📘 [pb_ds Notes – ordered_set / ordered_multiset]

🧠 Add to Template:

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node_update> ordered_set;
typedef tree<int,null_type,less_equal<int>,rb_tree_tag,tree_order_statistics_node_update> ordered_multiset;
```

Functions

✅ 1. `order_of_key(x)`

```
s.order_of_key(x);
```

> *Returns the **number of elements strictly less than x**.*
> *Example:*

```
ordered_set s = {2, 4, 7};
s.order_of_key(6); // returns 2 (2 and 4 are < 6)
```

✅ 2. `find_by_order(k)`

```
*s.find_by_order(k);
```

> *Returns the **k-th smallest element** (0-based index).*
> *Example:*

```
ordered_set s = {10, 20, 30};
*s.find_by_order(1); // returns 20
```

🔷 Count of elements `< x`:

```
int count = s.order_of_key(x);
```

🔷 Count of elements `≤ x`:

```
int count = s.order_of_key(x + 1);
```

🔷 Count of elements in range `[L, R]`:

```
int count = s.order_of_key(R + 1) - s.order_of_key(L);
```

🔷 Check if element x exists:

```
if (s.find_by_order(s.order_of_key(x)) == s.end() || *s.find_by_order(s.order_of_key(x)) != x)
    cout << "Not found";
else
    cout << "Found";
```

🔷 Erase one instance of `x` in multiset:

```
auto it = s.lower_bound(x); // works for less_equal
if (it != s.end()) s.erase(it);
```

🔷 Get rank of value `x` (0-based):

```
int rank = s.order_of_key(x);
```

🔷 Get value with rank `k`:

```
int val = *s.find_by_order(k);
```

⚠ Notes:

- In `ordered_multiset`, `order_of_key(x)` returns number of elements strictly less than x, even if there are duplicates.
- `find_by_order(k)` returns the value at index `k` in the sorted structure.
- Erasing duplicates needs `lower_bound`, because `erase(x)` removes all occurrences.

✅ Use Cases:

Problem Type	Use Tool	Complexity
Count elements < x	`order_of_key`	O(log n)
K-th smallest element	`find_by_order`	O(log n)
Dynamic rank tracking	combo	O(log n)
Count in range [L, R]	`order_of_key`	O(log n)
Online inversion count	`order_of_key`	O(n log n)
Real-time sorted insert + query	both	O(log n)

Custom Multiset

// ⚠ Works only for small positive values (≤ 1e6 or so) (or use compression)
// Supports: insert, erase, count, kth smallest, order_of_key
// Based on Binary Indexed Tree (Fenwick Tree)

```
class Multiset {
private:
  //use with the positive number only and limited
  // so if you have a big or non-positive number you have to compress them
  vector<int> Bit;
  int SZ, size_set;

  void add(int pos, int val) {
    for (++pos; pos <= SZ; pos += pos & -pos)
      Bit[pos - 1] += val;
  }

  int get(int pos) {
    int ret = 0;
    for (++pos; pos; pos -= pos & -pos)
      ret += Bit[pos - 1];
    return ret;
  }

  int BS(int val) {
    int s = 0;
    for (int sz = SZ >> 1; sz; sz >>= 1) {
      if (Bit[s + sz - 1] < val)
        val -= Bit[(s += sz) - 1];
    }
    return s;
  }
public:
  Multiset() : size_set(0), SZ(1 << 20) {
    Bit.resize(SZ);
    add(0, -1);
  }

  void insert(int val) {
    ++size_set;
    add(val, 1);
  }

  int count(int val) {
    return get(val) - get(val - 1);
  }

  /// erase all occurrence of val in the multiset
  void erase_all(int val) {
    int c = count(val);
    size_set -= c;
    add(val, -c);
  }

  void erase_idx(int index) {
    --size_set;
    add(BS(index), -1);
  }

  int order_of_key(int val) {//get_val_idx //freq(numbers)<val
    return get(val) - count(val) + 1;
  }

  int operator[](int index) { return BS(index); }

  int size() const { return size_set; }

};
```