Simply stated: “Given a list of $n$ unique values, find the $i^{th}$ smallest.”

Common Examples
- $1^{st}$ smallest (Minimum)
- $n^{th}$ smallest (Maximum)
- $\lceil n/2 \rceil^{th}$ smallest (Median)

How can we approach solving such problems?

Trivial Way: Sort the list and then return the $i^{th}$ position.

This is clearly not a good approach for things such as minimum and maximum.

This may or may not be a good approach for other problems such as median finding.
**Minimum**

- In the worst case, finding the minimum requires \( n-1 \) comparisons.
- Finding the minimum can easily be done using at worst \( n-1 \) comparisons:
  - Call the first item in the list the smallest.
  - For each item remaining, compare it to the item currently considered smallest and if it is smaller than that item, set this new item as the smallest.
- Do other algorithms exist? Sure, but are they better? What would the runtime be of the following recursive algorithm?
  - Split the list in half.
  - Find the minimum of each half.
  - Take the minimum of the two “local” minima returned.

Assume \( n \) is a power of 2.

\[
T(1) = 0 \\
T(n) = 2T\left(\frac{n}{2}\right) + 1 \\
T(n) = n-1
\]

**Base:** \( n=1 \)

\( T(1) = 0 \)

\( T(2) = 1 \)

\( T(3) = 2 \)

\( T(4) = 3 \)

\( T(5) = 4 \)

\( T(6) = 5 \)

\( T(7) = 6 \)

\( T(8) = 7 \)

\( T(9) = 8 \)

\( T(10) = 9 \)

\( T(11) = 10 \)

\( T(12) = 11 \)

\( T(13) = 12 \)

\( T(14) = 13 \)

\( T(15) = 14 \)

\( T(16) = 15 \)

**I H:** \( \forall i < k \quad T(i) = i-1 \)

**I S:** \( T(k) = k-1 \)

\( 2T\left(\frac{k}{2}\right) + 1 \leq k-1 \)

\( 2\left(\frac{k}{2} - 1\right) + 1 \leq k-1 \)

\( k-2 + 1 \leq k-1 \)

\( k-1 \leq k-1 \quad \checkmark \)
Maximum

Is there any practical difference between algorithms for finding the maximum as opposed to finding the minimum value in a list?
Consider the following scenario:

You are given a list of coordinates and are asking to return a bounding box for these points.

Your `getBoundingBox()` method would need to find both the minimum x-coordinate as well as the maximum x-coordinate (and then do the same for the y-coordinates).

In general, given a list of items, it is easy to find the minimum and the maximum using $2(n-1)$ comparisons.

Can we do better?
Min/Max Algorithm #1

What is the runtime of the following algorithm to find the minimum and maximum “at the same time” and will it always give the correct results?

- Traverse the list once, two at a time, comparing pairs.
- As this is done, create two sub-lists: SubList1 for the greater of the pair-wise comparisons and SubList2 for the lesser.
- Call the regular maximum algorithm on SubList1 and the regular minimum algorithm on SubList2.

\[
\begin{align*}
\text{Stage 1} & : \quad \frac{n}{2} + (\frac{n}{2}) - 1 \\
\text{Stage 2} & : \quad 3 \left( \frac{n}{2} \right) - 2 = \frac{3n}{2} - 2 \\
\text{Total} & : \quad 2(n-1) = 2n - 2
\end{align*}
\]
Min/Max Algorithm #2

What is the runtime of the following algorithm to find the minimum and maximum “at the same time” and will it always give the correct results?

- Compare the first two elements in the list. Set the smaller as min and the larger as max. \( \text{Phase 1} \)
- For the remaining elements of the list:
  - Pair up and compare the items in each pair. \( \text{Phase 2} \)
  - Compare the smaller of the pair to the current min, replacing it if we have a new min.
  - Compare the larger of the pair to the current max, replacing it if we have a new max.

\[
\begin{array}{c|cccccc}
1 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & \ldots \ \\
\hline
\text{min: } x_1 & \text{max: } x_2
\end{array}
\]

\[
1 + \frac{n-2}{2} + \frac{n-2}{2} = \frac{3n}{2} - 2
\]