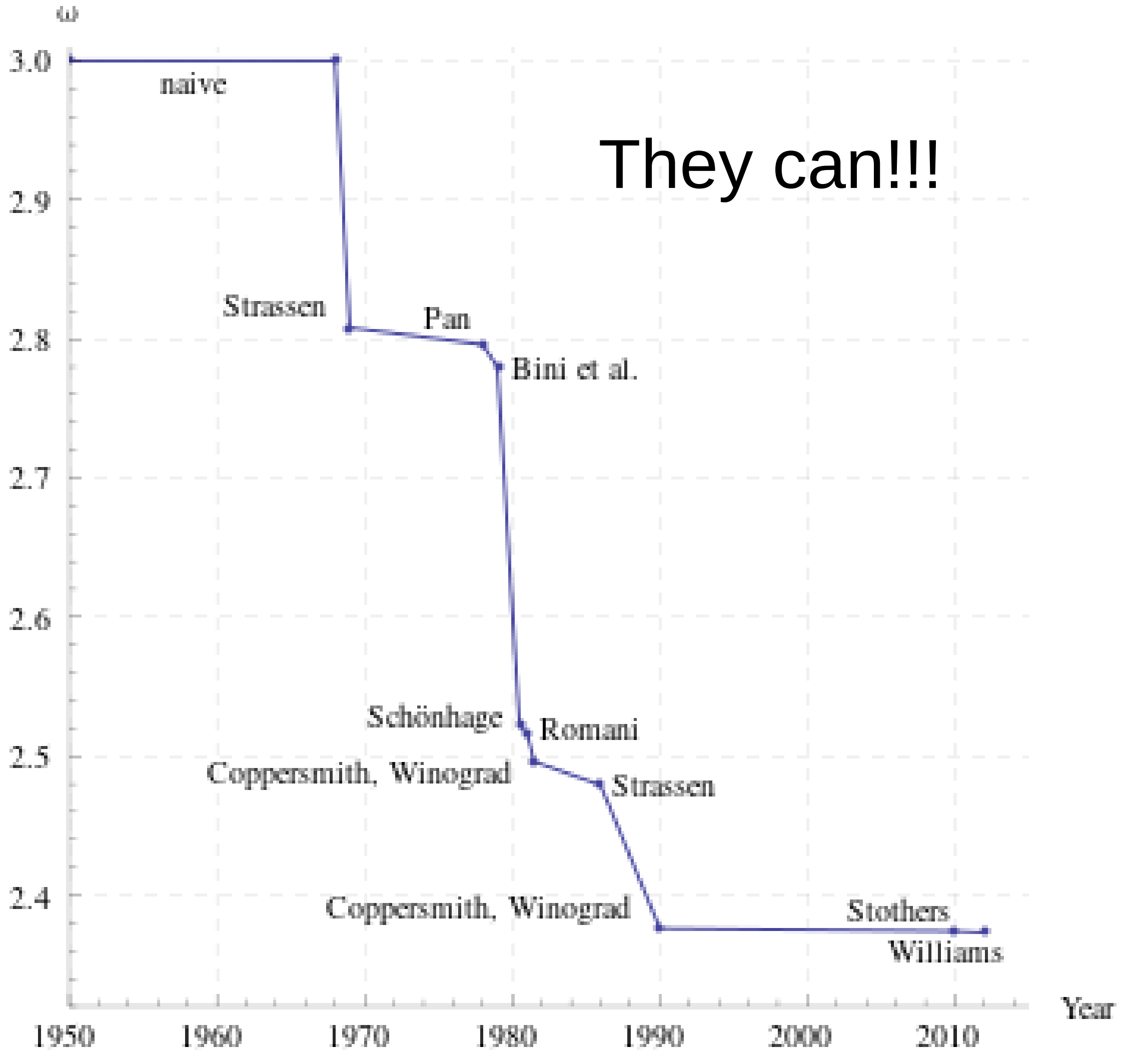
```
int main(void)
 {
        const int ROW=4, COL=4, INNER=4;
        int A[ROW][INNER], int B[INNER][COL], int C[ROW][COL];
        for (int row = 0; row != ROW; ++row)
        {
                for (int col = 0; col != COL; ++col)
                {
                        int sum = 0;
                        for (int inner = 0; inner != INNER; ++inner)
                        {
                                sum += A[row][inner] * B[inner][col];
                        }
                        C[row][col] = sum;
                }
        }
        return 0;
 }
```

Surprising example. Matrix multiplication

O(n^3)

Who can faster?

Surprising example. Matrix multiplication



They can!!!

Thanks

Backup. Strassen

$$\mathbf{C}_{1,1} = \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{1,2} = \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2}$$
$$\mathbf{C}_{2,1} = \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{2,2} = \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2}$$

$$\mathbf{M}_1 := (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2})$$
$$\mathbf{M}_2 := (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1}$$
$$\mathbf{M}_3 := \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2})$$
$$\mathbf{M}_4 := \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1})$$
$$\mathbf{M}_5 := (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2}$$
$$\mathbf{M}_6 := (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2})$$
$$\mathbf{M}_7 := (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})$$

$$\mathbf{C}_{1,1} = \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7$$
$$\mathbf{C}_{1,2} = \mathbf{M}_3 + \mathbf{M}_5$$
$$\mathbf{C}_{2,1} = \mathbf{M}_2 + \mathbf{M}_4$$
$$\mathbf{C}_{2,2} = \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6$$