You are given a list variable \( L \) assigned a nonempty list. All changes to \( L \) must be via procedure \( \text{swap} \), defined as

\[
\text{swap} = \langle i, j \mid \square L \triangleright L_i \rightarrow L_{j+1} | j \rightarrow L_i | L \rangle
\]

(a) Write a program to reassign \( L \) a new list obtained by rotating the old list one place to the right (the last item of the old list is the first item of the new).

\[
\text{Let } R = L'[\#L-1; \ 0; \ \#L-1] \ . \text{ Let } Q = L'[i; 0; \ i+1; \ \#L] \ . \text{ Then}
\]

\[
R \iff i = \#L-1, \ Q
\]

Then

\[
\text{if } i = 0 \ \text{then } \text{ok} \ \text{else} \ \text{swap} \ (i-1) \ . \ i := i-1. \ Q \ \text{fi}
\]

Proof of \( R \) refinement:

\[
i := \#L-1. \ Q
\]

\[
\Rightarrow L' = L[\#L-1; \ 0; \ \#L-1; \ \#L-1+1; \ \#L]
\]

\[
= L'[L[i; 0; \ i+1; \ \#L]
\]

\[
= R
\]

Proof of \( Q \) refinement, first case:

\[
i = 0 \land \text{ok}
\]

\[
\Rightarrow L' = L[i; \ 0; \ i+1; \ \#L]
\]

\[
= Q
\]

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

\[
i + 0 \land (\text{swap} \ (i-1) \ . \ i := i-1. \ Q)
\]

\[
i + 0 \land (L := (i-1) \rightarrow L_{i} | i \rightarrow L(i-1) | L_i := i-1. \ L' = L[i; 0; \ i+1; \ \#L])
\]

\[
\Rightarrow L' = L[i; \ 0; \ i+1; \ \#L]
\]

\[
= Q
\]

Now for the timing.

\[
t' = t + \#L-1 \iff i := \#L-1. \ t' = t + i
\]

\[
t' = t + i \iff \text{if } i = 0 \ \text{then ok} \ \text{else} \ \text{swap} \ (i-1) \ . \ i := i-1. \ t := t+1. \ t' = t+i \ \text{fi}
\]

Proof of first refinement:

\[
i := \#L-1. \ t' = t + i
\]

\[
= t' = t + \#L-1
\]

Proof of last refinement, first case:

\[
i = 0 \land \text{ok}
\]

\[
i = 0 \land L' = L \land i := i \land t' = t
\]

\[
\Rightarrow t' = t + i
\]

Proof of last refinement, last case:

\[
i + 0 \land (\text{swap} \ (i-1) \ . \ i := i-1. \ t := t+1. \ t' = t+i)
\]

\[
i + 0 \land t' = t + 1 + i - 1
\]

\[
\Rightarrow t' = t + i
\]

This should be just right for a for-loop. Let \( M \) be a list constant equal to the original value of \( L \). Since all changes to \( L \) are via \( \text{swap} \), \( \#M = \#L \) always. Let invariant

\[
A i = L = M[0; \#M-i; \ #M-1; \ #M-i; \#M-1]
\]

Then \( A \ L = M \) and \( A'(\#L) = R \).

\[
R \iff \text{for } i := 1; \#L \text{ do } \text{swap} \ (\#L-i-1) \ (\#L-i) \text{ od}
\]

(b) (rotate) Given an integer \( r \), write a program to reassign \( L \) a new list obtained by rotating the old list \( r \) places to the right. (If \( r < 0 \), rotation is to the left \( -r \) places.) Recursive execution time must be at most \( \#L \).
(c) (segment swap) Given an index $p$, swap the initial segment up to $p$ with the final segment beginning at $p$.

The problem can be stated as $0 \leq p < \# L \Rightarrow L' = L[p;\#L ; 0;p]$. We introduce variables $a$ and $b$ for the lengths of the left and right segments. During execution, the extreme parts of the list $L[0;..p–a]$ and $L[p+b;\#L]$ will be in place, with the center portions $L[p–a;..p]$ and $L[p;p+b]$ still to be swapped. Define

$$Q = 0 < b \Rightarrow L' = L[0;..p–a ; p;..p+b ; p–a;..p ; p+b;..\#L]$$

Now refine:

$$0 \leq p < \# L \Rightarrow L' = L[p;\#L ; 0;p] \iff a := p. b := \# L–p. Q$$

$$Q \iff \text{if } a = 0 \text{ then } ok \text{ else if } a < b \text{ then } L := L[0;..p–a ; p+b–a;..p+b ; p;..p+b–a ; p–a;..p ; p+b;..\#L]. b := b–a. Q \text{ else } L := L[0;..p–a ; p;..p+b ; p–a+b;..p ; p–a;..p–a+b ; p+b;..\#L]. a := a–b. Q \text{ fi fi}$$

The two assignments to $L$ are not allowed, so we must still refine them. But they swap segments of equal size, so they are easier than the original problem.

$$L := L[0;..p–a ; p+b–a;..p+b ; p;..p+b–a ; p–a;..p ; p+b;..\#L] \iff$$

$$\text{for } i := p–a;..p \text{ do swap } i (i+b) \text{ od}$$

$$L := L[0;..p–a ; p;..p+b ; p–a+b;..p ; p–a;..p–a+b ; p+b;..\#L] \iff$$

$$\text{for } i := p;..p+b \text{ do swap } i (i–a) \text{ od}$$

The time for the whole problem is $\# L$, and the time for $Q$ is $a+b$. 