# Random Delete Arrays

We are interested in a datatype which stores a collection of elements and supports the operations *Insert*, which adds a new element to the collection, and *DeleteRandom*, which deletes and returns a random (not an arbitrary) element.

We have the following contraints on the data structure which implements this datatype:

The elements should be stored in an array indexed from zero. We implement *DeleteRandom* using a random number generator. If `random()` return a number $r$ from the interval $[0..1)$, then we choose the element $\lfloor rn \rfloor$, where $n$ is the number of elements in the structure at the given time.

**Question a:** Assume that we know an upper bound on how large the size of the collection can become. Write pseudo-code which implements both operations in $O(1)$, assuming that `random()` runs in $O(1)$. □

Now we no longer have an upper bound on the size of the collection.

**Question b:** We want to limit space usage to $O(n)$. To do that, we sometimes allocate a new array of a different size, move all elements into the new array, and deallocate the old array (release the space to the operating system). We let $s$ denote the size of the array (which is always at least $n$).

- if $n = s$ and *Insert* is called, a new array of size $2s$ is created.
- if $n = \frac{s}{4}$ and *DeleteRandom* is called, a new array of size $\frac{s}{2}$ is created.

Show that both operations have running times amortized $O(1)$ and that space usage is $O(n)$. The potential function $\Theta(n, s) = 2 \cdot |\frac{s}{2} - n|$ (or some variant hereof) might be useful. □

**Question c:** Explain how both operations can be implemented to run in worst-case $O(1)$ while space usage is still $O(n)$. □