How HashMap Works? (Unordered Map in C++)

A **HashMap (unordered_map in C++)** is a data structure that provides **fast insertion, deletion, and lookup** using a **hash function**.

🔹 Key Concepts of HashMap

1. **Key-Value Pair Storage**
   - Each element is stored as `{key -> value}`.
   - Example: `{ "apple" -> 100, "banana" -> 50 }`.
2. **Hash Function (O(1) Access Time)**
   - A hash function **converts a key into an index** in an internal array.
   - This allows **O(1) average time complexity** for lookup.
3. **Collision Handling**
   - When two keys get mapped to the same index, **collision occurs**.
   - **Chaining (Linked List) or Open Addressing** is used to resolve collisions.
4. **Operations in HashMap**
   - **Insertion (`O(1)`)** → `map[key] = value;`
   - **Search (`O(1)`)** → `map.find(key);`
   - **Deletion (`O(1)`)** → `map.erase(key);`
   - **Traversal (`O(N)`)** → Using `for(auto x : map)`

🔹 C++ Implementation (unordered_map)

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

int main() {
    std::unordered_map<std::string, int> myMap;

    // Insert values
    myMap["apple"] = 100;
    myMap["banana"] = 50;
    myMap["cherry"] = 75;

    // Access value
    std::cout << "Apple Price: " << myMap["apple"] << std::endl;

    // Check if key exists
    if (myMap.find("banana") != myMap.end()) {
        std::cout << "Banana exists in map" << std::endl;
    }
    // Delete a key
    myMap.erase("cherry");

    // Iterate over map
    for (auto x : myMap) {
        std::cout << x.first << " -> " << x.second << std::endl;
    }

    return 0;
}
```

◇ Time Complexity of HashMap Operations

Operation	Average Case	Worst Case (Collision)
Insert	`O(1)`	`O(N)`
Search	`O(1)`	`O(N)`
Delete	`O(1)`	`O(N)`

◇ When to Use HashMap?

✔ **Fast lookups & insertions** → Searching, frequency counting.
✔ **Handling unique keys** → Finding duplicates, storing indices.
✔ **Solving problems in `O(N)` instead of `O(N²)`**.

Would you like an example of **collision handling** or **custom hash functions**? 🚀