STL: C++ : part 3

Multiset() → sorted

```
multiset<int> ms;
ms.insert(1);      {1}
ms.insert(1);      {1,1}
ms.insert(1);      {1,1,1}
int count = ms.count(1) → // 3
ms.erase(1); //all 1s erased
// rest functions same as set.
```

```
// only erase one 1
ms.erase(ms.find(1));
```

{1,1,1} find first occurence (iterator)

```
ms.erase(ms.find(1),
         ms.find(1)+2) → {1,1,3
```

[start, end)

---

unordered set : randomised order — O(1)

unique elements but not sorted

```
unordered set<int> us;
```

(lower bound & upper bound does not work the same, rest all are same as set)
(mostly O(1) ; worst case : O(N))
(once in a blue moon)

#Map : sejal / Roll no: 24 / 25 / 26 } differentiator Roll no.

O(logN)

Key value

Key - unique

name → values → can be duplicates

```
map <int, int> mpp;
Map < int, pair <int, int>> mpp;
Map < pair <int, int> , int > mpp;
```

```
mpp[1] = 2;
mpp.emplace ({3,1});
mpp. insert ({2,4});
```

{1,2}
{2,4}
{3,1}

Map : unique keys in sorted order of key

```
map [{2,3}] = 10;
```

{{2,3},10}

```
for (auto it:mpp) {
    cout<< it.first<< it.second <<endl;
}
```
traverse a map

```
cout<< mpp [1] ; → 2
cout << mpp [5] ; → 0
```

```
auto it = mpp.find(3) ; → gives iterator
cout << *(it).second ; → ① of {3,13}
```

```
auto it = mpp.find(5); → mp.end()
```

```
auto.it = mpp.lower_bound(2) ;
          upper_bound(3) ;
```

erase, swap, size, empty are same

Multimap : can store duplicate keys but sorted

order.

---

unordered map : no duplicate, unique keys

not sorted

O(1) → O(N)

Majority once in Blue moon

Algorithms :

Sorting

a[] = {1, 5, 3, 2}

sort(a, a+4)

a[] = {1, 2, 3, 5}

sort (a, a+n);

sort (v.begin(), v.end());

for {vectors, arrays}

sort(a+2, a+4); ## {1, 5, 3, 2} → {1, 5, 2, 3}

sort(a, a+n, greater<int>); → descending order

My way

pair<int, int> a[] = {{1,2}, {2,1}, {4,1}};

// sort it acc. to increasing 2nd element.
// if 2nd element is same, sort acc. to descending 1st ele.

{2,1} {4,1} {1,2}
{4,1} {2,1} {1,2}

sort(a, a+n, comp)
↳ self written comparator : boolean func.
↓
Just pick up 2 pairs [eg : p1 & p2 & analyse]

bool comp(pair<int, int> p1, pair<int, int> p2){

if (p1.second < p2.second) return true;
if (p1.second > p2.second) return false;
// they are same

```
if (p1.first > p2.first) return true;
return false;
```

```
int num = 7;
int cnt = __builtin_popcount();
```

→ returns the no of set bits (1)

111 → 3, 110 → 2

```
long long num = 165786578687;
int cnt = __builtin_popcountll();
```

Next permutation:

```
string s = "123";
do {
    cout << s << endl;
} while (next_permutation(s.begin(), s.end()));
```

123
132
213
231
312
321
null

} all permutation

* if you want all permutation, start from sorted. else for eg: if s = 213 ⇒ 213 231 312 321 null

int maxe = * max_element (a, a+n); {1, 10, 5, 6}

int mine = *min_element (a, a+n); {1, 10, 5, 6}
