(n sort) Given a list \( L \) such that \( L(0..\#L) = 0..\#L \), write a program to sort \( L \) in linear time and constant space. The only change permitted to \( L \) is to swap two items.

\[\begin{align*}
\text{The problem is } & P, \text{ defined as } \\
& P = L(0,..\#L) = 0,..\#L \Rightarrow L' = [0,..\#L]
\end{align*}\]

The only change permitted to \( L \) is swap, defined as

\[\text{swap } i j = L := i \rightarrow Lj \mid j \rightarrow Li \mid L\]

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.

\[\begin{align*}
P & \iff k:=0. \ Q \\
Q & \iff \text{if } k=\#L \text{ then } \text{ok} \text{ else if } Lk=k \text{ then } k:=k+1. \ Q \text{ else swap } (Lk)k. \ Q \text{ fi fi}
\end{align*}\]

To define \( Q \), we can look at \( P \) for inspiration. Perhaps

\[Q = \text{L}(k,..\#L) = k,..\#L \Rightarrow L' = L[0,..k] + [k,..\#L]\]

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

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

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.

\[\begin{align*}
\text{Proof of } P \text{ refinement:} & \\
& \equiv k:=0. \ Q & \text{replace } Q \\
& \equiv k:=0. L[0,..k] = [0,..k] \land L(k,..\#L) = k,..\#L \Rightarrow L' = L[0,..k] + [k,..\#L] & \text{Substitution Law} \\
& \equiv L[0,..0] = [0,..0] \land L(0,..\#L) = 0,..\#L \Rightarrow L' = L[0,..0] + [0,..\#L] & \text{simplify} \\
& \equiv P &
\end{align*}\]

\[\begin{align*}
\text{Proof of first case of } Q \text{ refinement:} & \\
& \equiv k=\#L \land \text{ok} \Rightarrow Q & \text{replace } \text{ok} \text{ and } Q \\
& \equiv k=\#L \land k'=k \land L'=L & \\
& \Rightarrow (L[0,..k] = [0,..k] \land L(k,..\#L) = k,..\#L \Rightarrow L' = L[0,..k] + [k,..\#L]) & \text{context} \\
& \equiv k=\#L \land k'=k \land L'=L & \\
& \Rightarrow (L[0,..\#L] = [0,..\#L] \land L(#L,..\#L) = #L,..\#L \Rightarrow L = L[0,..\#L] + [#L,..\#L]) & \text{simplify} \\
& \equiv \top &
\end{align*}\]

\[\begin{align*}
\text{Proof of middle case of } Q \text{ refinement:} & \\
& \equiv k+\#L \land Lk=k \land (k:=k+1. \ Q) & \text{replace } Q \\
& \equiv k+\#L \land Lk=k & \\
& \land (k:=k+1. \ L[0,..k] = [0,..k] \land L(k,..\#L) = k,..\#L \Rightarrow L' = L[0,..k] + [k,..\#L]) & \text{substitution law} \\
& \equiv k+\#L \land Lk=k & \\
& \land (L[0,..k+1] = [0,..k+1] \land L(k+1,..\#L) = k+1,..\#L \Rightarrow L' = L[0,..k+1] + [k+1,..\#L]) & \text{use context } Lk=k \text{ to simplify the implication}
\end{align*}\]
\[\begin{align*}
&\equiv \, k\#L \land Lk=k \land (L[0;..k] = [0;..k] \land L(k,..\#L) = k,..\#L) \implies L' = L[0;..k] + [k;..\#L]) \\
&\equiv \, k\#L \land Lk=k \land Q \\
&\implies Q \\
&\equiv \, k\#L \land Lk=k \land Q \\
&\implies L' = L[0;..k] + [k;..\#L] \\
&\implies \text{Proof of last case of } Q \text{ refinement:} \\
&\quad k\#L \land Lk=k \land (\text{swap } (Lk) k. \quad Q) \implies Q \\
&\quad \Rightarrow (L[0;..k] = [0;..k] \land L(k,..\#L) = k,..\#L) \implies L' = L[0;..k] + [k;..\#L]) \quad \text{portation} \\
&\quad \equiv k\#L \land Lk=k \land (\text{swap } (Lk) k. \quad Q) \land L[0;..k] = [0;..k] \land L(k,..\#L) = k,..\#L \\
&\quad \Rightarrow L' = L[0;..k] + [k;..\#L] \\
&\quad \text{To prove this implication, I'll go from the antecedent on the top line to the consequent on the bottom line.} \\
&\quad k\#L \land Lk=k \land (\text{swap } (Lk) k. \quad Q) \land L[0;..k] = [0;..k] \land L(k,..\#L) = k,..\#L \\
&\quad \Rightarrow \quad \text{replace last } Q \\
&\quad \equiv \, k\#L \land Lk=k \land (\text{swap } (Lk) k. \quad Q) \land L[0;..k] = [0;..k] \land L(k,..\#L) = k,..\#L \\
&\quad \land (L := Lk \rightarrow Lk \mid k \rightarrow L(Lk) \mid L) \\
&\quad \land L[0;..k] = [0;..k] \land L(k,..\#L) = k,..\#L \\
&\quad \Rightarrow L' = L[0;..k] + [k;..\#L] \quad \text{substitution law} \\
&\quad \equiv \, k\#L \land Lk=k \land L[0;..k] = [0;..k] \land L(k,..\#L) = k,..\#L \\
&\quad \land (L := Lk \rightarrow Lk \mid k \rightarrow L(Lk) \mid L) \\
&\quad \land L[0;..k] = [0;..k] \land L(k,..\#L) = k,..\#L \\
&\quad \Rightarrow L' = (L := Lk \rightarrow Lk \mid k \rightarrow L(Lk)) [0;..k] + [k;..\#L] \quad \text{swap does not affect length} \\
&\quad \quad \text{This next step is more complicated and less formal than I would like.} \\
&\quad \text{In the top line it says } L[0;..k] = [0;..k], \text{ and since each item in the list occurs once, the items less than } k \text{ are used up at indexes less than } k. \\
&\quad \text{The top line also says } Lk=k, \text{ therefore } Lk\#k. \text{ So the swap is swapping the item at } k \text{ with an item at an index greater than } k. \text{ The swap does not affect the first part of the list } L[0;..k]. \text{ 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). \text{ So the top line, used as context, allows us to simplify the bottom three lines.} \\
&\quad \equiv \, k\#L \land Lk=k \land L[0;..k] = [0;..k] \land L(k,..\#L) = k,..\#L \\
&\quad \land (L := Lk \rightarrow Lk \mid k \rightarrow L(Lk) \mid L) \\
&\quad \land L[0;..k] = [0;..k] \\
&\quad \land L(k,..\#L) = k,..\#L \\
&\quad \Rightarrow L' = L[0;..k] + [k;..\#L] \quad \text{discharge} \\
&\quad \equiv \, k\#L \land Lk=k \land L[0;..k] = [0;..k] \land L(k,..\#L) = k,..\#L \\
&\quad \land L' = L[0;..k] + [k;..\#L] \quad \text{specialize} \\
&\quad \Rightarrow \, L' = L[0;..k] + [k;..\#L] \\
&\quad \text{And that completes the last case of the } Q \text{ refinement.} \\
&\equiv \, \text{Recursive time is bounded by } 2\times\#L. \text{ Counting just } \text{swaps}, \text{ the time is bounded by } \#L. \\
&\text{To prove time bounds, it is helpful to define} \\
&\quad f_i = \phi_{\xi j} \cdot i_{..\#L} \cdot Lj + j \\
&\text{Then the timing specifications are } A \text{ and } B, \text{ defined as} \\
&\quad A \equiv \, t' \leq t + \#L + f0 \\
&\quad B \equiv \, t' \leq t + \#L - k + f k \]
With time, the refinements are
\[
A \iff k := 0. \quad B
\]
\[
B \iff \text{if } k = \#L \text{ then } \text{ok}
\]
\[
B \iff \text{else if } Lk = k \text{ then } k := k + 1. \quad t := t + 1. \quad B
\]
\[
B \iff \text{else swap } (Lk). \quad t := t + 1. \quad B \quad \Box \quad \Box
\]

Proof of last case of \( B \) refinement:
\[
k := 0. \quad B
\]
\[
\equiv \quad k := 0. \quad t' \leq t + \#L - k + f k
\]
\[
\equiv \quad t' \leq t + \#L - 0 + f 0
\]
\[
\equiv \quad A
\]

Proof of first case of \( B \) refinement:
\[
k = \#L \land \text{ok} \Rightarrow B
\]
\[
\equiv \quad k = \#L \land k' = k \land L' = L \land t' = t \Rightarrow t' \leq t + \#L - k + f k
\]
\[
\equiv \quad k = \#L \land k' = k \land L' = L \land t' = t \Rightarrow t = t + \#L - \#L + f (\#L)
\]
\[
\equiv \quad k = \#L \land k' = k \land L' = L \land t' = t \Rightarrow 0 \leq \$j; \#L, \#L; Lj + j
\]
\[
\equiv \quad k = \#L \land k' = k \land L' = L \land t' = t \Rightarrow 0 \leq 0
\]
\[
\equiv \quad \top
\]

Proof of middle case of \( B \) refinement:
\[
k = \#L \land Lk = k \land (k := k + 1. \quad t := t + 1. \quad B)
\]
\[
\equiv \quad k = \#L \land Lk = k \land (k := k + 1. \quad t := t + 1. \quad t' \leq t + \#L - k + f k)
\]
\[
\equiv \quad k = \#L \land Lk = k \land t' \leq t + 1 + \#L - k - 1 + f (k + 1)
\]
\[
\equiv \quad k = \#L \land Lk = k \land t' \leq t + \#L - k + f (k + 1)
\]
\[
\equiv \quad k = \#L \land Lk = k \land t' \leq t + \#L - k + f k
\]
\[
\Rightarrow \quad B
\]

Proof of last case of \( B \) refinement:
\[
k = \#L \land Lk = k \land (\text{swap } (Lk)). \quad t := t + 1. \quad B
\]
\[
\equiv \quad k = \#L \land Lk = k \land (L := Lk \rightarrow Lk \mid k \rightarrow L(Lk) \mid L. \quad t := t + 1. \quad t' \leq t + \#L - k + f k)
\]

The next step looks like it should be the Substitution Law.

But \( f \) is defined in terms of \( L \). So we have to apply \( f \) first.

\[
\equiv \quad k = \#L \land Lk = k
\]
\[
\land (L := Lk \rightarrow Lk \mid k \rightarrow L(Lk) \mid L. \quad t := t + 1. \quad t' \leq t + \#L - k + \$j; \#L; Lj + j)
\]

Now use the Substitution Law
\[
\equiv \quad k = \#L \land Lk = k
\]
\[
\land (t' \leq t + 1 + \#(Lk \rightarrow Lk) \mid k \rightarrow L(Lk) \mid L) - k
\]
\[
+ \$j; \#(Lk \rightarrow Lk) \mid k \rightarrow L(Lk) \mid L) \cdot (Lk \rightarrow Lk) \mid k \rightarrow L(Lk) \mid L) j + j
\]

swap does not affect length
\[
\equiv \quad k = \#L \land Lk = k \land t' \leq t + 1 + \#L - k + \$j; \#L; Lk \rightarrow Lk \mid k \rightarrow L(Lk) \mid L) j + j
\]

swap reduces the number of out-of-place items by 1 or 2
\[
\Rightarrow \quad k = \#L \land Lk = k \land t' \leq t + 1 + \#L - k + \$j; k, \#L; Lj + j - 1
\]
\[
\equiv \quad k = \#L \land Lk = k \land t' \leq t + \#L - k + f k
\]
\[
\Rightarrow \quad B
\]

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