Given a list \( L \) such that \( L(\square L) = \square L \), write a program to sort \( L \) in linear time and constant space. The only change permitted to \( L \) is to swap two items.

After trying the question, scroll down to the solution.
The problem is $P$, defined as

$$P \equiv L(\Box L) = \Box L \Rightarrow L' = [0;\ldots;\#L]$$

The only change permitted to $L$ is swap, defined as

$$\text{swap } i j = L := i \rightarrow L j \ j \rightarrow L i \ \mid L$$

Execution time has to be linear, so that suggests starting an index variable $k$ at 0, and moving up by $k := k+1$ until $k = \#L$, so that the part of the list before $k$ is in order, and therefore the part of the list from $k$ onward has the right items but maybe not yet in the right order.

$$P \iff k := 0. \ Q$$

$$Q \iff \begin{cases} k = \#L \text{ then } & \text{ok} \\ \text{else if } Lk = k & \text{then } k := k+1. \ Q \\ \text{else swap } (Lk)k. \ Q \ \text{fi} \end{cases}$$

To define $Q$, we can look at $P$ for inspiration. Perhaps

$$Q \equiv L(k,\ldots;\#L) = k,\ldots;\#L \Rightarrow L' = L[0;\ldots;k] ;; [k;\ldots;\#L]$$

I think that will work. But I think it will be easier to prove the $Q$ refinement if we weaken $Q$ by strengthening its antecedent. I'm going to try

$$Q \equiv L[0;\ldots;k] = [0;\ldots;k] \land L(k,\ldots;\#L) = k,\ldots;\#L \Rightarrow L' = L[0;\ldots;k] ;; [k;\ldots;\#L]$$

This says: if the first part of $L$ is done, and the last part has the right items (but not necessarily in the right order), then we complete the job by leaving the first part of $L$ alone and putting the last part in order.

Proof of $P$ refinement:

$$k := 0. \ Q$$

$$= \quad \begin{array}{c} k := 0. \ L[0;\ldots;k] = [0;\ldots;k] \land L(k,\ldots;\#L) = k,\ldots;\#L \Rightarrow L' = L[0;\ldots;k] ;; [k;\ldots;\#L] \\ \text{Substitution Law} \\ \text{simplify} \end{array}$$

$$= \quad \begin{array}{c} P \end{array}$$

Proof of first case of $Q$ refinement:

$$k = \#L \land \text{ok} \Rightarrow Q$$

$$= \quad \begin{array}{c} k = \#L \land k' = k \land L' = L \\ \Rightarrow (L[0;\ldots;k] = [0;\ldots;k] \land L(k,\ldots;\#L) = k,\ldots;\#L \Rightarrow L' = L[0;\ldots;k] ;; [k;\ldots;\#L]) \quad \text{context} \\ \text{simplify} \end{array}$$

$$= \quad \begin{array}{c} \top \end{array}$$

Proof of middle case of $Q$ refinement:

$$k = \#L \land Lk = k \land (k := k+1. \ Q)$$

$$= \quad \begin{array}{c} k = \#L \land Lk = k \\ \land (L[0;\ldots;k+1] = [0;\ldots;k+1] \land L(k+1,\ldots;\#L) = k+1,\ldots;\#L \Rightarrow L' = L[0;\ldots;k+1] ;; [k+1;\ldots;\#L]) \quad \text{use context } Lk=k \text{ to simplify the implication} \\ \text{simplify} \end{array}$$

$$= \quad \begin{array}{c} k = \#L \land Lk = k \land Q \end{array}$$
Proof of last case of \( Q \) refinement:

\[
\begin{align*}
& k+\#L \land L \neq k \land (\text{swap} (Lk) k). \; Q) \Rightarrow Q \\
\Rightarrow & \; k+\#L \land L \neq k \land (\text{swap} (Lk) k). \; Q) \\
\Rightarrow & \; k+\#L \land L \neq k \land (\text{swap} (Lk) k). \; Q) \\
\Rightarrow & \; (L[0;..;k] = [0;..;k] \land L(k,..#L) = k,..#L) \; \text{portation} \\
\Rightarrow & \; k+\#L \land L \neq k \land (\text{swap} (Lk) k). \; Q) \\
\Rightarrow & \; L'[0;..;k] = [0;..;k] \land L(k,..#L) = k,..#L \\
\Rightarrow & \; L' = L[0;..;k]. \\
\end{align*}
\]

To prove this implication, I'll go from the antecedent on the top line to the consequent on the bottom line.

\[
\begin{align*}
& k+\#L \land L \neq k \land (\text{swap} (Lk) k). \; Q) \\
\land & \; (L[0;..;k] = [0;..;k] \land L(k,..#L) = k,..#L) \\
\Rightarrow & \; L'[0;..;k] = [0;..;k] \land L(k,..#L) = k,..#L \\
\Rightarrow & \; L' = L[0;..;k]. \\
\end{align*}
\]

Proof of last case of \( Q \) refinement:

\[
\begin{align*}
& k+\#L \land L \neq k \land (\text{swap} (Lk) k). \; Q) \land L[0;..;k] = [0;..;k] \land L(k,..#L) = k,..#L \\
\land & \; (L[0;..;\neq k] = [0;..;k] \land L(k,..#L) = k,..#L) \\
\Rightarrow & \; L'[0;..;k] = [0;..;k] \land L(k,..#L) = k,..#L \\
\Rightarrow & \; L' = L[0;..;k]. \\
\end{align*}
\]

With time, the refinements are

Recursive time is bounded by \( 2 \times \#L \). Counting just \textit{swaps}, the time is bounded by \( \#L \) .
\[ A \iff k := 0. \ B \]
\[ B \iff \begin{align*}
& \text{if } k = \#L \text{ then } \text{ok} \\
& \text{else if } Lk = k \text{ then } k := k + 1. \ t := t + 1. \ B \\
& \text{else } \text{swap } (Lk) \ k. \ t := t + 1. \ B \fi \fi \]

Proof of \textit{A} refinement:
\[
\begin{align*}
k := 0. & \iff B \\
k := 0. & \iff t' \leq t + \#L - k + f \ k \\
t' \leq t + \#L - 0 + f \ 0 & \iff A
\end{align*}
\]

Proof of first case of \textit{B} refinement:
\[
\begin{align*}
k &\iff \#L \land k = k \land (k := k + 1. \ B) \iff B \\
k &\iff \#L \land k \land L' = L \land t' = t \implies t' \leq t + \#L - k + f \ k \land \text{context} \\
k &\iff \#L \land k \land L' = L \land t' = t \implies t' \leq t + \#L - \#L + f \ (#L) \land \text{simplify and apply} \\
k &\iff \#L \land k = k \land L' = L \land t' = t \implies 0 \leq \#L_{j+1} \land \text{simplify} \\
k &\iff \#L \land k = k \land L' = L \land t' = t \implies 0 \leq 0 \land \text{simplify and base} \\
& \iff \top
\end{align*}
\]

Proof of middle case of \textit{B} refinement:
\[
\begin{align*}
k \iff \#L \land \#L = k \land (k := k + 1. \ B) & \iff B \\
k \iff \#L \land \#L = k \land (k := k + 1. \ t' \leq t + \#L - k + f \ k) \land \text{substitution law} \\
k \iff \#L \land \#L = k \land t' \leq t + 1 + \#L - k - 1 + f \#k+1 \land \text{simplify} \\
k \iff \#L \land \#L = k \land t' \leq t + \#L - k + f \ (k+1) \land \text{context } L \leq k \implies f \# \leq f \# \land \text{specialize} \\
k \iff \#L \land \#L = k \land t' \leq t + \#L - k + f \ k \\
& \iff \top
\end{align*}
\]

Proof of last case of \textit{B} refinement:
\[
\begin{align*}
k \iff \#L \land \#L = k \land (\text{swap } (Lk) \ k. \ t := t + 1. \ B) & \iff \text{swap and } B \\
k \iff \#L \land \#L = k \land \text{(L := Lk } \rightarrow \text{Lk } \mid k \rightarrow \text{L(Lk) } \mid L. \ t := t + 1. \ t' \leq t + \#L - k + f \ k) \\
& \text{The next step looks like it should be the Substitution Law.} \\
& \text{But } f \text{ is defined in terms of } L. \text{ So we have to apply } f \text{ first.} \\
& \iff k \iff \#L \land \#L = k \land \text{swap does not affect length} \\
& \iff k \iff \#L \land \#L = k \land t' \leq t + 1 + \#L - k + f \# \land t' \leq t + \#L - k + f \# \land \text{swap reduces the number of out-of-place items by } 1 \text{ or } 2 \\
& \iff k \iff \#L \land \#L = k \land t' \leq t + 1 + \#L - k + f \# \land (L := Lk } \rightarrow \text{Lk } \mid k \rightarrow \text{L(Lk) } \mid L) j \iff j \\
& \iff k \iff \#L \land \#L = k \land t' \leq t + \#L - k + f \ k \land \text{specialize} \\
& \iff \top
\end{align*}
\]

And that completes the last case of the \textit{B} refinement.