In this post, we will look into insertion operation in an Array, i.e., how to insert into an Array, such as:

1. Inserting Elements in an Array at the End
2. Inserting Elements in an Array at any Position in the Middle
3. Inserting Elements in a Sorted Array
4. **1. Inserting Elements in an Array at the End**
5. In an unsorted array, the insert operation is faster as compared to a sorted array because we don’t have to care about the position at which the element is to be placed.

// C program to implement insert

// operation in an unsorted array.

#include <stdio.h>

// Inserts a key in arr[] of given capacity.

// n is current size of arr[]. This

// function returns n + 1 if insertion

// is successful, else n.

int insertSorted(int arr[], int n, int key, int capacity)

{

// Cannot insert more elements if n is

// already more than or equal to capacity

if (n >= capacity)

return n;

arr[n] = key;

return (n + 1);

}

// Driver Code

int main()

{

int arr[20] = { 12, 16, 20, 40, 50, 70 };

int capacity = sizeof(arr) / sizeof(arr[0]);

int n = 6;

int i, key = 26;

printf("\n Before Insertion: ");

for (i = 0; i < n; i++)

printf("%d ", arr[i]);

// Inserting key

n = insertSorted(arr, n, key, capacity);

printf("\n After Insertion: ");

for (i = 0; i < n; i++)

printf("%d ", arr[i]);

return 0;

}

Before Insertion: 12 16 20 40 50 70

After Insertion: 12 16 20 40 50 70 26

**Time Complexity:** O(n)

**Auxiliary Space:**O(1)

Let’s break down the time and space complexity of the provided code step by step:

1. \*\*Array Initialization:\*\*

– The array `arr` is initialized with 20 elements.

– This takes O(1) time and O(20) space, which simplifies to O(1) space.

2. \*\*Array Size Calculation:\*\*

– The size of the array (`capacity`) is calculated using `sizeof(arr) / sizeof(arr[0])`.

– This takes O(1) time.

3. \*\*Loop to Print the Array:\*\*

– There is a loop to print the elements of the array, which has a constant number of iterations (6 in this case).

– This takes O(n) time, where n is the number of elements in the array.

4. \*\*Insertion Function (`insertSorted`):\*\*

– The insertion function itself is very simple and consists of constant time operations:

– Checking if the array is full (`n >= capacity`) takes O(1) time.

– Assigning the key to the array at the end and updating `n` takes O(1) time.

5. \*\*Loop to Print the Array After Insertion:\*\*

– Similar to the loop before insertion, this loop has a constant number of iterations (n + 1 in this case).

– This takes O(n) time.

Therefore, the overall time complexity of the code is dominated by the loop iterations for printing the array, and the insertion and checking operations are constant time. Hence, the time complexity is O(n).

In terms of space complexity, the code uses a constant amount of extra space for variables (`i`, `key`, `capacity`) and the array itself. Thus, the space complexity is O(1).

To summarize:

– Time Complexity: O(n)

– Space Complexity: O(1)

2. Inserting Elements in an Array at any Position

Insert operation in an array at any position can be performed by shifting elements to the right, which are on the right side of the required position