Big-Oh O() - ways of measuring it; def; properties

- time complexity
- memory complexity

```
int x = 10;
int y = 5;
int z = x + y;
```

] Basic operations: assignments, arithmetic — constant time per operation

Trends in running time as a fn of the size of input to the code;

Eg above - running time constant

```
if z < 1  System.out.println(z);
```

z < 1: comparison (basic)

System.out.println(z) ↳ const time because this is simple I/O

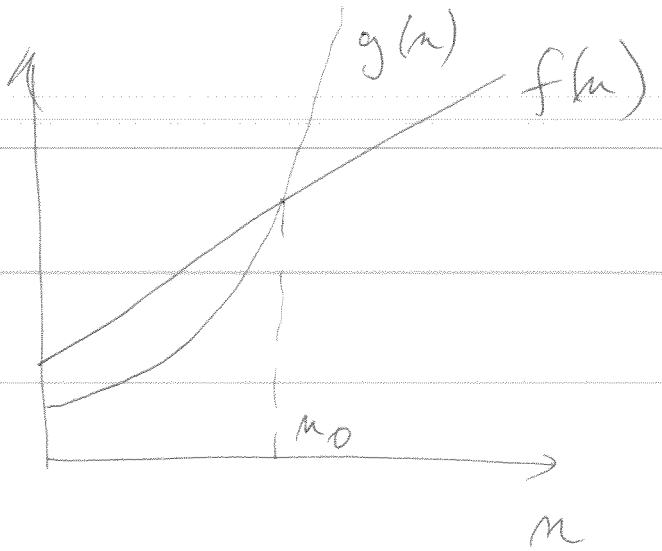
Running time of whole code = sum of running times of pieces

```
int n = 10;
int[] a = new int[n];
for (int i=0; i<a.length; i++)
        a[i] = 1;          <- constant
for (int i=0; i<a.length; i=i+2) {
        a[i] = 2;
}
```

(left margin, beside the two loops: O() ... linear in n)

First loop: n * constant (n → size of a) "O(n)"

Second loop: $\frac{n}{2}$ * constant ~ n * constant



g(n) quadratic

at some point it will dominate f(n)

$f(n) \in O(g(n))$

if at some point $g(n) \cdot c \geq f(n)$

there exists n_0 s.t for all $n \geq n_0$

$g(n) \cdot c \geq f(n)$ for some c

for any constant c_1 $O(c) \rightarrow O(1)$

constant time complexity

for any linear fn $an+b \rightarrow O(n)$

$an+b \in O(n^2)$

$an^2+bn+c \in O(n^2)$

$\in O(n^3)$

A fn $f(n)$ belongs to many classes $O(g(n))$

Usually we want a tight fit

Hierarchy of $O()$ classes; example algs in each class

Big-Oh definition: A function $f(n) \in O(g(n))$ if there exist constants c, n_0 s.t. $\forall n \geq n_0$, $f(n) \leq c \cdot g(n)$

↑ for all

Can be used for proofs

E.g. Prove $5 + 3n^2 \in O(1 + n^2)$

Pick c, n_0 show def. holds

Pick $c = 5$ $\quad 5 + 3n^2 \leq 5 + 5n^2$

true $\forall n \geq 0$ $\quad (n_0 = 0)$

Symbols $\forall$ = for all

$\exists$ = exists

n = size of input (e.g. array...)

$f(n)$ = running time (memory)

$g(n)$ = "tightest" complexity class

Example: Prove $n \sin n \in O(n)$

$\sin n \leq 1 \quad c = 1$, n_0 = anything eg. 1

$n \sin n \leq n \cdot 1$

$\forall n \geq n_0 \quad f(n) \leq c \cdot g(n)$

$\forall n \geq 1 \quad n \sin n \leq 1 \cdot n$

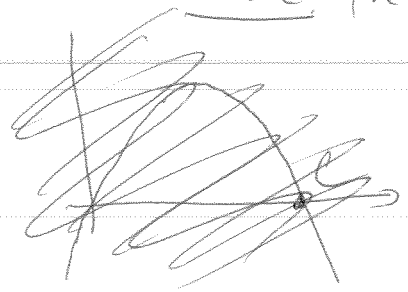
Prove that $n^2 \notin O(n)$

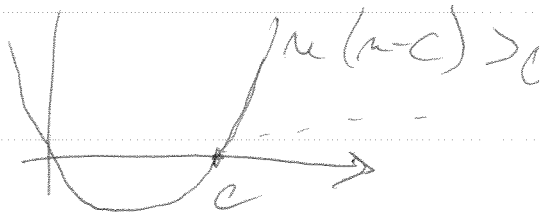
Proof by contradiction

Assume that $n^2 \in O(n)$

Exist n_0, c s.t $\forall n \geq n_0 \quad n^2 < c \cdot n$

False! $n(n-c) < 0 \quad \forall n > n_0$





Contradiction $\Rightarrow n^2 \notin O(n)$

Run a sequence of operations

```
int n = 10
int m = 1
int p = n + m
...
```

Running time

$T(\blacksquare) = T_1 + T_2 + T_3 + \ldots$

↑ whole sequence of operations

Theorem (Sum rule): Suppose we have f_1, f_2, $f_1 \in O(g(n))$, $f_2 \in O(g(n))$ then $f_1 + f_2 \in O(g(n))$

Proof: $f_1 \in O(g(n)) \Rightarrow \exists n_{0,1}, c_1$ s.t. $\forall n \geq n_{0,1}$ $f_1(n) \leq c_1 g(n)$

$f_2 \in O(g(n)) \Rightarrow \exists n_{0,2}, c_2$ s.t. $\forall n \geq n_{0,2}$ $f_2(n) \leq c_2 g(n)$

$f_1(n) + f_2(n) \leq (c_1 + c_2) g(n)$ $\forall n \geq \max(n_{0,1}, n_{0,2})$

Def of $O()$ applies with $n_0 = \max(n_{0,1}, n_{0,2})$
$c = c_1 + c_2$

Sequence of elementary operations

$c_1 + c_2 \dots$ $\in O(1)$ $\xrightarrow{\text{sum rule}}$ sequence $\in O(1)$

```
int n = 10;                              → O(1) ⊂ O(n)
int[] a = new int[n];                    → O(1) ⊂ O(n)
for (int i = 0; i < a.length; i++)       O(n)
    a[i] = 1;
for (int j = 0; j < n*n; j++)            O(n^2)
    System.out.println(j)
```

(first three statements: $O(n)$; $O(n) \subset O(n^2)$; whole code: $O(n^2)$)

Piece of code $\Rightarrow O()$ is going to be dominated by most complicate loops / fn. calls

```
int n=10;
int m=5;
for (int i=0; i<n; i++)
    for(int j=0; j<m; j++)
        System.out.println(i*j);
```

≈ m * constant

n * running time of inside

n * m * const => O(n·m)

Nesting loops → multiplication in running time

Product rule $f_1(n) \in O(g_1(n))$; $f_2(n) \in O(g_2(n))$

then $f_1(n) * f_2(n) \in O(g_1(n) * g_2(n))$

Proof: use def

$\exists n_1, c_1$ s.t. $f_1(n) \le c_1 \cdot g_1(n)$ $\forall n \ge n_1$

$\exists n_2\ c_2$ s.t. $f_2(n) \le c_2 g_2(n)$ $\forall n \ge n_2$

$f_1(n) f_2(n) \le (c_1 c_2)\, g_1(n) g_2(n)$ $\forall n \ge \max(n_1, n_2)$

Def applies! c

"Deep" loops → complicated, dominate O()