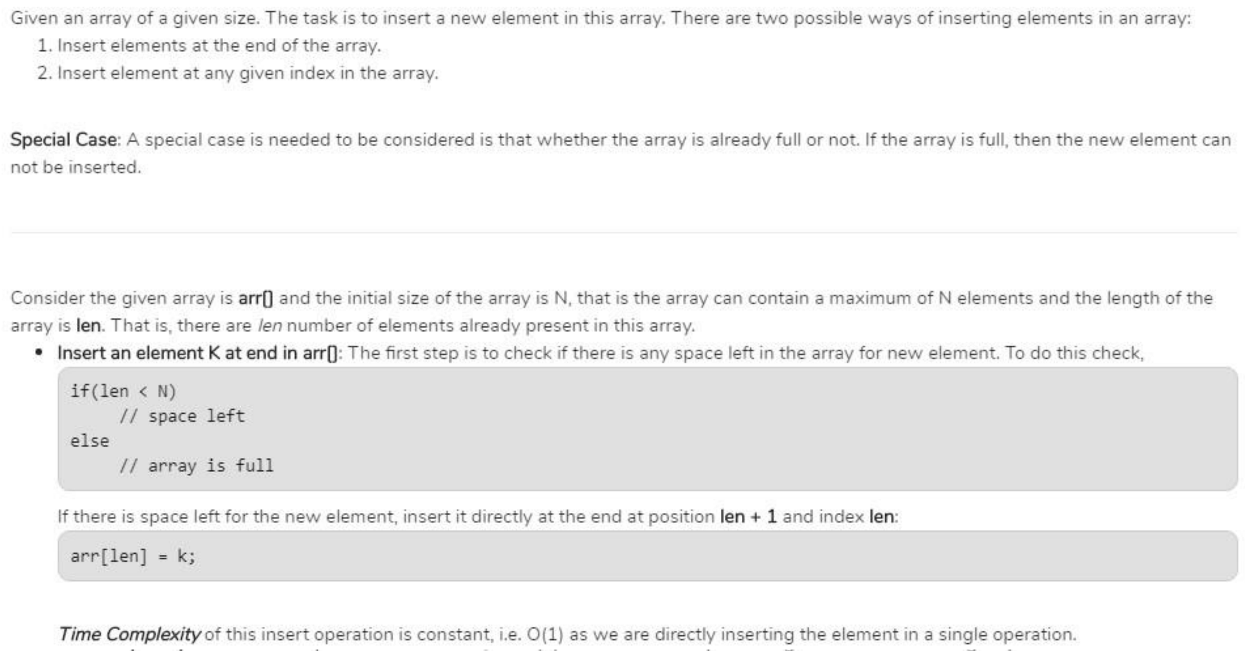
Insertion in Arrays



Given an array of a given size. The task is to insert a new element in this array. There are two possible ways of inserting elements in an array:

1. Insert elements at the end of the array.
2. Insert element at any given index in the array.

Special Case: A special case is needed to be considered is that whether the array is already full or not. If the array is full, then the new element can not be inserted.

Consider the given array is **arr[]** and the initial size of the array is N, that is the array can contain a maximum of N elements and the length of the array is **len**. That is, there are *len* number of elements already present in this array.

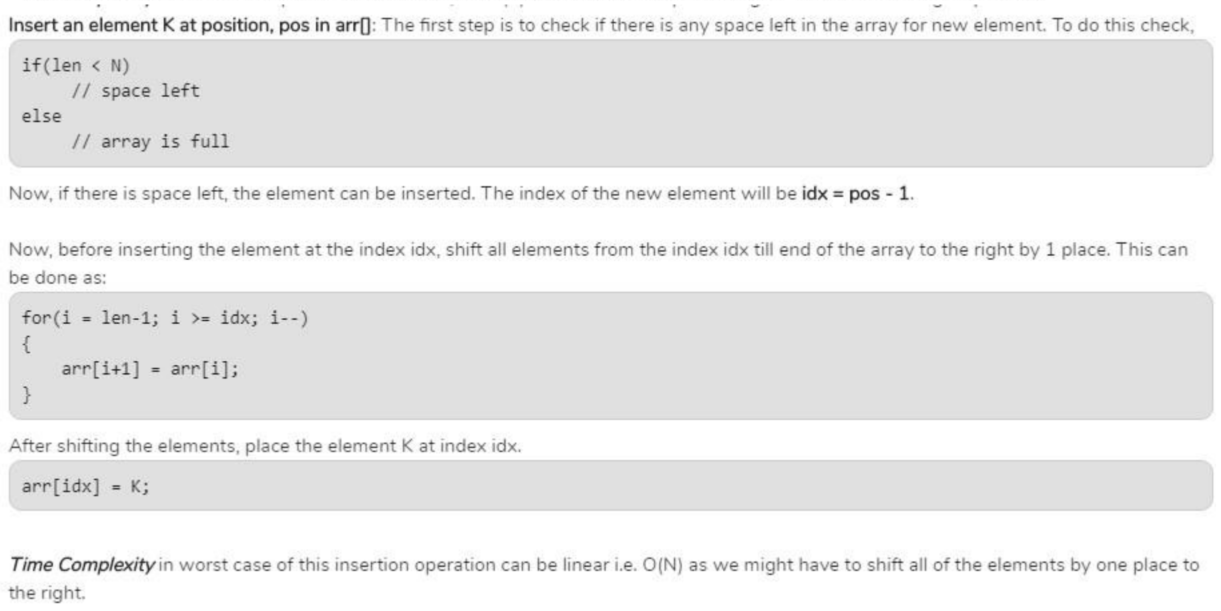
- **Insert an element K at end in arr[]**: The first step is to check if there is any space left in the array for new element. To do this check,

  ```
  if(len < N)
       // space left
  else
       // array is full
  ```

  If there is space left for the new element, insert it directly at the end at position **len + 1** and index **len**:

  ```
  arr[len] = k;
  ```

  Time Complexity of this insert operation is constant, i.e. O(1) as we are directly inserting the element in a single operation.



- **Insert an element K at position, pos in arr[]**: The first step is to check if there is any space left in the array for new element. To do this check,

  ```
  if(len < N)
       // space left
  else
       // array is full
  ```

  Now, if there is space left, the element can be inserted. The index of the new element will be **idx = pos - 1**.

  Now, before inserting the element at the index idx, shift all elements from the index idx till end of the array to the right by 1 place. This can be done as:

  ```
  for(i = len-1; i >= idx; i--)
  {
      arr[i+1] = arr[i];
  }
  ```

  After shifting the elements, place the element K at index idx.

  ```
  arr[idx] = K;
  ```

  Time Complexity in worst case of this insertion operation can be linear i.e. O(N) as we might have to shift all of the elements by one place to the right.

Deletion in Arrays

To delete a given element from an array, we will have to first search the element in the array. If the element is present in the array then delete operation is performed for the element otherwise the user is notified that the array does not contains the given element.

Consider the given array is **arr[]** and the initial size of the array is N, that is the array can contain a maximum of N elements and the length of the array is **len**. That is, there are *len* number of elements already present in this array.

Deleting an element K from the array arr[]: Search the element K in the array arr[] to find the index at which it is present.

```
for(i = 0; i < N; i++)
{
    if(arr[i] == K)
        idx = i; return;
    else
        Element not Found;
}
```

Now, to delete the element present at index **idx**, left shift all of the elements present after *idx* by one place and finally reduce the length of the array by 1.

```
for(i = idx+1; i < len; i++)
{
    arr[i-1] = arr[i];
}

len = len-1;
```

Time Complexity in worst case of this insertion operation can be linear i.e. O(N) as we might have to shift all of the elements by one place to the left.

Searching in an Array

Searching an element in an array means to check if a given element is present in an array or not. This can be done by accessing elements of the array one by one starting from the first element and checking if any of the element matches with the given element.

We can use loops to perform the above operation of array traversal and access the elements using indexes.

Suppose the array is named **arr[]** with size **N** and the element to be searched is referred as **key**. Below is the algorithm to perform to the search operation in the given array.

```
for(i = 0; i < N; i++)
{
    if(arr[i] == key)
    {
        print "Element Found";
    }
    else
    {
        print "Element not Found";
    }
}
```

Time Complexity of this search operation will be O(N) in the worst case as we are checking every element of the array from 1st to last, so the number of operations is N.