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 with a primary size consuming a large size

in heap memory and 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 but in Java , even after we create a dynamic array 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.

IN C++

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

where malloc stands for memory allocation

Replacing malloc and pointer, the new keyword introduced in C++ generally for address security. Hence the initialization with new keyword is dynamic memory allocation.

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

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 $\Longrightarrow O(1)$