Set in c++ → Internally uses Red Black Tree

RANKA
DATE / /
PAGE (Self balancing BST)

- default order is always sorted
- to store in decreasing order use.
  (greater<int>)

→ set<int, greater<int>> s;
  s.insert(10);
  s.insert(5);
  s.insert(20);

→ if remove greater
5 10 20

Stored → 20 10 5

- ignores duplicate values.
- find is used to find the element.
  auto it = s.find(10);
- s.clear() → remove all the elements.
- count → returns count of occurrence (it returns 1/0 in case of sets).
- erase → removes element from set / group of elements from set
- lower_bound on sets →
- rbegin() → reverse iterator

s.erase(it) → erase that iterator
s.erase(it, s.end()) → from that iterator to last

→ gives the iterator to element if present or gives the element just greater than this

- upper_bound → member function → gives the iterator to the next greater element, which is not present
  eg 2 no. is greater than largest given end().

RANKA
DATE / /
PAGE

Complexities

Set

begin(), end()
rbegin(), rend()
cbegin(), cend()
size(), empty()
→ O(1)

insert, find(),
count, lower_bound()
upper_bound, erase
→ O(log n)

erase(it) → Amortized O(1)

Maps in C++ STL (Tree)

RANKA
DATE / /
PAGE

key - value

1 → abc
5 → cde

internal implementation → (Red Black Tree)

maps → values are store sorted acc. to key.

unordered_map → no values is sorted (Random order)

```
map<int, string> m
m[1] = "abc";
m[5] = "cde";
m.insert({1, "abc"});
```

print →

```
for(auto &pr:m)
{
cout<< pr.first<< " " << pr.second;
}
```

→ No duplicate key is allowed in maps.
→ insertion & accessing time complexity → log n

→ Maps are sorted by default (Acc. to key)
→ No d
→ If we access an item with m[value] it and it is not present it get inserted in maps.
→ m.at() → (gets access when key is present)

Maps in c++ STL (Uses Red Black Tree)

RANKA
DATE / /
PAGE

key - value.

1 → abc
5 → cde

internal implementation → (Red Black Tree)

maps → values are store · sorted acc. to key.

unordered_map ⇒ no values is sorted (Random Order)

```
→ map<int, string> m
   m[1] = "abc";
   m[5] = "cde";
   m.insert({1, "abc"});
```

print →

```
for(auto &pr:m)
{
    cout<< pr.first<< " " <<pr.second;
}
```

→ No duplicate key is allowed in maps.
→ insertion & accessing time complexity → log n
→ Maps are sorted by default (Acc. to key)
→ No d
→ If we access an item with m[value] st and it is not present it get inserted in maps.
→ m.at() → (gets access when key is present)

Unordered_map

→ Used to store key, value pairs
→ Uses Hashing
→ No order of keys.

RANKA
DATE / /
PAGE

Multimaps (Red Black tree)

diff. b/w maps & multimaps → we can have different keys with same value.

Square bracket is not available in multimap. (m[nums])