

For quick reference, you can refer this curated list of ready-to-use data structures and methods.

## Vector

The most commonly used methods of vector are:

1. `push_back()`  
The run time of this method is  $O(1)$
2. `[]` (bracket operators)  
The run time of this method is  $O(1)$
3. `size()`  
The run time of this method is  $O(1)$

```
vector<int> v;           // v = {}
cout<< v.size() <<endl;  // outputs 0
v.push_back(20);        // v = {20}
v.push_back(10);        // v = {20, 10}
v.push_back(30);        // v = {20, 10, 30}
cout << v[1] << endl;    // outputs 10 (since, vector is zero-indexed)
cout << v.size() << endl; // outputs 3
```

## Set

Set stores its elements in sorted order and doesn't contain duplicate values.

The most commonly used methods of set are:

1. `insert()`  
The run time of this method is  $O(\log n)$
2. `find()`  
The run time of this method is  $O(\log n)$
3. `size()`  
The run time of this method is  $O(1)$

```
set<int> s;           // s = {}
cout<< s.size() <<endl; // outputs 0
s.insert(20);        // s = {20}
s.insert(10);        // s = {10, 20}
s.insert(10);        // s = {10, 20}
auto it = s.find(10); // it is an iterator that points to 10
cout << (it != s.end())? "FOUND" : "" << endl; // outputs FOUND
cout << s.size() << endl; // outputs 2
```

## Unordered\_Set



Unordered\_Set is same as set and has the same most common methods. The only difference is run time complexity.

The most commonly used methods of set are:

1. insert()  
The run time of this method is  $O(1)$
2. find()  
The run time of this method is  $O(1)$
3. size()  
The run time of this method is  $O(1)$

The unordered\_set() achieves this complexity because it does not keep it in sorted order.

```
unordered_set<int> s;           // s = {}
cout<< s.size() << endl;       // outputs 0
s.insert(20);                  // s = {20}
s.insert(10);                  // s = {10, 20}
s.insert(10);                  // s = {10, 20}
auto it = s.find(10);          // it is an iterator that points to 10
cout << (it != s.end())? "FOUND" : "" << endl;    // outputs FOUND
cout << s.size() << endl;       // outputs 2
```

## Map

Map is very similar to set, but instead of storing an element or a value, it stores a key and a value. The commonly used methods are:

1. insert()  
The run time of this method is  $O(\log n)$ , insertion in map is done using make\_pair
2. find()  
The run time of this method is  $O(\log n)$ , it returns pair of key and value to us.
3. size()  
The run time of this method is  $O(1)$
4. [] bracket operators  
The run time of this method is  $O(\log n)$ , if the key exists, then it returns reference to the value. If the key doesn't exist, then it will do an insertion in the map

```
map<int, int> m;                // m = {}
cout<< m.size() << endl;       // outputs 0
m.insert(make_pair(20, 1));     // m = {(20, 1)}
m.insert(make_pair(10,1));     // m = {(10, 1), (20, 1)}
m[10]++;                       // m = {(10, 2), (20, 1)}
auto it = m.find(10);           // it is an iterator that points to
(10, 2)
cout << (it != m.end() ? it -> second: 0) << endl; // outputs 2
```



```

auto it2 = m.find(20);           // it is an iterator that points to
(20, 1)
cout << (it2 != m.end() ? it2 -> first: 0) << endl; // outputs 20
cout << m.size() << endl;       // outputs 2

```

### Unordered\_Map

Unordered\_Map shares the same relationship with Map as unordered\_set shared with set. So, similarly the only difference lies in the run-time complexity and it is because map keeps key-value pairs in sorted order, while the unordered\_map keeps the key-value pair in any order. The commonly used methods are:

1. insert()  
The run time of this method is  $O(1)$ , insertion in map is done using make\_pair
2. find()  
The run time of this method is  $O(1)$ , it returns pair of key and value to us.
3. size()  
The run time of this method is  $O(1)$
4. [] bracket operators  
The run time of this method is  $O(1)$ , if the key exists, then it returns reference to the value. If the key doesn't exist, then it will do an insertion in the map

```

unordered_map<int, int> m;           // m = {}
cout << m.size() << endl;           // outputs 0
m.insert(make_pair(20, 1));         // m = {(20, 1)}
m.insert(make_pair(10, 1));         // m = {(10, 1), (20, 1)} (this
could be in any order)
m[10]++;                           // m = {(10, 2), (20, 1)} (this
could be in any order)
auto it = m.find(10);               // it is an iterator that points to
(10, 2)
cout << (it != m.end() ? it -> second: 0) << endl; // outputs 2
auto it2 = m.find(20);              // it is an iterator that points to
(20, 1)
cout << (it2 != m.end() ? it2 -> first: 0) << endl; // outputs 20
cout << m.size() << endl;           // outputs 2

```

