How long does it take?

```
SELECTIONSORT(A[1 .. n]):
  for i ← 1 to n
      for j ← i + 1 to n
          if A[j] < A[i]
              swap A[i] ↔ A[j]
```

- What computer? What language? What compiler? What OS?
- How long does it take to compare $A[i]$ and $A[j]$? To swap $A[i]$ and $A[j]$? To maintain i and j?
- How many times do we swap?
- What numbers are in the array $A[1\,..\,n]$?
- What is n?

How long does it take?

```
SELECTIONSORT(A[1 .. n]):
  for i ← 1 to n
      for j ← i + 1 to n
          if A[j] < A[i]
              swap A[i] ↔ A[j]
```

Let's count the number of comparisons ***as a function of n***:

- 1 iteration of the inner loop: 1 comparison
- ith iteration of the outer loop: $\displaystyle\sum_{j=i+1}^{n} 1 = n - i$ comparisons
- All iterations of the outer loop:

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n} (n - i) = \boxed{\frac{n(n-1)}{2} \text{ comparisons}}$$

- This doesn't depend on the input values in $A[1 \,..\, n]$.

Is that better or worse than n^2? $\dfrac{n^2}{100} + 5n$? $1000n$? $n^{3/2}$?

What does this tell us about the actual running time of the algorithm?

For any functions $f\colon \mathbb{N} \to \mathbb{R}$ and $g\colon \mathbb{N} \to \mathbb{R}$, $\boxed{\boldsymbol{f(n) = O(g(n))}}$ means

$$\exists c > 0\colon \exists N > 0\colon \forall n \geq N\colon\ f(n) \leq c \cdot g(n)$$

"If n is big enough, then $f(n)$ is at most a constant times $g(n)$."

- $n = O(n)$ .... $\forall n \geq \mathbf{0}\colon n \leq \mathbf{1} \cdot n$
  [$N = 0$ and $c = 1$]
- $5n = O(n)$ .... $\forall n \geq 0\colon 5n \leq 5 \cdot n$
  [$N = 0$ and $c = 5$]
- $\dfrac{n}{2} + 17 = O(n)$ .... $\forall n \geq 100\colon \dfrac{n}{2} + 17 \leq 100 \cdot n$
  [$N = 100$ and $c = 100$]
- $5n + 3 = O(n^2)$ .... $\forall n \geq 6\colon 5n + 3 \leq 25 \cdot n^2$
  [$N = 6$ and $c = 25$]
- $n^2 \neq O(5n + 3)$ .... $n^2 \leq c(5n + 3) \implies n \leq 5c + \frac{3c}{n} \leq 8c$
  [so ***no*** constant c works for *arbitrarily large* n]
- $\dfrac{n(n-1)}{2} = O(n^2)$ .... $\forall n \geq 0\colon \dfrac{n(n-1)}{2} \leq 1 \cdot n^2$
  [$N = 0$ and $c = 1$]

For *most* functions we will encounter:

$$\boxed{f(n) = O(g(n)) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty}$$

Theorem: $5n + 3 = O(n^2)$

Proof: $5n + 3 \le 8n \le 8n^2$ for all $n \ge 1$. [So take $c = 8$ and $N = 1$.] $\square$

Proof: Let's try $c = 2$.

$$\begin{aligned} 5n + 3 &\le 2 \cdot n^2 \\ \iff\ 2n^2 - 5n - 3 &\ge 0 \\ \iff\ (2n+1)(n-3) &\ge 0 \end{aligned}$$

This inequality is satisfied for any integer $n \ge 3$. $\square$

Proof: $\lim\limits_{n\to\infty} \dfrac{5n+3}{n^2} = \lim\limits_{n\to\infty} \left(\dfrac{5}{n} + \dfrac{3}{n^2}\right) = \lim\limits_{n\to\infty} \dfrac{5}{n} + \lim\limits_{n\to\infty} \dfrac{3}{n^2} = 0 + 0 = 0.$ $\square$

Big-Oh notation and its relatives show only how quickly functions ***grow*** as n gets bigger.

Suppose we have a machine that can execute 1,000,000,000 operations per second.

n	10	20	30	40	50	60	100
$\log n$	3.3ns	4.4ns	5ns	5.3ns	5.6ns	5.9ns	6.6ns
n	10ns	20ns	30ns	40ns	50ns	60ns	100ns
n^2	100ns	400ns	900ns	1.6μs	2.5μs	3.6μs	10μs
n^5	100μs	3.2ms	24.3ms	102.4ms	312.5ms	777.6ms	10s
2^n	1.02μs	1.04ms	1.07s	18.3min	13 days	36.5 yrs	40.2 Tyrs
3^n	59μs	3.48s	2.38 days	385 yrs	22 Myrs	1.34 Tyrs	$\cdots$
$n!$	3.6ms	77 yrs	8.4 Pyrs	$\cdots$			

1 ms = 10^{-3} second
1 μs = 10^{-6} second
1 ns = 10^{-9} second

1 Kyr = one thousand years
1 Myr = one million years
1 Gyr = one billion years
age of the universe $\approx$ 15 billion years
1 Tyr = 1 trillion years
1 Pyr = 1 quadrillion years

- $\boxed{\boldsymbol{f(n) = O(g(n))}}$ ('big Oh') means

$$\exists c > 0 \colon \exists N > 0 \colon \forall n \geq N \colon f(n) \leq c \cdot g(n)$$

- $\boxed{\boldsymbol{f(n) = \Omega(g(n))}}$ ('big Omega') means

$$\exists c > 0 \colon \exists N > 0 \colon \forall n \geq N \colon f(n) \geq c \cdot g(n)$$

- $\boxed{\boldsymbol{f(n) = \Theta(g(n))}}$ ('Theta') means

$$f(n) = O(g(n)) \quad \text{and} \quad f(n) = \Omega(g(n))$$

- $\boxed{\boldsymbol{f(n) = o(g(n))}}$ ('little oh') means

$$\forall c > 0 \colon \exists N > 0 \colon \forall n \geq N \colon f(n) < c \cdot g(n)$$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

- $\boxed{\boldsymbol{f(n) = \omega(g(n))}}$ ('little omega') means

$$\forall c > 0 \colon \exists N > 0 \colon \forall n \geq N \colon f(n) > c \cdot g(n)$$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

For all practical purposes:

- $\boxed{\boldsymbol{f(n) = O(g(n))}}$ ('big Oh') means

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

- $\boxed{\boldsymbol{f(n) = \Omega(g(n))}}$ ('big Omega') means

$$g(n) = O(f(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$$

- $\boxed{\boldsymbol{f(n) = \Theta(g(n))}}$ ('Theta') means

$$f(n) = O(g(n)) \quad \text{and} \quad f(n) = \Omega(g(n))$$

$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

- $\boxed{\boldsymbol{f(n) = o(g(n))}}$ ('little oh') means

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

$$f(n) = O(g(n)) \quad \text{but} \quad f(n) \neq \Omega(g(n))$$

- $\boxed{\boldsymbol{f(n) = \omega(g(n))}}$ ('little omega') means

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

$$g(n) = o(f(n))$$

$$f(n) = \Omega(g(n)) \quad \text{but} \quad f(n) \neq O(g(n))$$

Some identities for you to prove

- If $f(n) = O(g(n))$ and $F(n) = O(G(n))$ then
  - $f(n) + F(n) = O(g(n) + G(n))$,
  - $f(n) + F(n) = O(\max\{g(n), G(n)\})$,
  - $f(n) \cdot F(n) = O(g(n) \cdot G(n))$.

- If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$.

Ignoring constant factors:

$o(\,)$ **is like** $<$

$O(\,)$ **is like** $\leq$

$\Theta(\,)$ **is like** $=$

$\Omega(\,)$ **is like** $\geq$

$\omega(\,)$ **is like** $>$

$$f(n) = \Theta(g(n)) \iff g(n) = \Theta(f(n))$$
$$f(n) = \Omega(g(n)) \iff g(n) = O(f(n))$$
$$f(n) = \omega(g(n)) \iff g(n) = o(f(n))$$

$$f(n) = O(g(n)) \iff f(n) = o(g(n)) \text{ or } f(n) = \Theta(g(n))$$
$$f(n) = \Omega(g(n)) \iff f(n) = \omega(g(n)) \text{ or } f(n) = \Theta(g(n))$$

"At least $O(n)$" means
ABSOLUTELY NOTHING!

Some rules of thumb (for you to prove)

- For polynomials, only the largest term matters:

$$\sum_{i=1}^{k} a_i n^i = a_k n^k + a_{k-1} n^{k-1} + \cdots a_2 n^2 + a_1 n + a_0 = \Theta(n^k)$$

- For geometric series, only the largest term matters:

$$\sum_{i=1}^{n} c^i = \begin{cases} \dfrac{c^{n+1}-1}{c-1} = \begin{cases} \Theta(1) & \text{if } c < 1 \\ \Theta(c^n) & \text{if } c > 1 \end{cases} \\ n \qquad\qquad = \Theta(n) \qquad \text{if } c = 1 \end{cases}$$

- $\log n = o(n)$

Proof:

$$\begin{aligned} \lim_{n\to\infty} \frac{\ln n}{n} &= \lim_{n\to\infty} \frac{1/n}{1} && \text{[l'Hôpital's rule]} \\ &= \lim_{n\to\infty} \frac{1}{n} = 0 \end{aligned}$$

□

- Some common functions in increasing order:

$$1 \quad \log n \quad \log^2 n \quad \sqrt{n} \quad n \quad n \log n \quad n^2 \quad n^3 \quad n^{100} \quad 2^n \quad 3^n \quad n! \quad n^n$$

How long does it take?

$$\boxed{\begin{array}{l}\underline{\textsc{SelectionSort}(A[1\,..\,n]):}\\ \quad \text{for } i \leftarrow 2 \text{ to } n\\ \qquad\quad \text{for } j \leftarrow i+1 \text{ to } n\\ \qquad\qquad\quad \text{if } A[j] < A[i]\\ \qquad\qquad\qquad\quad \text{swap } A[i] \leftrightarrow A[j]\end{array}}$$

- The last two lines run in $\Theta(1)$ time.
- Thus, the total running time is $\displaystyle\sum_{i=2}^{n}\sum_{j=i+1}^{n}\Theta(1)$.

  Ignoring constant factors:

$$\sum_{i=2}^{n}\sum_{j=i+1}^{n} 1 = \sum_{i=2}^{n}(n-i) \begin{cases} \displaystyle\leq \sum_{i=1}^{n} n = n^2 = O(n^2) \\ \displaystyle\geq \sum_{i=n/2+1}^{n} \frac{n}{2} = \frac{n^2}{4} = \Omega(n^2) \end{cases}$$

  So the algorithm runs in $\boxed{\Theta(n^2)\text{ time}}$.

- This analysis is ***independent*** of language, compiler, architecture, clock speed, operating system, and even number of swaps.