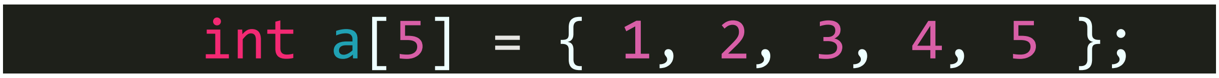
1. INITIALIZATION OF ARRAY AND ITS TIME COMPLEXITY.

THERE ARE TWO TYPES OF ARRAYS ON BASIS OF MEMORY ALLOCATION:

1. **STATIC ARRAY:** Which takes a fixed size of array at compile time, consumes a contiguous memory allocation with an index numbered opposite to each element that it consumes in the memory. They get destroyed by themselves after program gets compiled.



```
int a[5] = { 1, 2, 3, 4, 5 };
```

Therefore, at compile time it will all allocate $5 \times 4\ bytes = 20\ bytes\ of\ memory$.

2. DYNAMIC ARRAY:

→When it is created may be have a primary size but consume a large size in heap memory.

→The size is only be determined at the run time.

→In C/C++ ,it did not get destroyed but we destroy or free the memory manually .Note , "free" keyword is only associated with "malloc" .

IN C++

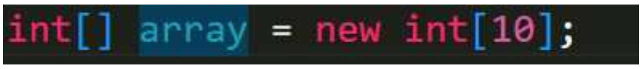
```
int *a;
a = (int*)malloc(size * sizeof(int));
free(a);//free up memory.
```

<u>where malloc stands for memory allocation</u>

→Replacing malloc , the new keyword introduced in C++ reducing malloc syntax and we "delete the array" with "delete" keyword. Hence the initialization with new keyword is dynamic memory allocation.

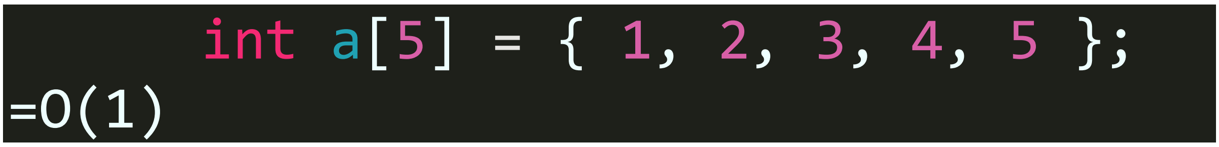
```
 int *a;
a = new int[size];
delete[] a;
```

→In Java , we remove pointer and the use of dot operator and new keyword continued due to security of address. Through pointer address is exposed , hence removed from java. Even after we create a dynamic array in java with new keyword and it stays in the memory , the reference of the unused array becomes useless and garbage collector of java compiler takes care of it i.e. first it deallocates and free up the memory.

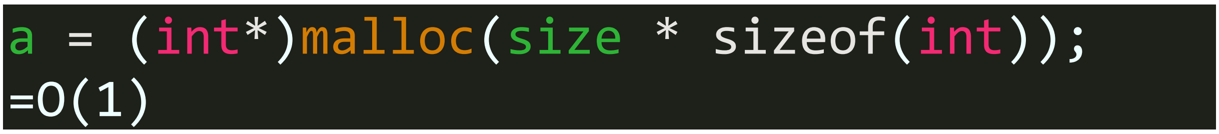


```
int[] array = new int[10];
```

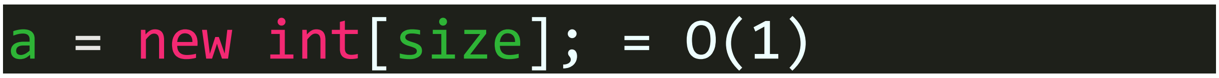
Time Complexity



```
     int a[5] = { 1, 2, 3, 4, 5 };
=O(1)
```



```
a = (int*)malloc(size * sizeof(int));
=O(1)
```



```
a = new int[size]; = O(1)
```

Though having distinct size , Array while initialization , it takes 1 unit of time to get initialized.

Hence upper bound is: Big-O ⟹ O(1)