SORTING

Input : a set of numbers $a_1, a_2, a_3, \ldots, a_n$
Output : a list of $a_i$ in sorted order (a permutation)

We want an algorithm that can handle any instance

1st objective : get it right, always
2nd objective : get it done quickly ... and don't use lots of resources

Correctness, time efficiency, space/storage efficiency

Could also ask for a clear/understandable algo, or easy to modify, etc.
... but what are the rules?

- are the numbers integers or reals? rational? positive? distinct?
- is their size bounded?
- can we add them or just compare them?
- how are they presented to us?

We'll discuss this soon, so let's focus on:

- only comparing elements: in time \( t \to \text{some constant} \)
- input in an array
Sorting: start with a simple algo that we can prove is correct

**Insertion sort**

- Assume that the prefix of your list (array) is sorted
- Increase the size of this sorted subset
- Repeat
  - start with a trivial prefix: size=1

**Before**

\[ \leq \leq \leq \leq \leq \text{ ? ? ? ? ? ? ? ? } \]

sorted

**After**

\[ \leq \leq \leq \leq \leq \leq \text{ ? ? ? ? ? ? ? ? } \]

sorted

Use the element next to the prefix, to increase the prefix size
If the prefix has size $j$ then we can insert "?" after at most $j$ comparisons.
With $\leq j$ comparisons we can increase the size of our sorted prefix from $j$ to $j+1$.

We want a prefix = the whole set = size $n$

$$\begin{align*}
\text{comparisons} &= \sum_{j=1}^{n} j \\
&= 1 + 2 + 3 + \ldots + (n-2) + (n-1) + n \\
&= \frac{n(n+1)}{2} \\
&= \frac{1}{2}n^2 + \frac{1}{2}n
\end{align*}$$

To actually implement this, you need some extra time & space to allow swapping but it's just a constant. i.e maybe time $= 5 \cdot (\frac{1}{2}n^2 + \frac{1}{2}n)$.