L21:Finding the Smallest and Second Smallest Elements in an Array

Finding the Smallest Element

To find the smallest element in an array, follow these steps:

1. **Initialize a variable** `min` to store the minimum value.
2. **Iterate through the array** and update `min` whenever a smaller element is found.

```
int min = array[0]; // Assume the first element is the smallest
for (int i = 1; i < array.length; i++) {
    if (array[i] < min) {
        min = array[i]; // Update min if a smaller element is found
    }
}
System.out.println("The smallest element is: " + min);
```

Finding the Second Smallest Element (Naive Approach)

The naive approach involves sorting the array and picking the first two elements. This method, however, has a time complexity of O(n log n) due to sorting.

```
Arrays.sort(array);
int smallest = array[0];
int secondSmallest = array[1];
System.out.println("The smallest element is: " + smallest);
System.out.println("The second smallest element is: " + secondSmallest);
```

Efficient Approach with O(n) Time Complexity

To find the second smallest element efficiently:

1. **Initialize two variables** `min` and `secondMin` to store the smallest and second smallest elements, respectively.
2. **Iterate through the array** to find these elements in a single pass.

Step-by-Step Implementation:

1. **Initialize `min` and `secondMin`:**

   ```
   int min = Integer.MAX_VALUE;
   int secondMin = Integer.MAX_VALUE;
   ```

2. **Iterate through the array:**

   ```
   for (int i = 0; i < array.length; i++) {
       if (array[i] < min) {
           // Update secondMin before changing min
           secondMin = min;
           min = array[i];
       } else if (array[i] < secondMin && array[i] != min) {
           // Update secondMin if current element is not equal to min
           secondMin = array[i];
       }
   }
   ```

3. **Handle Edge Cases:**
   Ensure the array has at least two distinct elements.

Code Example

```
public class SecondSmallest {
    public static void main(String[] args) {
        int[] array = {2, 4, 1, 3, 5, -2, -4};

        if (array.length < 2) {
```

```
            System.out.println("Array must have at least tw
o elements.");
            return;
        }

        int min = Integer.MAX_VALUE;
        int secondMin = Integer.MAX_VALUE;

        for (int i = 0; i < array.length; i++) {
            if (array[i] < min) {
                secondMin = min;
                min = array[i];
            } else if (array[i] < secondMin && array[i] !=
min) {
                secondMin = array[i];
            }
        }

        if (secondMin == Integer.MAX_VALUE) {
            System.out.println("No second smallest element
found.");
        } else {
            System.out.println("The smallest element is: "
+ min);
            System.out.println("The second smallest element
is: " + secondMin);
        }
    }
}
```

Explanation of the Efficient Approach

- **Initialization:** We start with `min` and `secondMin` set to `Integer.MAX_VALUE` to ensure any element in the array will be smaller initially.
- **Single Pass:** We iterate through the array once. The first condition (`array[i] < min`) ensures that we always have the smallest element in `min`. The second condition (`array[i] < secondMin && array[i] != min`) ensures that `secondMin` is the smallest element greater than `min`.

- **Edge Cases:** If the array has fewer than two distinct elements, we handle it by checking the length and final values of `min` and `secondMin`.

Summary

- **Understanding Array Manipulation:** Finding specific elements using loops.
- **Importance of Time Complexity:** Reducing nested loops to optimize performance.
- **Hands-On Coding:** Implementing both naive and efficient solutions.