**map vs unordered\_map in C++**

When it comes to efficiency, there is a huge difference between **maps** and **unordered maps**.  
**Difference :**

| map | unordered\_map

---------------------------------------------------------

Ordering | increasing order | no ordering

| (by default) |

----------------------------------------------------------

Implementation | Self balancing BST | Hash Table

| like [Red-Black Tree](https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/) |

----------------------------------------------------------

search time | log(n) | O(1) -> Average

| | O(n) -> Worst Case

----------------------------------------------------------

Insertion time | log(n) + Rebalance | Same as search

----------------------------------------------------------

Deletion time | log(n) + Rebalance | Same as search

**Use std::map when**

* You need ordered data.
* You would have to print/access the data (in sorted order).
* You need predecessor/successor of elements.

// CPP program to demonstrate use of std::map

#include <bits/stdc++.h>

Using namespace std;

int main()

{

// Ordered map

**map< int, int >** **order**;

// Mapping values to keys

**order**[5] = 10;

**order**[3] = 5;

**order**[20] = 100;

**order**[1] = 1;

// Iterating the map and printing ordered values

for (auto i = **order**. **begin()**; i != **order**. **end()**; i++)

{

cout << **i->first** << " : " << **i->second** << endl;

}

}

Output :

1 : 1

3 : 5

5 : 10

20 : 100

**Use std::unordered\_map when**

* You need to keep count of some data (Example – strings) and no ordering is required.
* You need single element access i.e. no traversal.

// CPP program to demonstrate unordered\_map

#include <bits/stdc++.h>

Using namespace std;

int main()

{

// Unordered map

**unordered\_map< int, int >order**

// Mapping values to keys

**order**[5] = 10;

**order**[3] = 5;

**order**[20] = 100;

**order**[1] = 1;

// Iterating the map and printing unordered values

for (auto i = **order**.**begin();**  i != **order**.**end();** i++)

{

cout << **i->first** << " : " << **i->second** << '\n';

}

}

Output :

1 : 1

3 : 5

20 : 100

5 : 10