Performance does NOT increase when multiple nodes are in one bucket.
It actually **decreases**.

Now let's break it properly like you'd explain in an interview.

1️⃣ How HashMap Works Internally

In Java (Java 8+):

HashMap uses:

Array of Buckets (Node<K,V>[])

Each bucket can contain:

- Single node
- Linked List (before Java 8)
- Linked List → converted to Red-Black Tree (after threshold)

Step-by-Step Working

When you do:

map.put("A", 10);

Step 1: Hash Calculation
int hash = key.hashCode();

Step 2: Index Calculation
index = hash % array.length

So bucket index is determined.

2️⃣ What Happens If Two Keys Go To Same Bucket?

This is called **Collision**

Example:

Let's say array size = 4

Key	hashCode	Index (hash % 4)
"A"	65	1
"E"	69	1

Both go to index 1.

Now bucket[1] becomes:

[A=10] → [E=20]

This is chaining (LinkedList).

3️⃣ Now The Interview Question:

If multiple nodes are in one bucket, how performance increases?

Correct Answer:

It DOES NOT increase.
It DEGRADES.

Because:

Normal case (No collision)

Time Complexity:

O(1)

Collision case (LinkedList)

If 5 elements in same bucket:

Search = O(n)

Because it has to iterate node by node.

4️⃣ Then How Has Java Improved Performance?

Java 8 introduced:

👉 If bucket size > 8 → convert LinkedList to Red-Black Tree.

Why?

LinkedList search:

O(n)

Red-Black Tree search:

O(log n)

So performance improves compared to LinkedList.

But still not better than O(1).

---

5 Visual Design Explanation (Interview Style)

Imagine:

Bucket Array (Size 4)

Index 0 → null
Index 1 → [A] → [E] → [I] → [M]
Index 2 → null
Index 3 → null

If 4 nodes in bucket 1:

Search "M"

Step 1: Go to index 1
Step 2: Compare A
Step 3: Compare E
Step 4: Compare I
Step 5: Compare M

Time = O(n)

After Tree conversion:

```
      E
     / \
    A   M
      /
     I
```

Search now:
O(log n)

Much better than O(n)

6️⃣ Final Interview Answer (Concise Version)

You can say:

> In HashMap, if multiple nodes are present in the same bucket due to hash collision, performance does not increase — it degrades from O(1) to O(n) in case of LinkedList traversal.
> To optimize this, Java 8 converts the bucket to a Red-Black Tree when the number of nodes exceeds 8, improving worst-case performance to O(log n).

Boom 💥 That's a strong answer.

7️⃣ Extra Pro-Level Point (To Impress Interviewer)

Mention:

- Default capacity = 16
- Load factor = 0.75
- Resize happens when:

  size > capacity * loadFactor
- Rehashing redistributes keys

That shows deep understanding.

Regards:

Srikanth Java