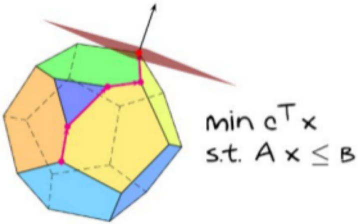
## How efficient is the simplex method?

- Algorithms and their analysis
- $O$-notation



# Algorithms

An algorithm is a finite set of instructions, used in common programming languages, like

- arithmetic operations
- comparisons
- conditional statements
- read/write instructions
- etc.

The *running time* of the algorithm is the number of instructions that it carries out. (Function depending on the *length of input*)

# Algorithms

An algorithm is a finite set of instructions, used in common programming languages, like

- arithmetic operations
- comparisons
- conditional statements
- read/write instructions
- etc.

The *running time* of the algorithm is the number of instructions that it carries out. (Function depending on the *length of input*)

First approximation: Count the number of *elementary operations* like

- arithmetic operations (addition, subtraction, multiplication, division)
- Comparisons such as $<, \leqslant, =$ etc.

# Example: Inner product

```
def multiply (a,b):
    s = 0
    for i in range(len(a)):
        s = s+ a[i]*b[i]
    return s
```

$\underbrace{\text{range(len(a))}}_{= [0, \dots, \mathrm{len}(a)-1]}$

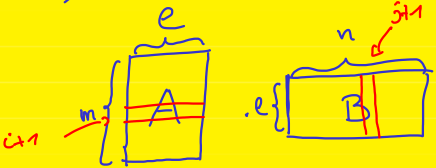
$0, \dots$

$$a \cdot b = \sum_{i=1}^{n} a_i \cdot b_i$$

- $a, b \in \mathbb{R}^n$
- Number of multiplications: $n$
- Number of additions: $n$
- Total: $2 \cdot n$ *elementary operations*
- Length of input is $2 \cdot n$.

# Example: Matrix multiplication

$A \in \mathbb{R}^{m \times l}, \quad B \in \mathbb{R}^{l \times n}$




$m$ $n$

```
def multiply(A,B):
    m = A.rows
    l = A.cols
    n = B.cols
    C = Matrix.zeros(m,n)

    for i in range(m):
        for j in range(n):
            s = 0
            for k in range(l):
                s = s + A[i,k]*B[k,j]
            C[i,j] = s
    return C
```

$2l$ arithmetic operations

Total: $m \cdot n \cdot 2l$ arithmetic operations !!

# Example: Matrix multiplication (cont.)

- $A \in \mathbb{R}^{m \times l}$, $B \in \mathbb{R}^{l \times n}$
- Multiplications: $m \cdot n \cdot l$
- Additions: $m \cdot n \cdot l$
- Length of input is $m \cdot l + l \cdot n$

Total: $2 \cdot m \cdot n \cdot l$ arithmetic operations

# Example: Matrix multiplication (cont.)

- $A \in \mathbb{R}^{m \times l}$, $B \in \mathbb{R}^{l \times n}$
- Multiplications: $m \cdot n \cdot l$
- Additions: $m \cdot n \cdot l$
- Length of input is $m \cdot l + l \cdot n$

- In these examples exact counting is possible.
- We are interested in the *rate of growth* of the number of elementary operations.

in terms of input length

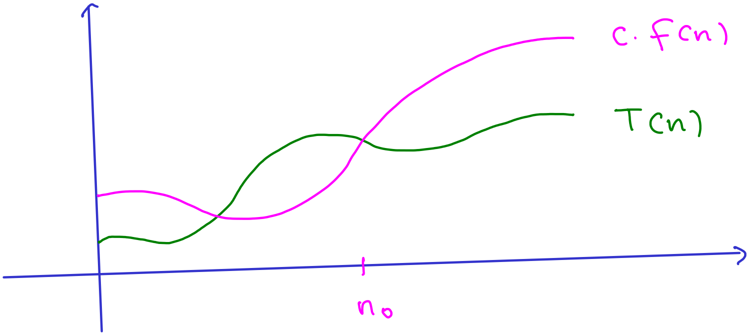
# $O, \Omega, \Theta$-notation

Let $T, f : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ be functions

- $T(n) = O(f(n))$, if there exist positive constants $n_o \in \mathbb{N}$ and $c \in \mathbb{R}_{>0}$ with

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

$$T(n) \leqslant c \cdot f(n) \text{ for all } n \geqslant n_0.$$



# $O, \Omega, \Theta$-notation

Let $T, f : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ be functions

- $T(n) = O(f(n))$, if there exist positive constants $n_o \in \mathbb{N}$ and $c \in \mathbb{R}_{>0}$ with

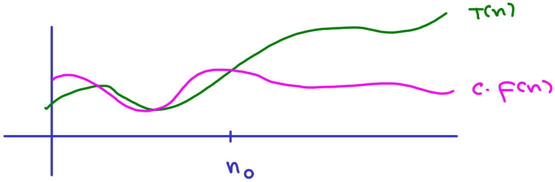
$$T(n) \leqslant c \cdot f(n) \text{ for all } n \geqslant n_0.$$

- $T(n) = \Omega(f(n))$, if there exist constants $n_o \in \mathbb{N}$ and $c \in \mathbb{R}_{>0}$ with

$$T(n) \geqslant c \cdot f(n) \text{ for all } n \geqslant n_0.$$

- $T(n) = \Theta(f(n))$ if

$$T(n) = O(f(n)) \text{ and } T(n) = \Omega(f(n)).$$



# $O, \Omega, \Theta$-notation

Let $T, f : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ be functions

- $T(n) = O(f(n))$, if there exist positive constants $n_o \in \mathbb{N}$ and $c \in \mathbb{R}_{>0}$ with
$$T(n) \leqslant c \cdot f(n) \text{ for all } n \geqslant n_0.$$
- $T(n) = \Omega(f(n))$, if there exist constants $n_o \in \mathbb{N}$ and $c \in \mathbb{R}_{>0}$ with
$$T(n) \geqslant c \cdot f(n) \text{ for all } n \geqslant n_0.$$
- $T(n) = \Theta(f(n))$ if
$$T(n) = O(f(n)) \text{ and } T(n) = \Omega(f(n)).$$

## Example

The function $T(n) = 2n^2 + 3n + 1$ is in $O(n^2)$, since for all $n \geqslant 1$ one has $2n^2 + 3n + 1 \leqslant 6n^2$. Here $n_0 = 1$ and $c = 6$.

# $O, \Omega, \Theta$-notation

Let $T, f : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ be functions

- $T(n) = O(f(n))$, if there exist positive constants $n_o \in \mathbb{N}$ and $c \in \mathbb{R}_{>0}$ with
$$T(n) \leqslant c \cdot f(n) \text{ for all } n \geqslant n_0.$$
- $T(n) = \Omega(f(n))$, if there exist constants $n_o \in \mathbb{N}$ and $c \in \mathbb{R}_{>0}$ with
$$T(n) \geqslant c \cdot f(n) \text{ for all } n \geqslant n_0.$$
- $T(n) = \Theta(f(n))$ if
$$T(n) = O(f(n)) \text{ and } T(n) = \Omega(f(n)).$$

## Example

The function $T(n) = 2n^2 + 3n + 1$ is in $O(n^2)$, since for all $n \geqslant 1$ one has $2n^2 + 3n + 1 \leqslant 6n^2$. Here $n_0 = 1$ and $c = 6$.
Similarly $T(n) = \Omega(n^2)$, since for each $n \geqslant 1$ one has $2n^2 + 3n + 1 \geqslant n^2$. Thus $T(n)$ is in $\Theta(n^2)$.

# Quiz

The function $f(n) = n^2 \log n$ is

- $= O(n^3)$
- $= O(n)$
- $= \Omega(n)$
- $= \Omega(n^2)$
- $= O(n^{2+\varepsilon})$ for each $\varepsilon > 0$

$$\lim_{n \to \infty} \frac{\log n}{n^{\varepsilon}} = 0$$

$\varepsilon > 0$ !

# Running time of algorithms

We measure the running time of algorithms in terms of the *length of the input*.

Example: The inner product of two $n$-dimensional vectors can be computed in time $O(n)$ *(linear time)*.

# Running time of algorithms

We measure the running time of algorithms in terms of the *length of the input*.

Example: The inner product of two $n$-dimensional vectors can be computed in time $O(n)$ *(linear time)*.

Example: Two $n \times n$ matrices can be multiplied in time $O(n^3)$. The input has length $\Theta(n^2)$.
Measured by the length of the input, the algorithm runs in time $O(m^{1.5})$. $(m = n^2)$

# Running time of algorithms

We measure the running time of algorithms in terms of the *length of the input*.

Example: The inner product of two $n$-dimensional vectors can be computed in time $O(n)$ *(linear time)*.

Example: Two $n \times n$ matrices can be multiplied in time $O(n^3)$. The input has length $\Theta(n^2)$.
Measured by the length of the input, the algorithm runs in time $O(m^{1.5})$. $(m = n^2)$

An algorithm runs in *polynomial time*, if it carries out $O(n^k)$ elementary operations for some $k \in \mathbb{N}$, where $n$ is the *length* of the input.

fixed constant

# Quiz

```
def listexp(L):
    s = 2
    for i in L:
        s = s**2
    return s
```

$len(L) = n$

$s = s^2$

$$\left(\cdots\left(\left(2^2\right)^2\right)^2 \cdots\cdots\right)^2$$

$n$ iterations

Suppose that $L$ has $n$ elements.

- The algorithm carries out $O(n)$ elementary operations.
- The algorithm carries out $\Omega(2^n)$ elementary operations.
- After the last iteration of the loop, $s = 2^{n+1}$.
- After the last iteration of the loop, $s = 2^{2^n}$.

$S = 2^{2^n}$

$\log(S) = 2^n \leftarrow$ exponential in $len(L)$

# Polynomial time: Re-definition

An algorithm runs in *polynomial time*, if it carries out $O(n^k)$ elementary operations *on rational numbers of size* $O(n^k)$ for some $k \in \mathbb{N}$, where $n$ is the length of the input.

Fixed

- $x \in \mathbb{Z}$: $\text{size}(x) = \lceil \log_2(|x| + 1) \rceil$
- $x \in \mathbb{Q}$: $\text{size}(x) = \text{size}(p) + \text{size}(q)$, where $x = p/q$ with $p, q \in \mathbb{Z}$, $q \geqslant 1$ and $\gcd(p, q) = 1$

$1\ 1\ 0\ 1 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

$= 13$

$\Theta$(number of bits needed to rep. $x$)

$\boxed{\text{Also account for binary encoding length of numbers}}$