Exercise 1:

(i) We now consider the unbounded arrays:

Given an array l with length $2n_0$ which has m elements, to delete the last k elements $(m > k)$, we just need to decrease the length of the original array by k. So for the $delete\_last\ (l, k)$ function, its original complexity is $\Theta(1)$.

But after the deletion, we may need to make deallocation if $(m - k) < \frac{1}{2}n_0$. In fact, since we do not know anything about the number of remaining elements, we may need to make several deallocations in succession. These deallocations should be also counted into complexity, so considering these exceptional cases, we calculate the amortized complexity instead.

In order to determine the amortized complexity, we apply the counter method here:

Since we want to prove that the amortized complexity is $\Theta(1)$, the contribution of the $delete\_last\ (l, k)$ itself to the whole sequence of operations must be a constant, let's assume it be X. A constant can't cover all the deallocations, so we need to increase the contribution of the operations when building the array. i.e. the $append\ (l, x)$ and $insert\ (l, i, x)$ operations.

Let's reset the contribution of each $append\ (l, x)$ and $insert\ (l, i, x)$ operation be $3 * c$ instead of $2 * c$, which doesn't change the amortized complexity of them. As we have proved in the lecture, if the contribution is $2 * c$, then all the $append\ (l, x)$ and $insert\ (l, i, x)$ operations will cover the subtraction of all the allocation operations. So after we have built the array l with length $2n_0$ which has m elements, the counter C satisfies:

$$C \geq 3 * c * m - 2 * c * m = c * m$$

After adding the contribution of $delete\_last\ (l, x)$, we have:

$$C \geq c * m + X$$

Then we need to consider all the deallocation operations:

First, we need to determine how many deallocations we need:

$$N = \left\lfloor \log_2 \frac{2n_0}{m-k} \right\rfloor - 1$$

Here, $\frac{2n_0}{l-k}$ is the fraction of the remaining elements to the length of original array. For example, if $2n_0 = 16,\ m - k = 3$, we need to only deallocate $\left\lfloor \log_2 \frac{16}{3} \right\rfloor - 1 = 1$ time.

Now, we can do the following:

Since after the final allocation operation when building the array, we must have $m \geq n_0$, so:

$$N \leq \log_2 \frac{2n_0}{m-k} - 1 \leq \log_2 \frac{2m}{m-k} - 1 = \log_2 \frac{m}{m-k} < \frac{m}{m-k}$$

For each deallocation, we copy all the $m - k$ elements, so the total number of steps of deallocations is $(m-k) * c * N < (m-k) * c * \frac{m}{m-k} = m * c$.

So, after subtracting these, the counter C remains:

$$C \geq c * m + X - \big((m-k) * c * N\big) > X$$

So X can be any number, including 0. That is, the contribution of the $delete\_last\ (l, k)$ function itself is 0.

So we have the amortized complexity of $delete\_last(l, k)$ function is $\Theta(1)$, which is independent of k.

Now we consider the double-liked list:

Following the same procedure, for the $append\ (l, x)$ and $insert\ (l, i, x)$ operations when building the list, we set the contribution of each operation is 1, which doesn't change the amortized complexity.

Since we have m elements for the original list, the contribution of all the $append\ (l, x)$ and $insert\ (l, i, x)$ operations for the counter C is m. When we delete the last k elements, we need to apply $remove\ (l, x)$ for 1 time, but we need k steps to find the position where to separate the list.

Considering the two steps above, the final counter number is:

$$C = m - k \geq 0$$

So the $append\ (l, x)$ and $insert\ (l, i, x)$ operations can cover the delete operation. The amortized complexity of the $delete\_last\ (l, x)$ is $\Theta(1)$.