```
def _resize(self, c):
    B = self._make_array(c)
    for k in range(self._n):
        B[k] = self._A[k]
    self._A = B
    self._capacity = c

def _make_array(self, c):
    return (c * ctypes.py_object)()
```

Amortized Analysis of Dynamic Arrays

→ using amortization, we can show that performing a sequence of append operations on a dynamic array is actually efficient.

→ **Proposition:**

Let S be a sequence implemented by means of a dynamic array with initial capacity one, using the strategy of doubling array size when full. The total time to perform a series of n append operations in S, starting with S being empty, is $O(n)$.

→ **Proposition:**

Performing a series of n append operations of an initially empty dynamic array using a fixed increment with each resize taking $\Omega(n^2)$ time.

Efficiency of Python's Sequence Types

Operation	Running Time
len(data)	$O(1)$
data[j]	$O(1)$
data.count(val)	$O(n)$
data.index(val)	$O(k+1)$
value in data	$O(k+1)$
data <=>!= data2	$O(k+1)$
data[j:k]	$O(k-j+1)$
data1+data2	$O(n_1+n_2)$
c* data	$O(cn)$