Problem

Input: A sequence of n numbers $\langle a_0, a_1, \ldots, a_{n-1} \rangle$.

Output: A reordering $\langle a'_0, a'_1, \ldots, a'_{n-1} \rangle$ of the input sequence such that $a'_0 \leq a'_1 \leq \cdots \leq a'_{n-1}$.

Algorithm

```
merge(A[], T[], l, m, r)
    i = l
    p = l
    q = m
    while True
        if p >= m
            T[i .. r) = A[q .. r)
            break
        else if q >= r
            T[i .. r) = A[p .. m)
            break

        if A[p] <= A[q]
            T[i] = A[p]
            p++
        else
            T[i] = A[q]
            q++
        i++
    A[l .. r) = T[l .. r)

merge-sort-r(A[], T[], l, r)
    if l + 1 >= r
        return
    m = (l + r) // 2
    merge-sort-r(A, T, l, m)
    merge-sort-r(A, T, m, r)
    merge(A, T, l, m, r)

merge-sort(A[0 .. n))
    T = array with size n
    merge-sort-r(A, T, 0, n)
```

Notation

For the following proof, we have some predefined notations.

- For any array A and integers l and r, $A[l:r)$ represents the subarray of A containing the element $A[l], A[l+1], \ldots, A[r-2], A[r-1]$, excluding $A[r]$ (hence the open interval on r). For the case where $l \geq r$, $A[l:r)$ is defined as an empty array.
- When we use the interval notation $i \in [l, r)$, it implies that i, l and r are integers and $l \leq i < r$, not the usual real interval definition.

Correctness

Merge

We first prove that given the two sorted subarray and one working array:

$$A[l:m),\ A[m:r) \text{ and } T[l:r), \text{ where } l < m < r,$$

the procedure `merge` correctly merges them into the sorted subarray $A[l:r)$.

Define the loop invariant $\mathcal{L}$ as follow:

$\mathcal{L}(i)$: At the start of the iteration i of the while loop at line 5-19, $T[l:i)$ contains $(i-l)$ smallest elements from $A[l:r)$ in sorted order. Moreover, $\min(A[p], A[q])$ is the smallest element from $A[l:r)$ that has not been copied into T. If one of p and q is *out of bounds* ($p \in [l, m)$ or $q \in [m, r)$), then we define $\min(A[p], A[q])$ as the one still in bounds. (If $p \geq m$, then $\min(A[p], A[q])$ is $A[q]$. If $q \geq r$, then $\min(A[p], A[q])$ is $A[p]$.)

It is not possible for both of p and q to be out of bounds in an iteration. We will discuss this in the Maintenance section.

1. **Initialization:** Prove that $\mathcal{L}(l)$ is true.

   At the start of the iteration $i = l$, we have $p = l$ and $q = m$.

   (a) $T[l:l)$ is an empty array. It does contain $l - l = 0$ smallest elements from $A[l:r)$.

   (b) $p = l < m$ and $q = m < r$, so $p \in [l, m)$ and $q \in [m, r)$

   (c) Given the fact that $A[l:m)$ and $A[m:r)$ are sorted, $A[l]$ is the smallest in $A[l:m)$ and $A[m]$ is the smallest in $A[m:r)$, so $\min(A[l], A[m])$ is the smallest element from $A[l:r)$.

2. **Maintenance:** Prove that for every $i \in [l, r)$, if $\mathcal{L}(i)$ is true, then $\mathcal{L}(i+1)$ is true. Assume that $\mathcal{L}(i)$ is true. For the case where at least one of p and q is *out of bounds*, it triggers the termination condition at the start of the loop. We will discuss this in the Termination section. For now, we have the following assumption:

   (a) $T[l:i)$ contains $(i-l)$ smallest elements from $A[l:r)$ in sorted order.

   (b) p and q are not *out of bounds*.

   (c) $\min(A[p], A[q])$ is the smallest element from $A[l:r)$ that has not been copied into T.

   Now we have to ensure that $\mathcal{L}(i+1)$ is true. Let's assume $\min(A[p], A[q]) = A[p]$. Then, at the end of this iteration, we have:

   (a) $T[i]$ is the next smallest element to be copied, which is $A[p]$ based on the assumption. This is established through line 12-17. The smaller one of them gets stored in $T[i]$. Thus, $T[l:i+1)$ contains $(i+1-l)$ smallest elements from $A[l:r)$ in sorted order.

   (b) In line 12-17, $A[p]$ gets stored in $T[i]$ and p is increased by one, which may goes out of bounds. Note that only one of p and q gets increased by one in an iteration, so it is not possible for both of them to be out of bounds. For the case where $p = m$, the subarray $A[l:m)$ has all been copied into T, so the elements that has not been copied into T are from the sorted subarray $A[q:r)$. Thus, $A[q]$ is the smallest element from $A[l:r)$ that has not been copied into T.

   (c) For the case where p stays bounded, we have to ensure that $\min(A[p], A[q])$ is still the smallest element from $A[l:r)$ that has not been copied into T. This is also established by the fact that the elements that has not been copied into T are from the sorted subarrays $A[p:m)$ and $A[q:r)$.

   Similarly, if $\min(A[p], A[q]) = A[q]$, we have the same conclusion that $\mathcal{L}(i+1)$ is true.

3. **Termination:** In each iteration, either p or q increments, and both are bounded above, so the loop must terminate. At the iteration $i = t$ where termination occurs, we have either $p = m$ or $q = r$. By the initialization and the maintenance of $\mathcal{L}$, we have $\mathcal{L}(t)$. Assume that $p = m$, we have $t = m + (q - m) = q$, then

   (a) $T[l:t)$ contains $(t-l)$ smallest elements from $A[l:r)$ in sorted order.

   (b) $A[q]$ is the smallest element from $A[l:r)$ that has not been copied into T.

   Since $A[l:m)$ has all been copied into T, and $A[q]$ is the smallest element from $A[l:r)$ that has not been copied into T, we can copied the whole the sorted subarray $A[q:r)$ at $T[t:r)$. Then the termination occurs, we have $A[l:r)$ merged and sorted into $T[l:r)$.

   Similarly, for the case where $q = r$, $t = p + (r - m)$, we have the conclusion that by copying the remaining $A[p:m)$ to $T[t:r)$, we have $A[l:r)$ merged and sorted into $T[l:r)$.

At line 20, we copy T back to A. Therefore, the procedure `merge` correctly merge $A[l:m)$ and $A[m:r)$ into the sorted subarray $A[l:r)$.

Merge Sort

We prove by strong induction that for all $n \geq 0$, the following $P(n)$ is true.

$P(n)$: for every integers l and r, if $n = r - l$, then `merge-sort-r(A, T, l, r)` correctly sorts the subarray $A[l : r)$ given a working array $T[l : r)$.

1. **Base Case:** When $n \leq 1$, the function immediately returns without modifying any content. Since a subarray with size 0 or 1 is already sorted, $P(0)$ and $P(1)$ are both true.

2. **Inductive Hypothesis:** Assume that for some integer $k \geq 1$, $P(i)$ is true for all integers i where $0 \leq i \leq k$.

3. **Inductive Step:** Prove that $P(k+1)$ is true based on the inductive hypothesis.

   For every l and r such that $r - l = k + 1$, since $k + 1 \geq 2$, that is $r - l \geq 2$, the procedure goes through line 25-28. In line 25, we have

   $$m = \left\lfloor \frac{l+r}{2} \right\rfloor = \left\lfloor l + \frac{r-l}{2} \right\rfloor = l + \left\lfloor \frac{r-l}{2} \right\rfloor = l + \left\lfloor \frac{k+1}{2} \right\rfloor.$$

   In line 26-27, the two subarrays are sorted based on the inductive hypothesis:

   $m - l = \left\lfloor \frac{k+1}{2} \right\rfloor$

   Upper bound: $\left\lfloor \frac{k+1}{2} \right\rfloor < k+1$

   Lower bound: $\left\lfloor \frac{k+1}{2} \right\rfloor \geq 1$

   $\Rightarrow 1 \leq m - l < k + 1$,

   $\Rightarrow P(m - l)$ is true.

   $r - m = k + 1 - \left\lfloor \frac{k+1}{2} \right\rfloor = \left\lceil \frac{k+1}{2} \right\rceil$

   Upper bound: $\left\lceil \frac{k+1}{2} \right\rceil < \frac{k+1}{2} + 1 \leq k + 1$

   Lower bound: $\left\lceil \frac{k+1}{2} \right\rceil \geq \frac{k+1}{2} \geq 1$

   $\Rightarrow 1 \leq r - m < k + 1$,

   $\Rightarrow P(r - m)$ is true.

   In line 28, the procedure `merge` correctly merge and sorted $A[l : r)$ given the working array $T[l : r)$ since $A[l : m)$ and $A[m : r)$ are sorted, and $l < m < r$.

   In addition, the recursion terminates because each recursive call operates on a strictly smaller subarray, and the base case handles subarrays of size ≤ 1. Thus, $P(k+1)$ is true.

4. **Conclusion:** By the principle of strong induction, $P(n)$ is true for all integers $n \geq 0$.

Therefore, in the procedure `merge-sort`, by creating a working array $T[0 : n)$, the array $A[0 : n)$ is correctly sorted by `merge-sort-r(A, T, 0, n)`, thus solving the sorting problem.