1. Unique Elements in an Array

Problem Statement:
Given an array of integers, remove duplicate elements and print the unique elements in sorted order.

Test Case:
Input:
arr = {4, 2, 7, 2, 4, 8, 1, 9, 7}

Output:
1 2 4 7 8 9

C++ Implementation using set

```
#include <iostream>
#include <set>
#include <vector>
using namespace std;

void printUniqueElements(const vector<int>& arr) {
   set<int> uniqueSet(arr.begin(), arr.end()); // Insert elements into set

   // Print elements in sorted order
   for (int num : uniqueSet) {
      cout << num << " ";
   }
   cout << endl;
}


int main() {
   vector<int> arr = {4, 2, 7, 2, 4, 8, 1, 9, 7};
   printUniqueElements(arr);
   return 0;
}
```

Output

```
1 2 4 7 8 9
```

Java Implementation using TreeSet

```
import java.util.*;

public class UniqueElements {
   public static void printUniqueElements(int[] arr) {
      TreeSet<Integer> uniqueSet = new TreeSet<>(); // TreeSet stores unique elements in sorted order

      // Insert elements into TreeSet
      for (int num : arr) {
          uniqueSet.add(num);
      }


      // Print elements in sorted order
      for (int num : uniqueSet) {
          System.out.print(num + " ");
      }
      System.out.println();
   }


   public static void main(String[] args) {
      int[] arr = {4, 2, 7, 2, 4, 8, 1, 9, 7};
      printUniqueElements(arr);
   }
}
```

Output

```
1 2 4 7 8 9
```

2. Check if an Element Exists in a Set

Problem Statement:

Given a set of integers and a query integer, determine if the integer is present in the set.

Test Case:

Input:

set = {3, 6, 9, 12, 15}
query = 9

Output:

Yes

```
#include <iostream>
#include <set>

using namespace std;
bool checkElement(const set<int>& s, int query) {
    return s.find(query) != s.end(); // Returns true if element exists, false otherwise
}

int main() {
    set<int> s = {3, 6, 9, 12, 15};
    int query = 9;

    if (checkElement(s, query)) {
        cout << "Yes\n" << endl;
    } else {
        cout << "No\n" << endl;
    }

    return 0;
}
```

Java Implementation using TreeSet

```
import java.util.*;

public class CheckElementInSet {
    public static boolean checkElement(TreeSet<Integer> set, int query) {
        return set.contains(query); // Returns true if element exists, false otherwise
    }

    public static void main(String[] args) {
        TreeSet<Integer> set = new TreeSet<>(Arrays.asList(3, 6, 9, 12, 15));
        int query = 9;

        if (checkElement(set, query)) {
            System.out.println("Yes");
        } else {
            System.out.println("No");
        }
    }
}
```

Example Code in C++ (unordered_set)

```
#include <iostream>
#include <unordered_set>

bool checkElement(const std::unordered_set<int>& s, int query) {
    return s.find(query) != s.end(); // O(1) avg case
}

int main() {
    unordered_set<int> s = {3, 6, 9, 12, 15};
    int query = 9;

    if (checkElement(s, query)) {
        std::cout << "Yes" << std::endl;
    } else {
        std::cout << "No" << std::endl;
    }

    return 0;
}
```

Output

```
Yes
```

Time Complexity

- **insert()** → O(1) avg, O(N) worst
- **erase()** → O(1) avg, O(N) worst
- **find()** → O(1) avg, O(N) worst

Java HashSet

A **HashSet** in Java is a hash table-based implementation that provides **constant-time operations (O(1) average case)** for inserting, deleting, and searching.

Example Code in Java (HashSet)

```java
import java.util.*;

public class HashSetExample {
    public static boolean checkElement(HashSet<Integer> set, int query) {
        return set.contains(query); // O(1) avg case
    }

    public static void main(String[] args) {
        HashSet<Integer> set = new HashSet<>(Arrays.asList(3, 6, 9, 12, 15));
        int query = 9;

        if (checkElement(set, query)) {
            System.out.println("Yes");
        } else {
            System.out.println("No");
        }
    }
}
```

Output

```
Yes
```

Time Complexity

- **add()** → O(1)
- **remove()** → O(1)
- **contains()** → O(1)

3. Find the Smallest and Largest Element in a Set

Problem Statement:

Given a set of integers, find and print the smallest and largest element.

Test Case:

Input:

set = {10, 20, 5, 30, 15}

Output:

```
Smallest: 5
Largest: 30
```

C++ Implementation using std::set

```cpp
#include <iostream>
```

```
#include <set>
using namespace std;
void findMinMax(const set<int>& s) {
    if (s.empty()) {
        cout << "Set is empty" << endl;
        return;
    }

    int smallest = *s.begin();   // First element (smallest)
    int largest = *s.rbegin();   // Last element (largest)

    cout << "Smallest: " << smallest << endl;
    cout << "Largest: " << largest << endl;
}

int main() {
    set<int> s = {10, 20, 5, 30, 15};

    findMinMax(s);

    return 0;
}
```

Output

```
Smallest: 5
Largest: 30
```

Java Implementation using TreeSet

```
import java.util.*;

public class FindMinMax {
    public static void findMinMax(TreeSet<Integer> set) {
        if (set.isEmpty()) {
            System.out.println("Set is empty");
            return;
        }

        int smallest = set.first();  // First element (smallest)
        int largest = set.last();    // Last element (largest)

        System.out.println("Smallest: " + smallest);
        System.out.println("Largest: " + largest);
    }

    public static void main(String[] args) {
        TreeSet<Integer> set = new TreeSet<>(Arrays.asList(10, 20, 5, 30, 15));

        findMinMax(set);
    }
}
```

Output

```
Smallest: 5
Largest: 30
```

4. Erase Elements Less Than a Given Value

Problem Statement:

Given a set of integers and a threshold x, remove all elements that are strictly less than x and print the modified set.

Test Case:

Input:

set = {5, 10, 15, 20, 25}
x = 15

Output:

```
15 20 25
```

```
#include <iostream>
#include <set>

using namespace std;
void eraseElementsLessThan(set<int>& s, int x) {
    auto it = s.lower_bound(x); // Find the first element >= x
    s.erase(s.begin(), it);     // Erase all elements before this iterator
}

void printSet(const set<int>& s) {
    for (int num : s) {
        cout << num << " ";
    }
    cout << endl;
}

int main() {
    set<int> s = {5, 10, 15, 20, 25};
    int x = 15;

    eraseElementsLessThan(s, x);
    printSet(s);

    return 0;
}
```

Output

```
15 20 25
```

Java Implementation using TreeSet

```
import java.util.*;

public class EraseElements {
    public static void eraseElementsLessThan(TreeSet<Integer> set, int x) {
        set.headSet(x).clear(); // Remove all elements less than x
    }

    public static void main(String[] args) {
        TreeSet<Integer> set = new TreeSet<>(Arrays.asList(5, 10, 15, 20, 25));
        int x = 15;

        eraseElementsLessThan(set, x);

        System.out.println(set);
    }
}
```

Output

```
[15, 20, 25]
```

5. Count Distinct Elements in a Range

Problem Statement:

Given a list of integers and a range [L, R], count the number of distinct elements present in the range.

Test Case:

Input:

arr = {4, 2, 2, 6, 4, 8, 10, 8, 6}
L = 2, R = 8

Output:

4 (distinct elements: {2, 4, 6, 8})

```
#include <iostream>
#include <set>
#include <vector>
using namespace std;
int countDistinctInRange(const vector<int>& arr, int L, int R) {
    set<int> uniqueElements;

    for (int num : arr) {
        if (num >= L && num <= R) {
            uniqueElements.insert(num);
        }
    }

    return uniqueElements.size();
}
```

```
int main() {
    vector<int> arr = {4, 2, 2, 6, 4, 8, 10, 8, 6};
    int L = 2, R = 8;

    cout << "Distinct count: " << countDistinctInRange(arr, L, R) << endl;

    return 0;
}
```

Output

```
Distinct count: 4
```

```
import java.util.*;
public class CountDistinctInRange {
    public static int countDistinctInRange(int[] arr, int L, int R) {
        TreeSet<Integer> uniqueElements = new TreeSet<>();

        for (int num : arr) {
            if (num >= L && num <= R) {
                uniqueElements.add(num);
            }
        }

        return uniqueElements.size();
    }

    public static void main(String[] args) {
        int[] arr = {4, 2, 2, 6, 4, 8, 10, 8, 6};
        int L = 2, R = 8;

        System.out.println("Distinct count: " + countDistinctInRange(arr, L, R));
    }
}
```

C++ Implementation using unordered_set

```
#include <iostream>
#include <unordered_set>
#include <vector>

using namespace std;
int countDistinctInRange(const vector<int>& arr, int L, int R) {
    unordered_set<int> uniqueElements;

    for (int num : arr) {
        if (num >= L && num <= R) {
            uniqueElements.insert(num); // O(1) average time complexity
        }
    }

    return uniqueElements.size();
}

int main() {
    vector<int> arr = {4, 2, 2, 6, 4, 8, 10, 8, 6};
    int L = 2, R = 8;

    cout << "Distinct count: " << countDistinctInRange(arr, L, R) << std::endl;

    return 0;
}
```

Time Complexity

- **O(N)** (since unordered_set insertions take **O(1) average** but **O(N) worst-case**

Java Implementation using HashSet

```
import java.util.*;

public class CountDistinctInRangeHashSet {
    public static int countDistinctInRange(int[] arr, int L, int R) {
        HashSet<Integer> uniqueElements = new HashSet<>();

        for (int num : arr) {
            if (num >= L && num <= R) {
                uniqueElements.add(num); // O(1) average time complexity
            }
        }

        return uniqueElements.size();
    }

    public static void main(String[] args) {
        int[] arr = {4, 2, 2, 6, 4, 8, 10, 8, 6};
        int L = 2, R = 8;

        System.out.println("Distinct count: " + countDistinctInRange(arr, L, R));
    }
}
```

Output

```
Distinct count: 4
```