Complexity:

What is "big Oh" ? for:

- *1:*

```
for(i=0;i<n*n; i++)
      {
       for(j=i; j<n; j++)
           k++;
      }
```

  NOTES:

- *2:*

```
for(i=0; i<n; i++)
      {
       for(j=i; j>0; j=j/2)
           k++;
      }
```

  NOTES:

- *3:*

```
for(i=0; i<n; i=i*2)
      {
       for(j=i; j<n; j*j)
           k++;
      }
```

  NOTES:

Lecture 2

Data Structure

C++ Arrays: Operations, Complexities, and Copying

Dr. Abdul Hadi Mohammed

1. Introduction to Arrays in C++

An array in C++ is a collection of elements of the same data type, stored in contiguous memory locations. Arrays provide a way to store multiple items under a single name, with each element accessible through an index.

Basic syntax:

```
1. data_type array_name[array_size];
```

Example:

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

2. Initialization of array

2.1.Static Initialization

```
    int arr1[5] = {10, 20, 30, 40, 50}; // Static initialization
with explicit size
```

2.2. Default Initialization

```
int arr[5] = {}; // All elements initialized to 0
```

2.3. Partial Initialization

```
int arr[5] = {10, 20}; // First two elements initialized to 10
and 20, others are 0
```

2.4. Initialization with Implicit Size

```
int arr[] = {1, 2, 3, 4, 5}; // Array size is implicitly
determined by the initializer
```

When initializing arrays in C++, several types of errors can occur. Below are some common errors specifically related to array initialization:

1. **Too Many Initializers**

```
int arr[3] = {1, 2, 3, 4}; // Error: too many initializers for 'arr'
```

2. **Invalid Array Size**

```
int size = 5;
int arr[size]; // Error: variable size is not a constant expression
```

3. **Incomplete Type Error**

```
class MyClass; // Forward declaration
MyClass arr[5]; // Error: 'MyClass' is an incomplete type
```

4. **Mismatched Types**

```
int arr[3] = {1, 2, 3.5}; // Error: initializing with a float value in an int array
```

5. **Using Non-Constant Expression for Size**

```
int size = 5;
int arr[size]; // Error: variable size is not a constant expression
```

6. **Partial Initialization Without Clear Specification**

```
int arr[5] = {1, 2, 3}; // OK: rest elements initialized to 0
int arr2[5] = {1, 2, 3, 4, 5, 6}; // Error: too many initializers for 'arr2'
```

3. Array Operations and Their Complexities

3.1 Insertion

Insertion in an array depends on where you're inserting the element:

1. **Insertion at the beginnig** :

```
void insertAtBeginning(int arr[], int size, int element, int
capacity)
{
    if (size >= capacity)
    {
        cout << "Array is full, cannot insert." << endl;
        return;
    }
    for (int i = size; i > 0; i--)
    {
        arr[i] = arr[i - 1];
    }
    arr[0] = element;
    size++;
}
```

2. **Insertion at the end** (when there's space):

```
void insertAtEnd(int arr[], int size, int element, int capacity)
{
    if (size >= capacity)
    {
        cout << "Array is full, cannot insert." << endl;
        return;
```

```
    }
    arr[size] = element;
    ++size;
}
```

3. **Insertion at a specific index:**

```
void insertAt(int arr[], int size, int element, int index, int
capacity)
{
    if (size >= capacity)
    {
        cout << "Array is full, cannot insert." << endl;
        return;
    }
    if (index < 0 || index > size)
    {
        cout << "Invalid index." << endl;
        return;
    }
    for (int i = size; i > index; i--)
    {
        arr[i] = arr[i - 1];
    }
    arr[index] = element;
    size++;
}
```

3.2 Deletion

Similar to insertion, deletion complexity depends on the operation:

1. **Deletion from the beginnig**:

```
void deleteFromBeginning(int arr[], int size)
{
    if (size <= 0)
    {
        cout << "Array is empty, cannot delete." << endl;
        return;
    }
    for (int i = 0; i < size - 1; i++)
    {
        arr[i] = arr[i + 1];
    }
    size--;
}
```

1. **Deletion from the end**:

```
void deleteFromEnd(int arr[], int size)
{
    if (size <= 0)
    {
        cout << "Array is empty, cannot delete." << endl;
        return;
    }
    size--;
}
```

2. **Deletion from a specific index:**

```
void deleteAt(int arr[], int size, int index)
{
    if (size <= 0)
    {
        cout << "Array is empty, cannot delete." << endl;
        return;
    }
    if (index < 0 || index >= size)
    {
        cout << "Invalid index." << endl;
        return;
    }
    for (int i = index; i < size - 1; i++)
    {
        arr[i] = arr[i + 1];
    }
    size--;
}
```

3.3 Accessing Elements

Accessing elements in an array is straightforward:

```
int accessElement(const int arr[], int size, int index)
{
    if (index >= 0 && index < size)
    {
        return arr[index];
    }
    else
    {
        cout << "Index out of bounds." << endl;
        return -1; // Return an invalid value to indicate error
```

```
    }
}
```

3.4 Updating Elements

Updating elements is also simple:

```
void updateElement(int arr[], int size, int index, int element)
{
    if (index >= 0 && index < size)
    {
        arr[index] = element;
    }
    else
    {
        cout << "Index out of bounds." << endl;
    }
}
```

3.5 Print array

```
void printArray(const int arr[], int size)
{
    for (int i = 0; i < size; ++i)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}
```

4. Complexity Table

Operation	Description	Time Complexity	Notes
Insert at Beginning	Insert an element at the start of the array.	O(n)	Requires shifting all elements to the right.
Insert at End	Insert an element at the end of the array.	O(1)	Appends the element if there's space available.
Insert at Specific Index	Insert an element at a specified index.	O(n)	Requires shifting elements after the index.
Delete from Beginning	Remove the first element of the array.	O(n)	Requires shifting all elements to the left.
Delete from End	Remove the last element of the array.	O(1)	Simply decreases the size of the array.
Delete at Specific Index	Remove an element at a specified index.	O(n)	Requires shifting elements after the index.
Update Element	Update an element at a specific index.	O(1)	Direct access to the element.
Access Element	Access an element at a specific index.	O(1)	Direct access to the element.
Print Array	Print all elements in the array.	O(n)	Iterates through the array.