Arrays:

1) Static vs. dynamic arrays.
2) Find the Maximum and Minimum Elements in an Array,
3) Reverse an Array,
4) Find the Kth Smallest/Largest Element in an Array.

Static Vs. Dynamic Arrays

By Dr. GC Jana

---

1) Static Vs. Dynamic Arrays

Static Arrays

Definition:

- Static arrays are data structures with a fixed size. Once an array is created, its size cannot be altered. The memory for the array is allocated at compile time.

Key Features:

- **Fixed Size:** The size of the array is set during its declaration and cannot be changed.
- **Memory Allocation:** Memory is allocated on the stack, which is faster but has limited size.
- **Access Time:** Elements can be accessed in constant time, O(1)O(1)O(1), since the address of each element can be calculated using the base address and the index.
- **Usage:** Ideal for situations where the number of elements is known beforehand, such as storing fixed-size data like days of the week, months of the year, etc.

Advantages:

- Simple and easy to use.
- Fast access time.
- No dynamic memory management overhead.

Disadvantages:

- Wastes memory if the allocated size is not fully used.
- Inflexible, as the size cannot be changed once declared.

Dynamic Arrays

Definition:

- Dynamic arrays, also known as resizable arrays, can change size during runtime. Memory allocation happens on the heap, allowing the array to grow or shrink as needed.

Key Features:

- **Resizable:** The array can be resized, allowing it to grow or shrink based on the requirements.
- **Memory Allocation:** Memory is allocated on the heap, which is more flexible but can be slower due to the overhead of dynamic memory management.

- **Access Time:** Similar to static arrays, elements can be accessed in constant time, O(1)O(1)O(1), after they are created.
- **Usage:** Useful when the number of elements is not known in advance or when the array needs to expand or contract during program execution.

Advantages:

- Flexible size, which can be adjusted as needed.
- More efficient use of memory, as it only allocates what is necessary.

Disadvantages:

- Slower than static arrays due to the overhead of dynamic memory management.
- May cause memory fragmentation if not managed carefully.
- Requires additional memory for maintaining the array's capacity and size.

Summary

Feature	**Static Arrays**	**Dynamic Arrays**
Size	Fixed	Flexible/Resizable
Memory Allocation	Stack	Heap
Access Time	O(1)	O(1)
Memory Efficiency	Less efficient if size is not fully used	More efficient
Overhead	Low	Higher due to dynamic memory management
Use Cases	When size is known beforehand	When size may change or is not known

Java

Static Array Declaration

```
int[] staticArray = new int[5]; // Static array of size 5

staticArray[0] = 10; // Assigning value to the first element
```

Explanation: In Java, static arrays are created with a fixed size using the new keyword. The size of the array is specified when the array is initialized.

Dynamic Array Declaration (Using ArrayList)

```
import java.util.ArrayList;


ArrayList<Integer> dynamicArray = new ArrayList<>(); // Dynamic array
```

```
dynamicArray.add(10); // Adding an element to the array
```

Explanation: In Java, ArrayList is used to create dynamic arrays that can grow and shrink at runtime.

C++

Static Array Declaration

```
int staticArray[5]; // Static array of size 5
staticArray[0] = 10; // Assigning value to the first element
```

Explanation: In C++, static arrays are declared with a fixed size that cannot be changed during runtime.

Dynamic Array Declaration:

```
int* dynamicArray = new int[5]; // Dynamic array of size 5
dynamicArray[0] = 10; // Assigning value to the first element

// If you need to resize the array, you can do so by creating a new array
// and copying the elements from the old array (manually).
delete[] dynamicArray; // Don't forget to free the memory
```

Explanation: In C++, dynamic arrays are created using the new keyword. The memory is allocated on the heap, and the array size can be adjusted by reallocating memory.

Python

Static Array Declaration (Using Lists)

```
static_array = [0] * 5  # Static array of size 5
static_array[0] = 10  # Assigning value to the first element
```

Explanation: In Python, lists can be used to create static-like arrays by specifying a fixed size during initialization. However, Python lists are inherently dynamic.

Dynamic Array Declaration (Using Lists)

```
dynamic_array = []  # Dynamic array
dynamic_array.append(10)  # Adding an element to the array
```

Explanation: Python lists are dynamic by default, meaning their size can change during runtime. You can append or remove elements as needed.

Summary

Language	Static Array Declaration	Dynamic Array Declaration
Java	int[] arr = new int[5];	ArrayList<Integer> arr = new ArrayList<>();
C++	int arr[5];	int* arr = new int[5];
Python	arr = [0] * 5	arr = [];

Each language has its own way of handling arrays, with Java and C++ providing both static and dynamic options, while Python lists are inherently dynamic but can be initialized with a fixed size.

To find the maximum and minimum elements in an array, you can use the following algorithm:

Algorithm:

1. **Initialize** two variables: max_element and min_element with the first element of the array.
2. **Iterate** through the array:
    - For each element, compare it with max_element. If it's greater, update max_element.
    - Similarly, compare it with min_element. If it's smaller, update min_element.
3. **Return** max_element and min_element after the loop ends.

Python Code:

```python
def find_max_and_min(arr):
    # Initializing the maximum and minimum elements with the first element
    max_element = arr[0]
    min_element = arr[0]

    # Iterating through the array
    for num in arr[1:]:
        if num > max_element:
            max_element = num
        if num < min_element:
            min_element = num

    return max_element, min_element

# Example usage:
arr = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
max_element, min_element = find_max_and_min(arr)
print(f"Maximum element: {max_element}")
print(f"Minimum element: {min_element}")
```

Java Code:

```java
public class MaxMinArray {
    public static void main(String[] args) {
        int[] arr = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
        int maxElement = arr[0];
        int minElement = arr[0];

        for (int i = 1; i < arr.length; i++) {
            if (arr[i] > maxElement) {
                maxElement = arr[i];
            }
            if (arr[i] < minElement) {
                minElement = arr[i];
            }
        }

        System.out.println("Maximum element: " + maxElement);
        System.out.println("Minimum element: " + minElement);
    }
}
```

C++ Code:

```cpp
#include <iostream>
using namespace std;

int main() {
    int arr[] = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    int maxElement = arr[0];
    int minElement = arr[0];

    for (int i = 1; i < n; i++) {
        if (arr[i] > maxElement) {
            maxElement = arr[i];
        }
        if (arr[i] < minElement) {
            minElement = arr[i];
        }
    }

    cout << "Maximum element: " << maxElement << endl;
    cout << "Minimum element: " << minElement << endl;

    return 0;
}
```

Explanation:

1. **Initialization**:
    - Both programs initialize maxElement and minElement with the first element of the array.
2. **Iteration**:
    - They iterate through the array starting from the second element.
    - During each iteration, they compare the current element with maxElement and minElement, updating them if a larger or smaller element is found.
3. **Output**:
    - After completing the loop, both the maximum and minimum elements are printed.

Example Output:

For the array {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5}, the output would be:

Maximum element: 9

Minimum element: 1

This solution is efficient with a time complexity of O(n).

2. Reverse an array:

python

```python
def reverse_array(arr):
    start = 0
    end = len(arr) - 1

    while start < end:
        # Swap the elements at the start and end
        arr[start], arr[end] = arr[end], arr[start]
        start += 1
        end -= 1

    return arr

# Example usage:
arr = [1, 2, 3, 4, 5]
reversed_arr = reverse_array(arr)
print("Reversed array:", reversed_arr)
```

Java:

```java
public class ReverseArray {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        int start = 0;
        int end = arr.length - 1;

        while (start < end) {
            // Swap the elements at start and end
            int temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;

            start++;
            end--;
        }

        // Print the reversed array
        System.out.print("Reversed array: ");
        for (int i : arr) {
            System.out.print(i + " ");
        }
    }
}
```

C++:

```cpp
#include <iostream>
using namespace std;

void reverseArray(int arr[], int n) {
    int start = 0;
    int end = n - 1;

    while (start < end) {
        // Swap the elements at start and end
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;

        start++;
        end--;
    }
```

```
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    reverseArray(arr, n);

    // Print the reversed array
    cout << "Reversed array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }

    return 0;
}
```

Explanation:

1. **Initialization**:
    - The algorithms initialize two pointers: start at the beginning of the array and end at the end.
2. **Swapping**:
    - While start is less than end, the elements at these positions are swapped.
3. **Update Pointers**:
    - After each swap, start is incremented, and end is decremented.
4. **Termination**:
    - The loop terminates when start is no longer less than end, which means the array has been reversed.

Example Output:

For the array {1, 2, 3, 4, 5}, the output will be: 5 4 3 2 1

find the Kth smallest or largest element in an array,

Python: Using Sorting

```
def kth_smallest(arr, k):
    arr.sort()  # Sort the array
    return arr[k-1]  # Return the k-th smallest element

def kth_largest(arr, k):
    arr.sort(reverse=True)  # Sort the array in descending order
    return arr[k-1]  # Return the k-th largest element

# Example usage:
arr = [7, 10, 4, 3, 20, 15]
k = 3
print("3rd Smallest element:", kth_smallest(arr, k))
print("3rd Largest element:", kth_largest(arr, k))
```

Python: Quickselect

```
import random

def partition(arr, low, high):
    pivot = arr[high]
    i = low
    for j in range(low, high):
        if arr[j] <= pivot:
            arr[i], arr[j] = arr[j], arr[i]
            i += 1
    arr[i], arr[high] = arr[high], arr[i]
    return i

def quickselect(arr, low, high, k):
    if low == high:
        return arr[low]

    pivot_index = partition(arr, low, high)
```

```
    if k == pivot_index:
        return arr[k]
    elif k < pivot_index:
        return quickselect(arr, low, pivot_index - 1, k)
    else:
        return quickselect(arr, pivot_index + 1, high, k)

def kth_smallest(arr, k):
    return quickselect(arr, 0, len(arr) - 1, k - 1)

def kth_largest(arr, k):
    return quickselect(arr, 0, len(arr) - 1, len(arr) - k)

# Example usage:
arr = [7, 10, 4, 3, 20, 15]
k = 3
print("3rd Smallest element:", kth_smallest(arr, k))
print("3rd Largest element:", kth_largest(arr, k))
```

Java Code:

Using Sorting:

```
import java.util.Arrays;

public class KthElement {
    public static int kthSmallest(int[] arr, int k) {
        Arrays.sort(arr);  // Sort the array
        return arr[k - 1];  // Return the k-th smallest element
    }

    public static int kthLargest(int[] arr, int k) {
        Arrays.sort(arr);  // Sort the array
        return arr[arr.length - k];  // Return the k-th largest element
    }
```

```
    public static void main(String[] args) {
        int[] arr = {7, 10, 4, 3, 20, 15};
        int k = 3;
        System.out.println("3rd Smallest element: " + kthSmallest(arr, k));
        System.out.println("3rd Largest element: " + kthLargest(arr, k));
    }
}
```

C++ Code:

Using Sorting:

```
#include <iostream>
#include <algorithm>

using namespace std;

int kthSmallest(int arr[], int n, int k) {
    sort(arr, arr + n);  // Sort the array
    return arr[k - 1];  // Return the k-th smallest element
}

int kthLargest(int arr[], int n, int k) {
    sort(arr, arr + n, greater<int>());  // Sort the array in descending order
    return arr[k - 1];  // Return the k-th largest element
}

int main() {
    int arr[] = {7, 10, 4, 3, 20, 15};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    cout << "3rd Smallest element: " << kthSmallest(arr, n, k) << endl;
    cout << "3rd Largest element: " << kthLargest(arr, n, k) << endl;
    return 0;
}
```