Arrays

September 9, 2021

3. Arrays

0.0.1 3.1 The Array structure

A **one-dimensional array** is composed of multiple sequential elements stored in contiguous bytes of memory and allows for random access to the individual elements.

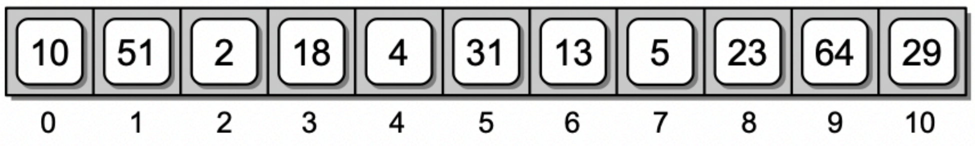
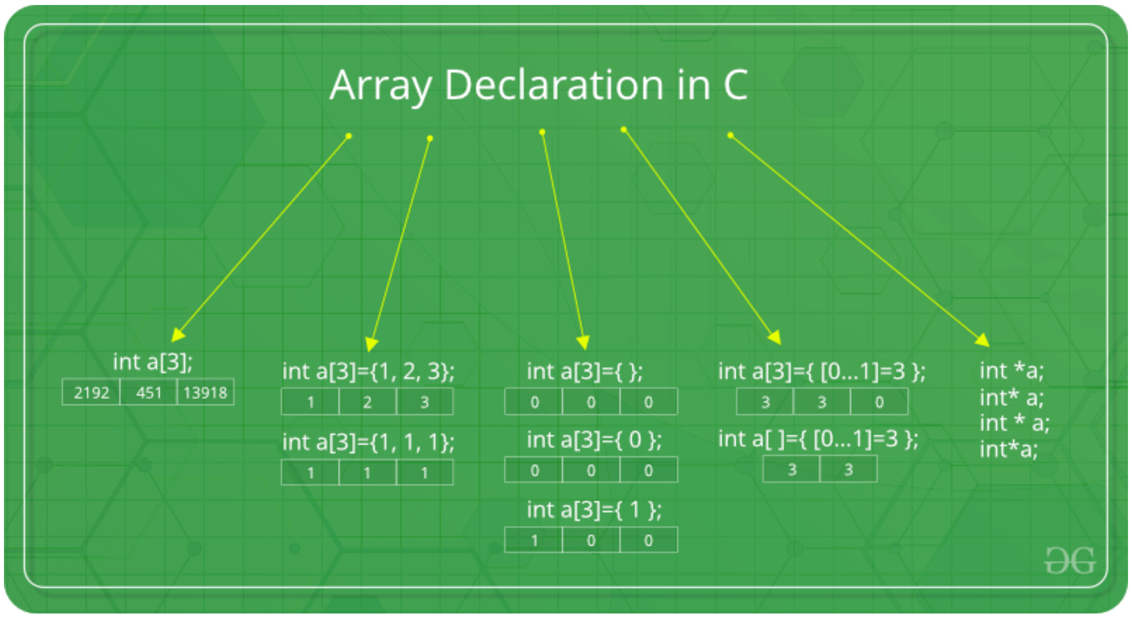


Figure 2.1: A sample 1-D array consisting of 11 elements.



0.0.2 3.1.1 Why Study Arrays?

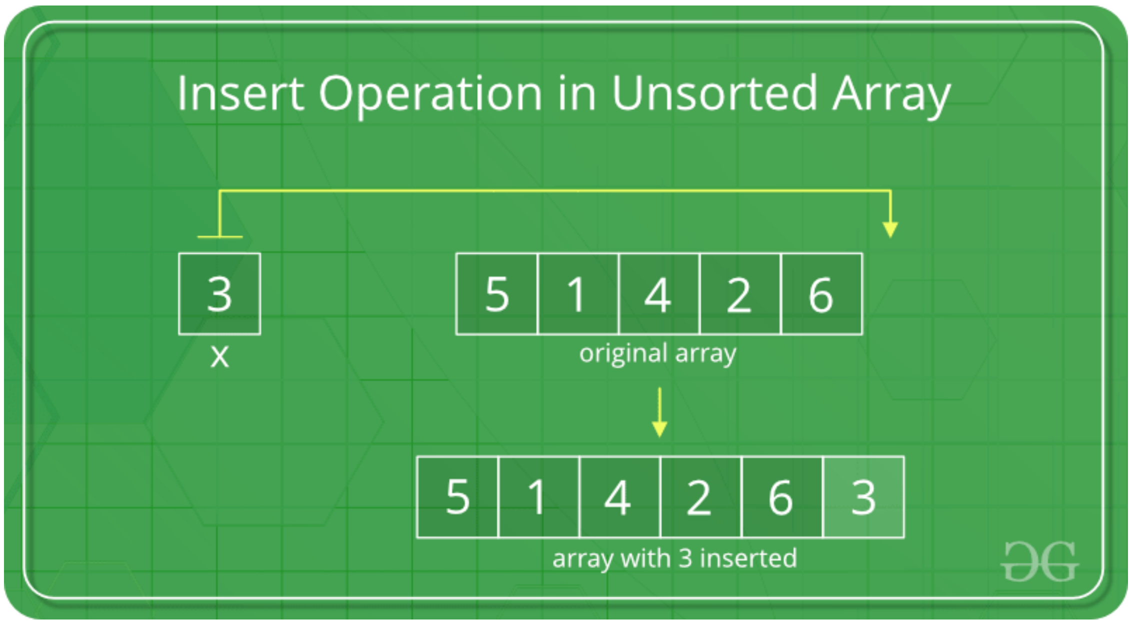
The array structure looks very similar to Python's list structure. That's because the two structures are both sequences that are composed of multiple sequential elements that can be accessed by position. But there are two major differences between the array and the list. - **First**, an array has a limited number of operations, which commonly include those for array creation, reading a value from a specific element, and writing a value to a specific element. The list, on the other hand, provides a large number of operations for working with the contents of the list. - **Second**, the list can grow and shrink during execution as elements are added or removed while the size of an array cannot be changed after it has been created.

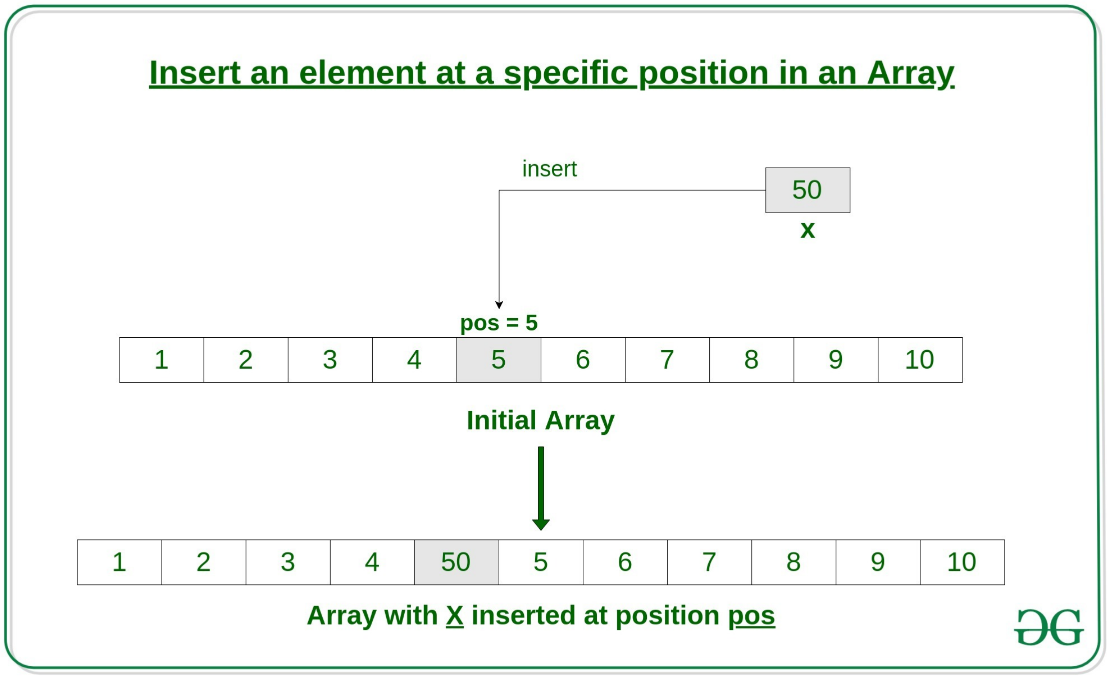
0.0.3 3.1.2 The Array Abstract Data Type

A **one-dimensional** array is a collection of contiguous elements in which individual elements are identified by a unique integer subscript starting with zero. Once an array is created, its size cannot be changed.

- **Array(size)**: Creates a one-dimensional array consisting of size elements with each element initially set to None. size must be greater than zero.
- **length ()**: Returns the length or number of elements in the array.
- **getitem (index)**: Returns the value stored in the array at element position index. The index argument must be within the valid range. Accessed using the subscript operator.
- **setitem (index, value)**: Modifies the contents of the array element at position index to contain value. The index must be within the valid range. Accessed using the subscript operator.
- **removeitem (index)**: Remove the array element at position index. The index must be within the valid range. Accessed using the subscript operator.
- **clearing(value)**: Clears the array by setting every element to value.
- **iterator ()**: Creates and returns an iterator that can be used to traverse the elements of the array.

0.1 Array `insert`




Insert an element at a specific position in an Array
insert
50
X
pos = 5
1 2 3 4 5 6 7 8 9 10
Initial Array
1 2 3 4 50 5 6 7 8 9 10
Array with X inserted at position pos

[38]:
```
def insert(arr, index, value):
    if index < 0 and index >= len(arr):
        raise Exception
    tmp = arr[index]
    for i in range(len(arr)+1, index+1):
        a[i] = a[i-1]
    a[index] = value
    return arr #arr[:index] + tmp + arr[index:]
```

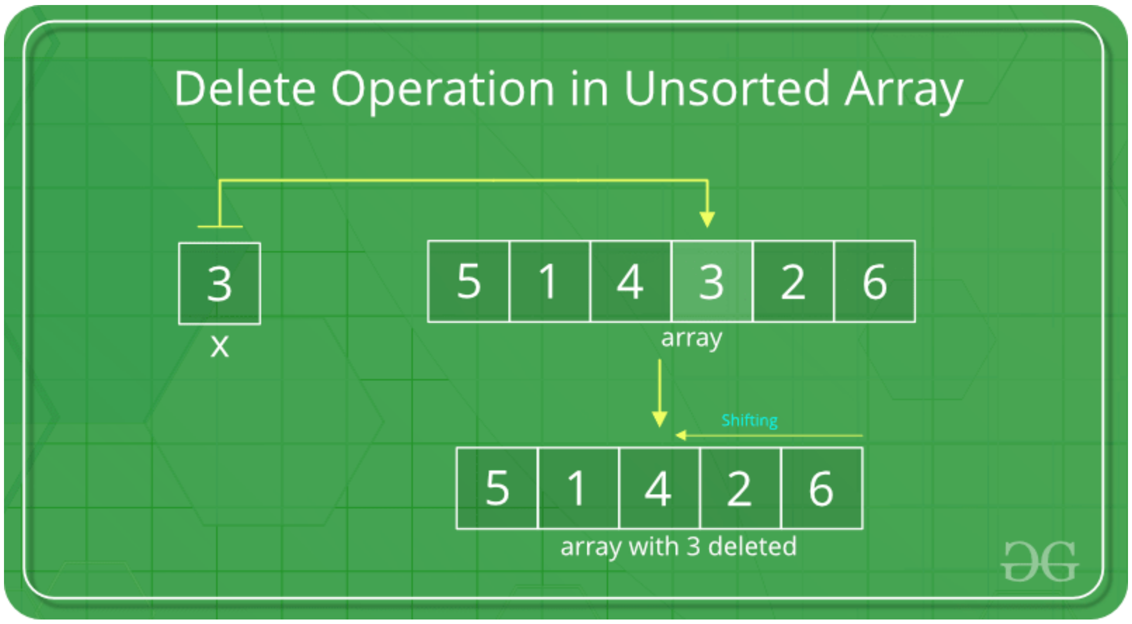
[39]:
```
a = [10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]
print(a)
```

```
[10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]
```

[40]:
```
insert(a, 2, 100)
```

[40]:
```
[10, 51, 100, 18, 4, 31, 13, 5, 23, 64, 29]
```

0.2 Array remove



[ ]:
```

```

[41]:
```
def remove(arr, index):
    if index < 0 and index >= len(arr):
        raise Exception
    for i in range(index, len(arr)-1):
        a[i] = a[i+1]
```

[42]:
```
a = [10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]
print(a)
```

```
[10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]
```

[43]:
```
remove(a, 2)
print(a)
```

```
[10, 51, 18, 4, 31, 13, 5, 23, 64, 29, 29]
```

0.2.1 Reference:

[1] **Chapter 2**: Necaise, Rance D - Data structures and algorithms using Python-John Wiley & Sons (2010_2011)

```
[ ]:
```

“‘C int unique(S) { int i, j ; i = 0; j = 0;

for(i = 0; i < length(S); i ++) { for(j = i +1; j < length(S)-1; j++ { if(S[i] == S[j] return 0; } } return 1; }“‘

```
[ ]:
```