Recall the Quick Select algorithm. We will prove it is correct. To do so, we must first prove Partition is correct. To do that, we will need what we call a **loop invariant** to prove that its loop terminates with the state we need.

A loop invariant is some statement which, if it's true at the beginning of an iteration must also be true at the end of an iteration.

**Ex.**

\[ \text{sum} = 0 \]

For \(i = 1\) to \(n\)

\[ \text{sum} + i \]

We define \(i\) to be 0 outside the loop and to increment at the beginning of the loop. It often helps to write pseudocode explicitly state what's going on.

A LI here would be \(\text{sum} = \frac{i(i+1)}{2}\).

To prove a LI is correct, we prove 3 things.

**Initialization** (The LI must be true before the loop)

\[ \text{sum} = 0 = \frac{0(0+1)}{2} = \frac{0(0+1)}{2} \]

**Maintenance** (LI \(\Rightarrow\) LI')

\[ i' = i+1 \]

\[ \text{sum}' = \text{sum} + i' \]

So \(\text{sum}' = \text{sum} + i' = \frac{i(i+1)}{2} + i' = \frac{(i'-1)i'}{2} + \frac{2i'}{2} \]

**Termination** (Prove the loop ends and that the algorithm is correct)

Since \(i\) is increasing by 1 each iteration, the loop ends when \(i = n\). Then \(\text{sum} = \frac{n(n+1)}{2}\), which is presumably what we wanted.
We can now prove Partition is correct.

First, let’s write down what Partition is supposed to do.

For an array $A$ with $0 \leq l \leq r < |A|$, Partition($A, l, r, p$):

- Returns an index $l \leq i \leq r$ such that $A'[i] = A[p]$.
- Leave $A'[i] = A[i]$ for $0 \leq i \leq l$ and $r < i \leq |A|$
- Satisfies $A'[i] \leq A'[I]$ for $l \leq i \leq I$
- Satisfies $A'[i] > A'[I]$ for $I < i \leq r$
- Satisfies for $l \leq i \leq r$ there is $l \leq j \leq r$ such that $A'[j] = A[j]$. (Assume each entry unique for convenience)

Less formally, Partition rearranges $A$ from $l$ to $r$ such that everything to the left of wherever $A[p]$ ends up is smaller and everything to the right is bigger. It then returns the new index of $A[p]$.

We’ll now need 3 LIs. Often it’s easiest to write them down from outermost to innermost.

\[\text{LIO}(A, l, r, p, l, r, p) : \forall i, l \leq i \leq l, A[i] \leq p\]
\[\forall i, r < i < r, A[i] > p\]
\[A[r] = p\]

\[\text{LII1}(A, l, r, p, l, r, p) : \text{LIO}(A, l, r, p, l, r, p)\]
\[\text{LII2}(A, l, r, p, l, r, p) : \text{LIO}(A, l, r, p, l, r, p)\]

\[\text{LIO Init}\]
This is vacuously true except for the third point, which is explicitly true.

\[\text{LIO Maintainence}\]
Assume $\text{LIO}(A, l, r, p, l, r, p)$.

\[\text{LII1 Init}\]
True by assumption.

\[\text{LII1 Maintainence}\]
Assume $\text{LII1}(A, l, r, p, l, r, p)$.
If the loop terminates, we’re done.
Otherwise, $l < r$ and $A[l] \leq p$.
\[l' = l+1\]
Since $A[l] \leq p$ and nothing else has changed, $\text{LII1}(A, l', r, p, l, r, p)$. 
LII1 Termination

Since $l$ is incremented by 1 each iteration and $R \leq \infty$, the loop must terminate eventually. When it does, either $l' = R$ or $A[l'] > P$.

LII2 Init

True by assumption because LII1$(A, l', r, l, l, R, P)$ is true.

LII2 Maintenance

Assume LII2$(A, l', r, p, l, l, R, P)$.
If the loop terminates, we’re done.
Otherwise $r > l$ and $A[r] > P$.

$r' = r - 1$


LII2 Termination

Since $r$ is decremented by 1 each iteration and $l > -\infty$, the loop must terminate eventually. When it does, either $r' = l$ or $A[r'] \leq P$.

Lastly swap $A[l']$ and $A[r']$ if $l' < l$, which has no bearing on LIO$(A, l', r', p, l, l, R, P)$. However, we note that in this case $r' > l$ and $l' \leq R$, which results in $A[l'] \leq P$ and $A[r'] > P$.
Otherwise since LII2$(A, l', r', p, l, l, R, P)$, LIO$(A, l', r', p, l, l, R, P)$ as $A'$ has otherwise not changed.
When instead $l' \geq r'$, we don’t change $A$ at all, so LII2$(A, l', r', p, l, l, R, P)$ gives us LIO$(A, l', r', p, l, l, R, P)$.

LIO Termination

Unless $l' \geq r'$, in which case the loop terminates, each iteration necessarily swaps two elements of $A$ so that $A'[l'] \leq P$ and $A'[r'] > P$. This will cause $r' - l'$ to decrease by at least 2 each iteration. This difference must go to 0, at which point $l' \geq r'$ and the loop terminates.
When the loop terminates, we have \( L(0, l', r', p, l, r, p) \).

Since \( l' \geq r' \) we know that when we swap \( A'[l'] \) and \( A'[R] \),
\( A''[R] > p \). Therefore, we have

\[ \begin{align*}
A''[i] &= A[i] & \text{for } 0 \leq i \leq l \text{ and } l \leq i \leq |A| \\
A''[i] &= A'[i'] & \text{for } l < i \leq l' \\
A''[i] &= A''[l] & \text{for } l < i \leq r \\
\text{For each } i, l \leq i \leq r, \text{ there is a } l < j < r \text{ such that } A''[i] = A[j] \\
\text{Lastly, Partition returns } l', \text{ and } A''[l'] = A[p].
\end{align*} \]

We now prove \text{QuickSelect} correct through induction.

\text{QuickSelect} \((A, l, r, K)\) should:

- For an array \( A \) with \( 0 \leq l \leq K \leq r \leq |A| \)
- and the \( l^{th} \) to \( r^{th} \) smallest elements of \( A \) within the range \( l \) to \( r \);
  - Return the \( k^{th} \) smallest element of \( A \).

\begin{enumerate}
\item \text{When } l = r, \text{ it must be the case that } l = k \text{ and }
\item \text{BC: } A[l] \text{ contains the } k^{th} \text{ smallest element of } A. \text{ As we return this QuickSelect is correct.}
\item \text{IS: } \text{Now assume that QuickSelect} \((A, l, r, K)\) \text{ is correct for any } l \text{ and } r \text{ subject to } r - l \leq n.
\end{enumerate}

\text{Let } l \text{ and } r \text{ be such that } r - l = n + 1.

We know \( l \neq r \) since \( r - l > 0 \).

\text{Let } p \text{ be whatever pivot index we choose between } l \text{ and } r.

\text{Partition} \((A, l, r, p)\) \text{ sets } A \text{ to } A' \text{ and } p \to p' \text{ such that everything in } A'[l..r] \text{ smaller than } A[p] \text{ is to its left in the subarray. Similarly, everything larger than } A[p] \text{ in } A'[l..r] \text{ is to its right in the subarray.
Since \( A[l..r] \) contains the \( q \)th to \( r \)th smallest elements of \( A \), it then follows that \( A'[p'] \) is the \( p' \)th smallest element of \( A' \).

If \( k=p' \), we return \( A'[p'] \) correctly.

If \( k \neq p' \), we return \( \text{QuickSelect}(A', l', p'-1, k) \).

As \( k \leq p' \), we have \( 0 \leq l \leq k \leq p'-1 \leq l' \). Moreover, 
\[ A'[l..p'-1] \] must contain the \( k \)th to \( (p'-1) \)th smallest elements of \( A' \) since partition put the elements of \( A[l..r] \) smaller than \( A[p] \) into \( A'[l..p'-1] \) left of \( A'[p'] \).

Lastly, since \( (p'-1) - l \leq r - l = n+1 \), it follows that 
\( \text{QuickSelect}(A', l', p'-1, k) \) returns the \( k \)th smallest element of \( A' \), which is also the \( k \)th smallest element of \( A \) and we return correctly immediately.

Finally, in the \( k > p' \) case, a similar train of logic follows.

\[ \square \]