The problem is $P$, defined as

$$ P = L(\square L) = \square L \Rightarrow L' = [0;..#L] $$

The only change permitted to $L$ is $\text{swap}$, 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.

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

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

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

$$ Q \iff L(k;..#L) = k;..#L \Rightarrow L' = L[0;..k] ;; [k;..#L] $$

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

$$ Q \iff L[0;..k] = [0;..k] \land L(k;..#L) = k;..#L \Rightarrow L' = L[0;..k] ;; [k;..#L] $$

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:

$$ k := 0. \ Q $$

$$ = k := 0. \ L[0;..k] = [0;..k] \land L(k;..#L) = k;..#L \Rightarrow L' = L[0;..k] ;; [k;..#L] $$

Substitution Law

$$ = L[0;..0] = [0;..0] \land L(\square L) = \square L \Rightarrow L' = L[0;..0] ;; [0;..#L] $$

simplify

$$ = P $$

Proof of first case of $Q$ refinement:

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

replace $\text{ok}$ and $Q$

$$ = k = #L \land k' = k \land L' = L $$

context

$$ = (L[0;..k] = [0;..k] \land L(k;..#L) = k;..#L \Rightarrow L' = L[0;..k] ;; [k;..#L]) $$

simplify

$$ = \top $$

Proof of middle case of $Q$ refinement:

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

replace $Q$

$$ = k + #L \land L k = k \land (k := k+1. \ L[0;..k] = [0;..k] \land L(k;..#L) = k;..#L \Rightarrow L' = L[0;..k] ;; [k;..#L]) $$

substitution law

$$ = k + #L \land L k = k \land (L[0;..k+1] = [0;..k+1] \land L(k+1;..#L) = k+1;..#L \Rightarrow L' = L[0;..k+1] ;; [k+1;..#L]) $$

use context $L k = k$ to simplify the implication
\[ k \# L \land L \bar{k} = k \land (L[0..k] = [0..k] \land L(k,..\#L) = k,..\#L) \Rightarrow L' = L[0..k];(k;..\#L) \]
\[ k \# L \land Lk = k \land Q \]
\[ \Rightarrow Q \]

Proof of last case of \( Q \) refinement:
\[ k \# L \land Lk = k \land (\text{swap} (Lk) \land Q) \Rightarrow Q \]
\[ k \# L \land Lk = k \land (\text{swap} (Lk) \land Q) \]
\[ \Rightarrow (L[0..k] = [0..k] \land L(k,..\#L) = k,..\#L) \Rightarrow L' = L[0..k];(k;..\#L) \]
\[ k \# L \land Lk = k \land (\text{swap} (Lk) \land Q) \land L[0..k] = [0..k] \land L(k,..\#L) = k,..\#L \]
\[ \Rightarrow L' = L[0..k];(k;..\#L) \]

To prove this implication, I'll go from the antecedent on the top line to the consequent on the bottom line.
\[ k \# L \land Lk = k \land (\text{swap} (Lk) \land Q) \land L[0..k] = [0..k] \land L(k,..\#L) = k,..\#L \]
\[ \Rightarrow \]
\[ k \# L \land Lk = k \land (\text{swap} (Lk) \land Q) \land L[0..k] = [0..k] \land L(k,..\#L) = k,..\#L \]
\[ \land (L := Lk \rightarrow Lk | k \rightarrow L(Lk) | L) \]
\[ \Rightarrow \]
\[ L' = (Lk \rightarrow Lk | k \rightarrow L(Lk)) \Rightarrow L[0..k],..\#(Lk \rightarrow Lk | k \rightarrow L(Lk) | L) \]

This next step is more complicated and less formal than I would like. In the top line it says \( L[0..k] = [0..k] \), and since each item in the list occurs once, the items less than \( k \) are used up at indexes less than \( k \). The top line also says \( Lk \neq k \), therefore \( Lk > k \). So the swap is swapping the item at \( k \) with an item at an index greater than \( k \). The swap does not affect the first part of the list \( L[0..k] \). The swap affects the last part of the list, but it does not change the bunch of items in the last part of the list \( L(0..k) \). So the top line, used as context, allows us to simplify the bottom three lines.

\[ k \# L \land Lk = k \land L[0..k] = [0..k] \land L(k,..\#L) = k,..\#L \land \]
\[ L[0..k] = [0..k] \land L(k,..\#L) = k,..\#L \land \]
\[ \Rightarrow L' = L[0..k];(k;..\#L) \]
\[ \Rightarrow L' = L[0..k];(k;..\#L) \]

And that completes the last case of the \( Q \) refinement.

Recursive time is bounded by \( 2 \times \# L \). Counting just \textit{swaps}, the time is bounded by \# \( L \).

To prove time bounds, it is helpful to define
\[ f(i) = \phi[\beta: i..\#L \land j \neq j] \]

Then the timing specifications are \( A \) and \( B \), defined as
\[ A \equiv i' \leq t + \# L + f0 \]
\[ B \equiv i' \leq t + \# L - k + f k \]
With time, the refinements are
\[ A \iff k := 0. \ B \]
\[ B \iff \text{if } k = \#L \text{ then } \text{ok} \]
\[ \text{else if } Lk = k \text{ then } k := k + 1. \ t := t + 1. \ B \]
\[ \text{else } \text{swap } (Lk) \ k. \ t := t + 1. \ B \fi \fi \]

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

Proof of first case of \( B \) refinement:
\[
\begin{align*}
k = \#L \land \text{ok} & \implies B \quad \text{replace } \text{ok} \text{ and } B \\
\iff & k = \#L \land k' = k \land L' = L \land t' = t \implies t' \leq t + \#L - k + f k \\
\iff & k = \#L \land k' = k \land L' = L \land t' = t \implies t \leq t + \#L - \#L + f (\#L) \\
\iff & k = \#L \land k' = k \land L' = L \land t' = t \implies 0 \leq \& j; \#L; \#L \land j = j \\
\iff & k = \#L \land k' = k \land L' = L \land t' = t \implies 0 \leq 0 \\
\iff & \top
\end{align*}
\]

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

Proof of last case of \( B \) refinement:
\[
\begin{align*}
k = \#L \land Lk = k \land (\text{swap } (Lk) k. \ t := t + 1. \ B) & \quad \text{replace } \text{swap} \text{ and } B \\
\iff & k = \#L \land Lk = k \land (L := Lk \rightarrow Lk \mid k \rightarrow L(Lk) \mid L. \ t := t + 1. \ t' \leq t + \#L - k + f k) \\
\text{The next step looks like it should be the Substitution Law.} \\
\text{But } f \text{ is defined in terms of } L. \text{ So we have to apply } f \text{ first.} \\
\iff & k = \#L \land Lk = k \\
\iff & \left( L := Lk \rightarrow Lk \mid k \rightarrow L(Lk) \mid L. \ t := t + 1. \ t' \leq t + \#L - k + \& j; k, \ldots; L \land j \neq j \right) \\
\text{Now use the Substitution Law} \\
\iff & k = \#L \land Lk = k \\
\iff & \left( t' \leq t + 1 + \#L - k + \& j; k, \ldots; L \land j \neq j \right) \\
\iff & \left( L := Lk \rightarrow Lk \mid k \rightarrow L(Lk) \mid L. \ t := t + 1 + \#L - k + \& j; k, \ldots; L \land j \neq j \right) \\
\text{swap does not affect length} \\
\iff & k = \#L \land Lk = k \land t' \leq t + 1 + \#L - k + \& j; k, \ldots; L \land j \neq j - 1 \\
\iff & k = \#L \land Lk = k \land t' \leq t + \#L - k + f k \\
\iff & B
\end{align*}
\]

And that completes the last case of the \( B \) refinement.