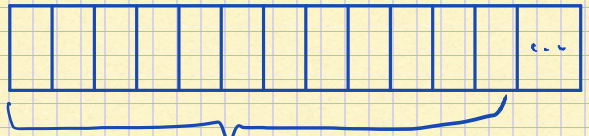
Array.

1. Generics: No restrictions on the type
2. Memory controller
   the low level implementation



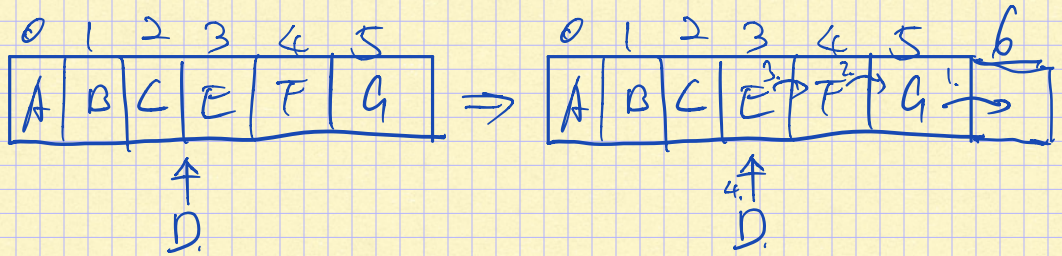
Contiguous space in memory.

3. Time complexity is O(1) for random lookup
4. Time complexity is O(n) for delete or add.

recap:

Time Complexity.
- O(n) ⇒ delete/add on average
- O(1) ⇒ random lookup.

Array add

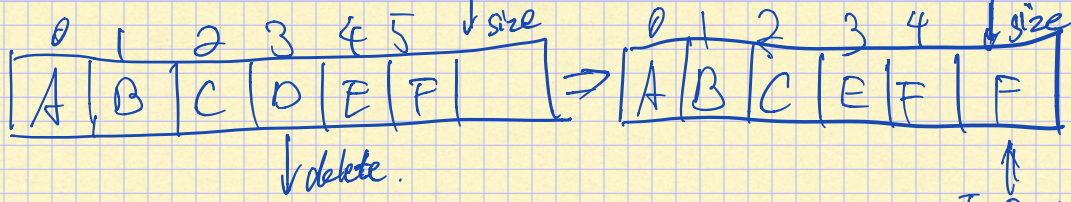



```
def add(idx, ele):
    # assume capacity is enough, idx is valid.
    for i in range(size-1, idx, -1): → O(n)
        arr[i+1] = arr[i]          # first make space for D.
    arr[idx] = D  #  add D
    size += 1  # update size
```

recap: The key is to start from the rear, move last element first, here I assume capacity is enough (no need to resize)

make sure capacity > size and $0 \le idx \le size$

Array delete




```
def delete(idx):
    # again assume idx is valid   0 ≤ idx ≤ size.
    for i in range(idx, size-1):        → O(n)
        arr[i] = arr[i+1]

    size -= 1
```

Recap: Starts from idx, replace. with the value behind, update size

Array resizing

1. when array size is approaching array capacity we need to enlarge the capacity
   a. Create a new array of 2x capacity.
   b. Copy all existing elements to the new array
   c. reference array name to new array.
2. when array size is decreasing to a small number, we need to shrink the capacity to save space
3. if the trigger of enlarging / shrinking is not set properly, it might cause the volatility at the Border

recap: ↑ size → capacity ⇒ capacity × 2

↓ size → 1/4 capacity ⇒ capacity × 1/2 ⇒ to avoid volatility