How to sort a **HashMap** by key and value is one of the most important interview questions employers ask, and there is no single answer to it. We did not discuss this topic in the **HashMap** section because it requires some knowledge of **TreeMap**, which we have discussed in this section.

Let's discuss some of the ways to sort a **HashMap**.

Using a TreeMap

As we have already seen that the elements are stored in a **TreeMap** in sorted order by default, we can create a **TreeMap** and then add all the elements from our **HashMap** to the **TreeMap** using the `putAll()` method.

```java
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;

public class HashMapDemo {

    public static void main(String args[]) {

        Map<Integer, String> employeeMap = new HashMap<>();
        employeeMap.put(123, "Alex");
        employeeMap.put(342, "Ryan");
        employeeMap.put(143, "Joe");
        employeeMap.put(234, "Allen");
        employeeMap.put(432, "Roy");

        System.out.println("Unsorted map " + employeeMap);

        TreeMap<Integer, String> sortedMap = new TreeMap<>();
        sortedMap.putAll(employeeMap);
        System.out.println("Sorted map " + sortedMap);

    }
}

```

Using an ArrayList

We can store all the keys in an **ArrayList**, and then use the `sort()` method of the **Collections** class to sort the list. If we want to sort the values, then we can store the values in **ArrayList** and sort them.

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class HashMapDemo {

    public static void main(String args[]) {

        Map<Integer, String> employeeMap = new HashMap<>();
        employeeMap.put(123, "Alex");
        employeeMap.put(342, "Ryan");
        employeeMap.put(143, "Joe");
        employeeMap.put(234, "Allen");
        employeeMap.put(432, "Roy");

        List<Integer> keyList = new ArrayList<>(employeeMap.keySet());
        Collections.sort(keyList);
        System.out.println(keyList);


        List<String> valuesList = new ArrayList<>(employeeMap.values());
        Collections.sort(valuesList);
        System.out.println(valuesList);

    }
}
```

Using lambdas and streams

Java 8 introduced some methods to easily sort a **HashMap** by key or value. The **comparingByKey** comparator is used to sort the elements by key and **comparingByValue** comparator is used to sort the elements by value.

The below example shows how we can sort a **HashMap** by key or value.

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class HashMapDemo {

    public static void main(String args[]) {

        Map<Integer, String> employeeMap = new HashMap<>();
        employeeMap.put(123, "Alex");
        employeeMap.put(342, "Ryan");
        employeeMap.put(143, "Joe");
        employeeMap.put(234, "Allen");
        employeeMap.put(432, "Roy");

        System.out.println("Sorting by key");
        employeeMap.entrySet()
        .stream()
        .sorted(Map.Entry.<Integer, String>comparingByKey())
        .forEach(System.out::println);

        System.out.println("Sorting by value");
        employeeMap.entrySet()
        .stream()
        .sorted(Map.Entry.comparingByValue())
        .forEach(System.out::println);
```