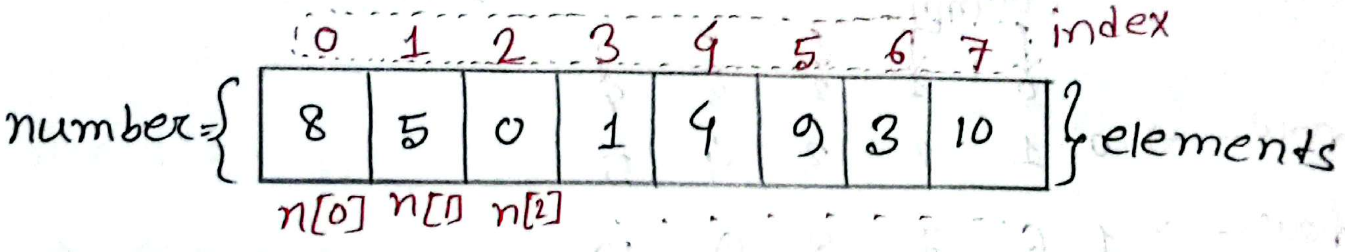
# Array:

An array is a collection of elements accessible by index.



- The items of an array are called elements.
- Each element is a value
- Indexing begins at zero.
- The array forms a continuous list in memory.
- Array can be different types.
- An array have a name: like → number={1,2}
- Array provide random access.

n[ ]={1,2}
index
element

- time complexity o(1).
- ~~Array size is (index~~
- Array size is (last index + 1)

## Two dimensional Array

Two dimensional array is a collection of similar data elements where each element is referenced by two subscripts.

column

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| row → 0 | 1 | 2 | 3 | 4 | 5 |
| row → 1 | 6 | 7 | 8 | 9 | 10 |
| row → 2 | 11 | 12 | 13 | 14 | 15 |
| row → 3 | 16 | 17 | 18 | 19 | 20 |

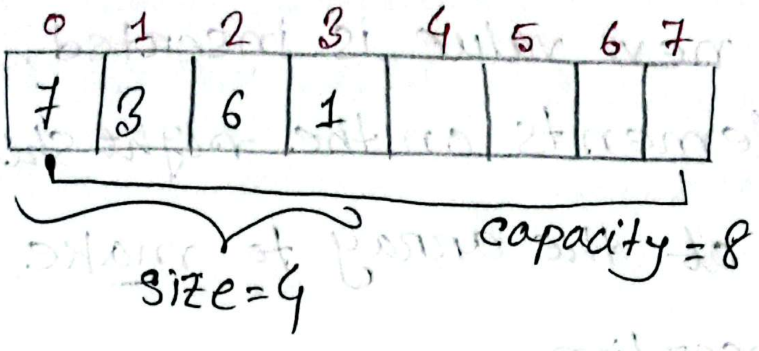
A[1][2] = 8

row

column

- Such arrays ~~are~~ are called matrix in math and tables in business application.

# Dynamic Array



- The size in a dynamic array is the number of elements it actually uses.
- The number of all elements that a dynamic array can use is called the capacity. The capacity increases automatically.

in python we know it "List".

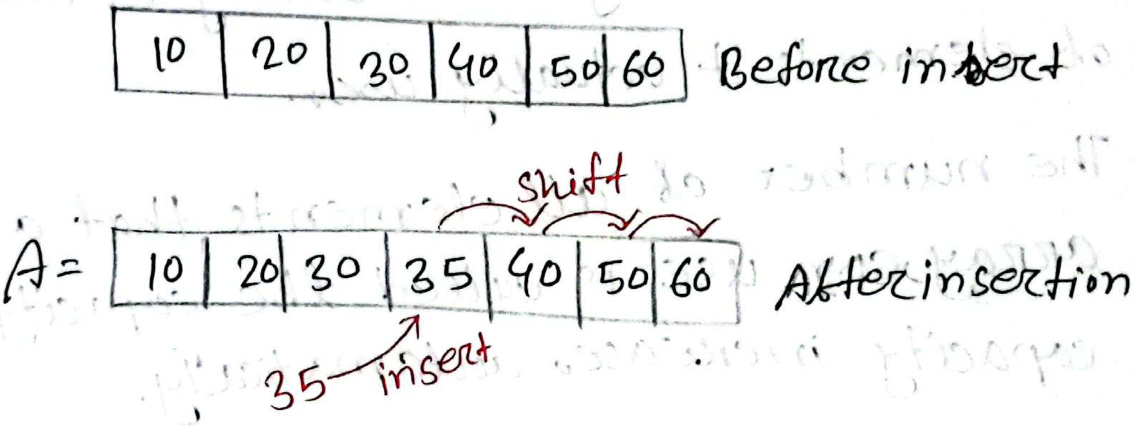
in Java we know it "ArrayList".

In C++ we can create a dynamic array.

```
int* array = new int[size];
```

## Insertion

When a new value is inserted, the values of the elements on the right shifted to the end of the array to make a space for the insertion.



### Pseudocode

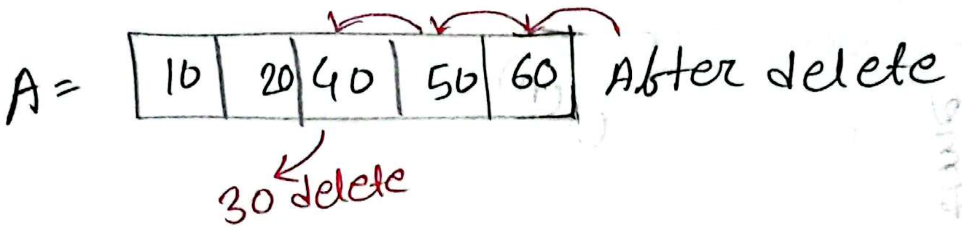
```
initialize j=size
WHILE J>=position
    A[J+1] = A[J] (shift right)
    j=j-1
A[position]=item (insert element)
set, size ← size+1
```

## Deletion

When a value is deleted, the values of the elements on the right are shifted to the beginning of the array because on gaps are allowed in the array.

| 10 | 20 | 30 | 40 | 50 | 60 |
|---|---|---|---|---|---|

Before delete

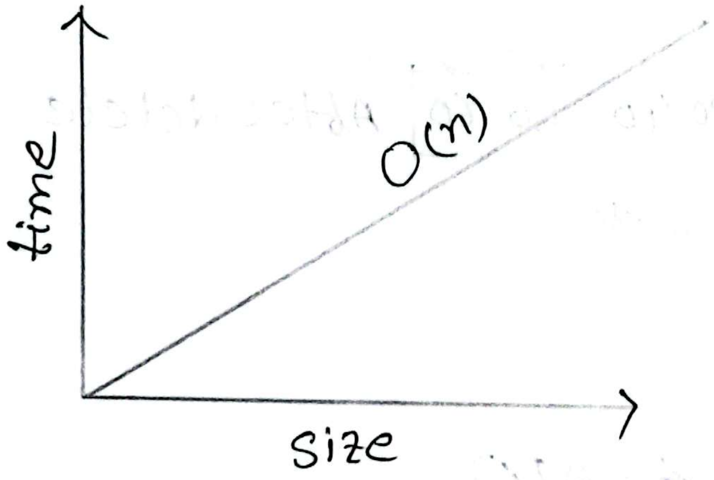


### Pseudocode

```
1. set, position ← A[i]
2. initialize j = position
3. WHILE j <= size
4. A[j] ← A[j+1]
5. j = j+1
6. reset, size ← size-1
```

## Complexity

The time complexity of insertion and deletion at the beginning or middle part of the array is $O(n)$. This is due to the fact that we shift left or right $k*n (0 \leq k \leq 1)$ elements.



## Linear search

Linear search is an algorithm for searching through elements of an array.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 3 | 9 | 8 | 2 | 1 | 4 | 6 | 5 | 7 |

Search: 6

First, we examine the leftmost number in the array. We compare it with 6, and if matches, the search ends. If it doesn't match, we examine the next number to the right. We repeat the comparisons untill the 6 is found.

# Since the search has to start from the beginning, linear search takes more time for large amounts of data.

Time complexity → O(n)

## Binary Search

Binary search is an algorithm for searching through elements of a presorted array.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Search : 5

First, we will find the mid by dividing the starting index and the ending index by 2. If the value of mid is less than the searched value, then we will set the starting index to mid+1. If the value of mid is greater than the searched value, then we will set ending index to mid-1.

Time complexity → $O(1)$, target is found at the middle on first try.

$O(\log_2 n)$

# Pseudocode

```
1. Set, beginning index = 0 , last index = size-1.
2. set,   mid = $\left( \frac{beg + last}{2} \right)$

3. WHILE (beg <= last)
        if, item = A[mid]
            return mid,

        if, item < A[mid]
            set, last = mid-1
        if,
            item > A[mid]
                set, beg = mid+1

4. Return -1
```