```
    st.emplace(2); // {1, 2}
    st.insert(2); // {1, 2}
    st.insert(4); // {1, 2, 4}
    st.insert(3); // {1, 2, 3, 4}

    // Functionality of insert in vector
    // can be used also, that only increases
    // efficiency

    // begin(), end(), rbegin(), rend(), size(),
    // empty() and swap() are same as those of above

    // {1, 2, 3, 4, 5}
    auto it = st.find(3);

    // {1, 2, 3, 4, 5}
    auto it = st.find(6);

    // {1, 4, 5}
    st.erase(5); // erases 5 // takes logarithmic time

    int cnt = st.count(1);

    auto it = st.find(3);
    st.erase(it); // it takes constant time

    // {1, 2, 3, 4, 5}
    auto it1 = st.find(2);
    auto it2 = st.find(4);
    st.erase(it1, it2); // after erase {1, 4, 5} [first, last)

    // lower_bound() and upper_bound() function works in the same way
```

set => unique sorted

erase (el)

erase (iter

1

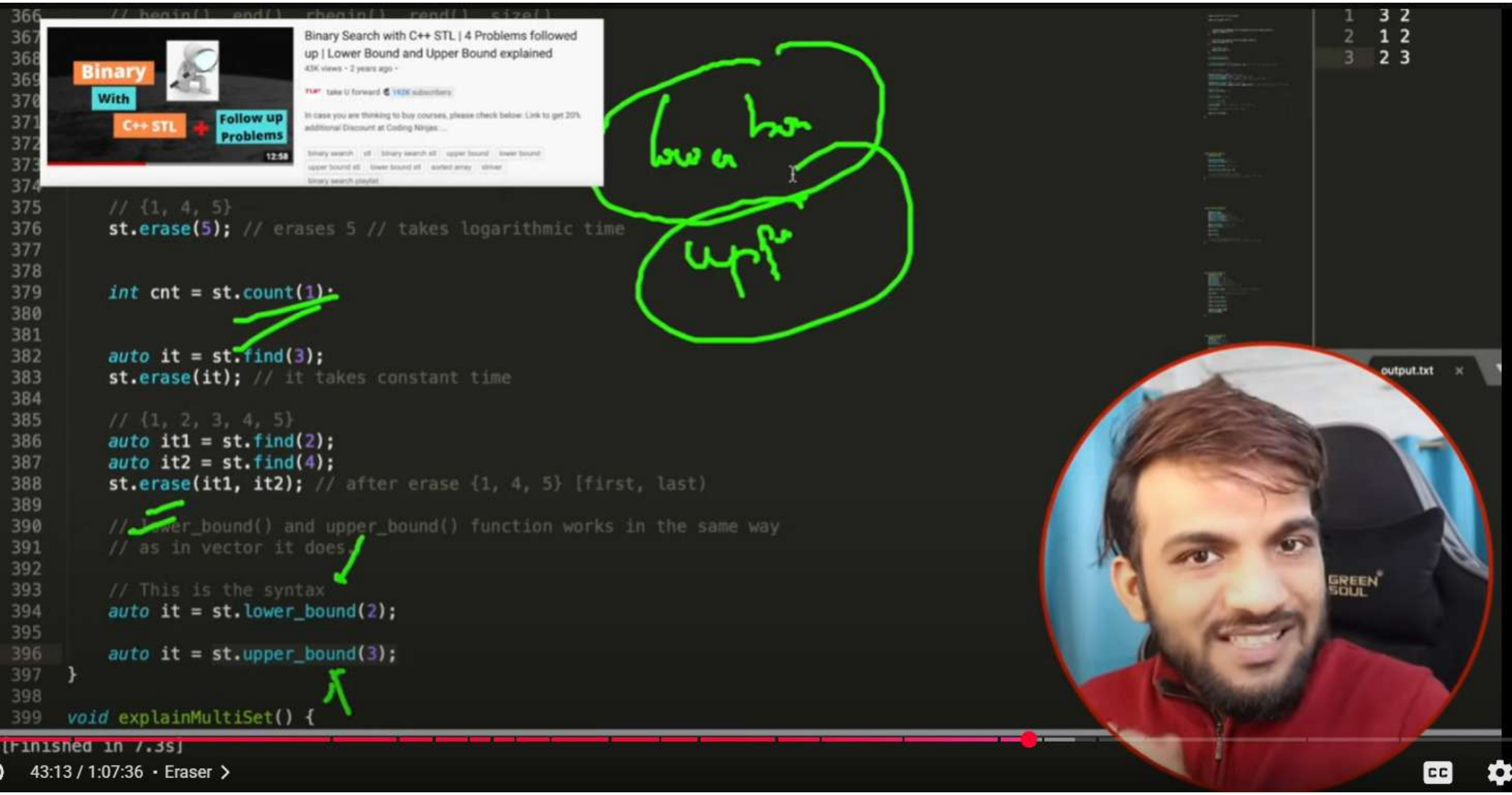
0

```
1 3 2
2 1 2
3 2 3
```

output.txt

[Finished in 7.3s]

42:05 / 1:07:36 • Set >

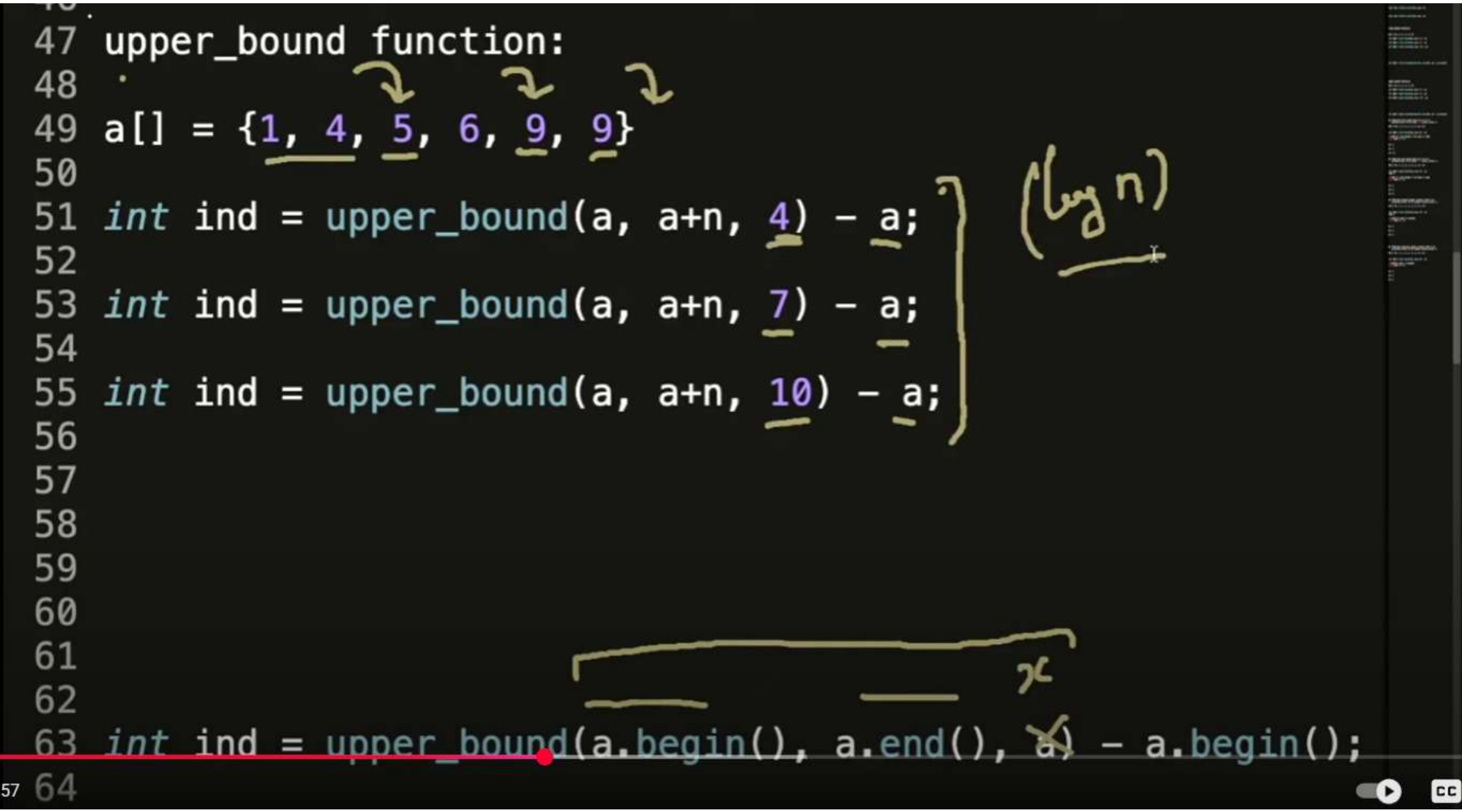

Binary
With
C++ STL
Follow up
Problems
12:58
Binary Search with C++ STL | 4 Problems followed up | Lower Bound and Upper Bound explained
43K views • 2 years ago •
take U forward
In case you are thinking to buy courses, please check below: Link to get 20% additional Discount at Coding Ninjas: ...
binary search
stl
binary search stl
upper bound
lower bound
upper bound stl
lower bound stl
sorted array
striver
binary search playlist
// {1, 4, 5}
st.erase(5); // erases 5 // takes logarithmic time
int cnt = st.count(1);
auto it = st.find(3);
st.erase(it); // it takes constant time
// {1, 2, 3, 4, 5}
auto it1 = st.find(2);
auto it2 = st.find(4);
st.erase(it1, it2); // after erase {1, 4, 5} [first, last)
// lower_bound() and upper_bound() function works in the same way
// as in vector it does
// This is the syntax
auto it = st.lower_bound(2);
auto it = st.upper_bound(3);
}
void explainMultiSet() {
1 3 2
2 1 2
3 2 3
output.txt
[Finished in 7.3s]
43:13 / 1:07:36 • Eraser >

BINARY SEARCH

Q1. Check if X exists in the sorted array or not?

```
A[] = {1, 4, 5, 8, 9}

bool res = binary_search(a, a+n, 3);


bool res = binary_search(a, a+n, 4);
```

```
21 lower_bound function:
22
23
24 a[] = {1, 4, 5, 6, 9, 9}
25
26 int ind = lower_bound(a, a+n, 4) - a;
27
28 int ind = lower_bound(a, a+n, 7) - a;
29
30 int ind = lower_bound(a, a+n, 10) - a;
31
32
33
34
35
36
37
38 int ind = lower_bound(a.begin(), a.end(), x) - a.begin();
```


upper_bound function:
a[] = {1, 4, 5, 6, 9, 9}
int ind = upper_bound(a, a+n, 4) - a;
int ind = upper_bound(a, a+n, 7) - a;
int ind = upper_bound(a, a+n, 10) - a;
(log n)
x
int ind = upper_bound(a.begin(), a.end(), x) - a.begin();

```
}


void explainMultiSet() {
    // Everything is same as set
    // only stores duplicate elements also


    multiset<int>ms;
    ms.insert(1); // {1}
    ms.insert(1); // {1, 1}
    ms.insert(1); // {1, 1, 1}

    ms.erase(1); // all 1's erased

    int cnt = ms.count(1);

    // only a single one erased
    ms.erase(ms.find(1));

    ms.erase(ms.find(1), ms.find(1)+2);

    // rest all function same as set
}
```

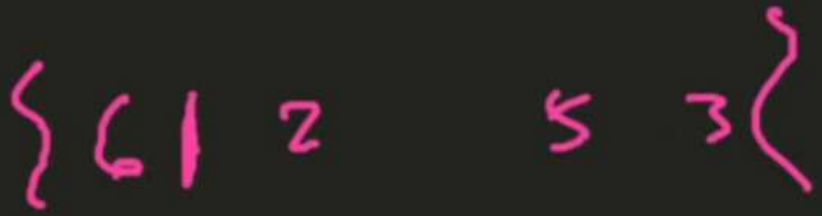
```
1 3 2
2 1 2
3 2 3
```

output.txt

[Finished in 7.3s]

:26 / 1:07:36 · Multiset >

```
427
428
429
430
431
432
433
434 void explainUSet() {
435     unordered_set<int> st;
436     // lower_bound and upper_bound function
437     // does not works, rest all functions are same
438     // as above,  it does not stores in any
439     // particular order it has a better complexity
440     // than set in most cases, except some when collision happens
441 }
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
```



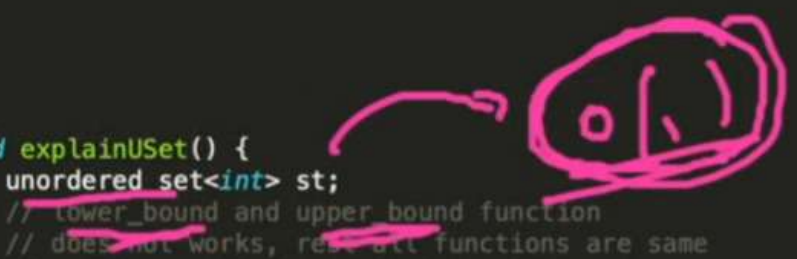

```
1  3 2
2  1 2
3  2 3
```

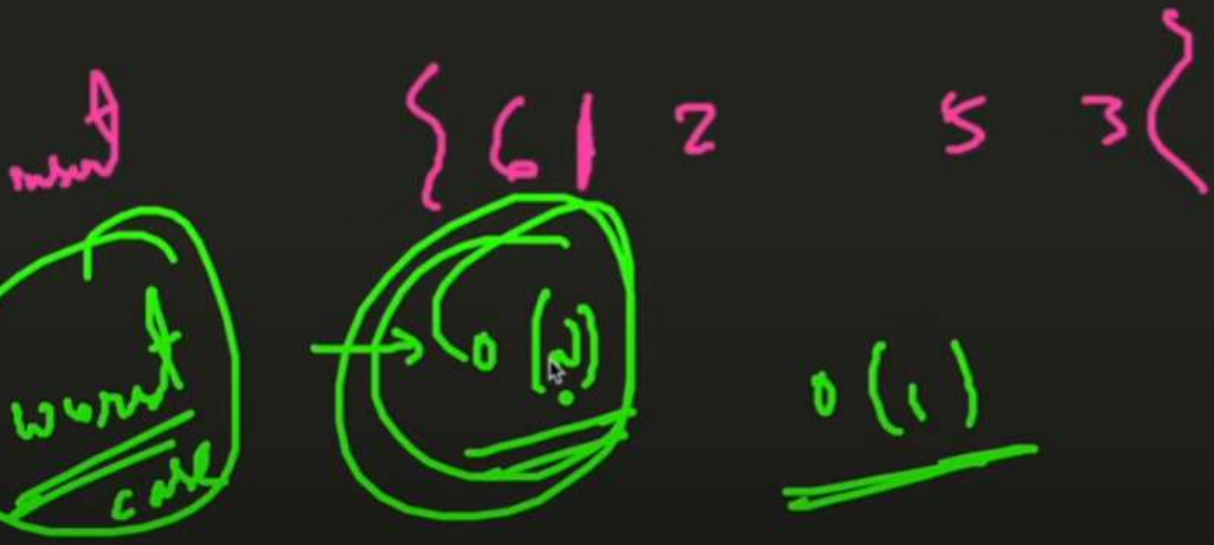
output.txt

[Finished in 7.3s]

6:14 / 1:07:36 • Unordered Set >

```
427
428
429
430
431
432
433
434 void explainUSet() {
435     unordered_set<int> st;
436     // lower_bound and upper_bound function
437     // does not works, rest all functions are same
438     // as above,  it does not stores in any
439     // particular order it has a better complexity
440     // than set in most cases, except some when collision happens
441 }
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
```






```
1  3 2
2  1 2
3  2 3
```

output.txt

[Finished in 7.3s]

08 / 1:07:36 • Unordered Set >

```cpp
void explainMap() {

    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});

    mpp[{2,3}] = 10;

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

    for(auto it : mpp) {
        cout << it.first << " " << it.second << endl;
    }


    cout << mpp[1];
    cout << mpp[5];

```

{key, value}

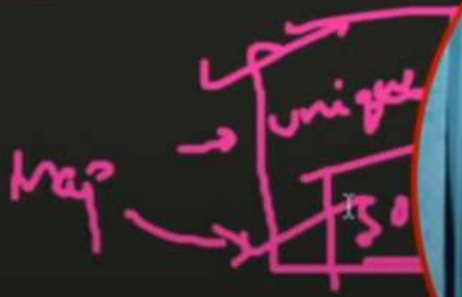
ds int double pair

```
1 3 2
2 1 2
3 2 3
```


```
[Finished in 7.3s]
```

:10 / 1:07:36 · Map >

```cpp
void explainMap() {

    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});

    mpp[{2,3}] = 10;

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

    for(auto it : mpp) {
        cout << it.first << " " << it.second << endl;
    }


    cout << mpp[1];
    cout << mpp[5];

}
```

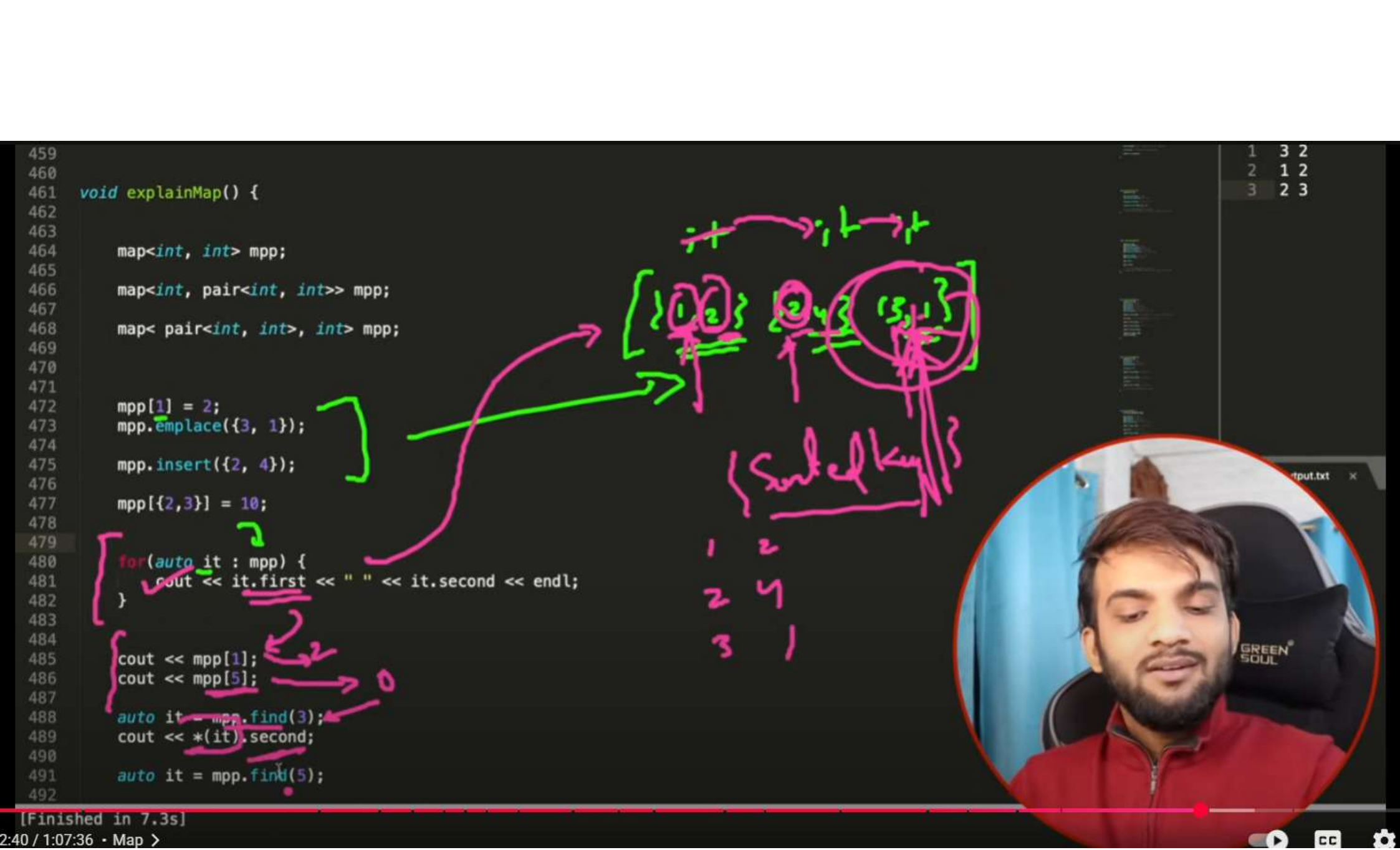


```
[Finished in 7.3s]
```

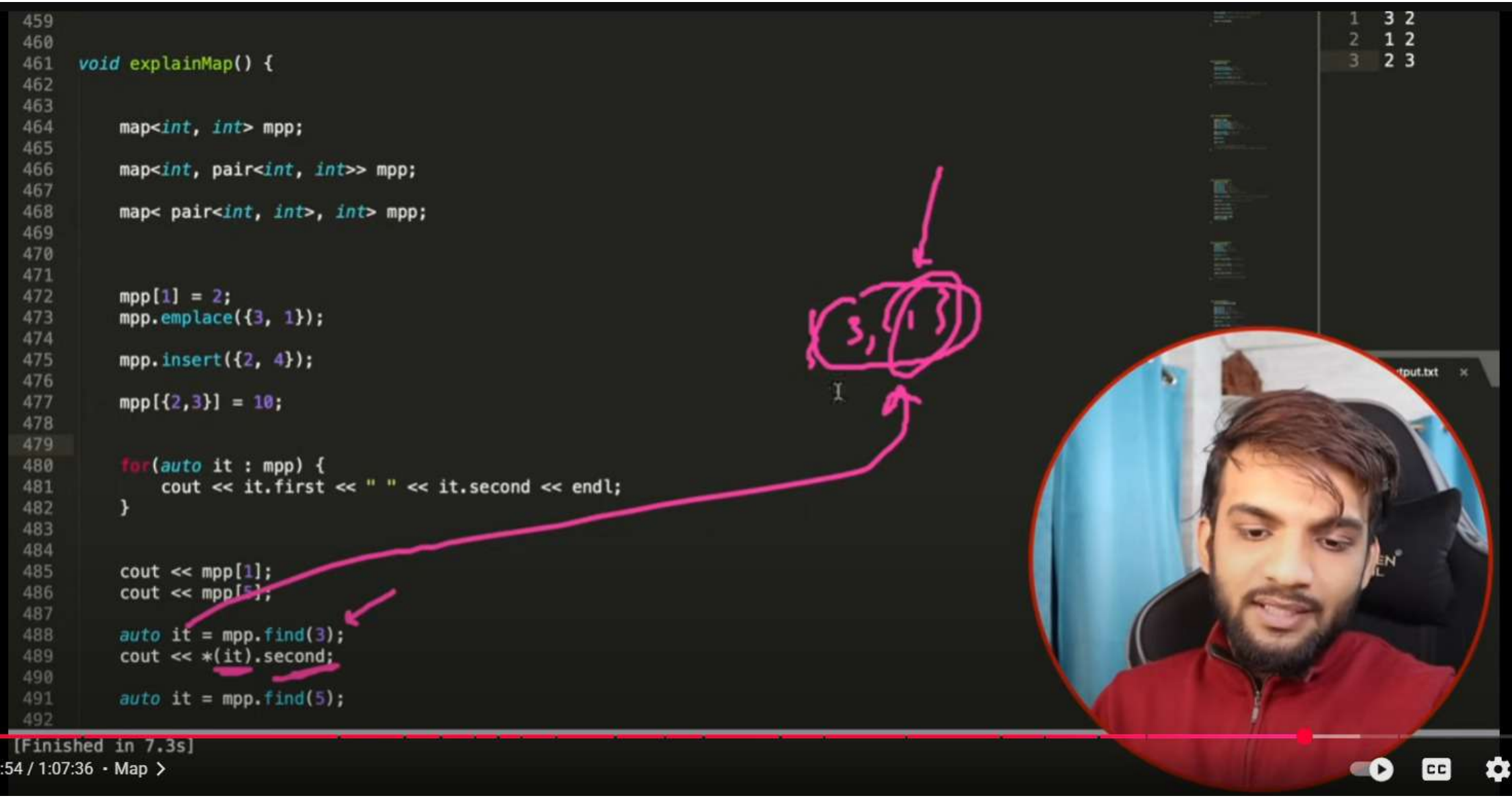
8 / 1:07:36 • Map >

```cpp
void explainMap() {

    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});

    mpp[{2,3}] = 10;

    for(auto it : mpp) {
        cout << it.first << " " << it.second << endl;
    }

    cout << mpp[1];
    cout << mpp[5];

    auto it = mpp.find(3);
    cout << *(it).second;

    auto it = mpp.find(5);

[Finished in 7.3s]
```

```
1 3 2
2 1 2
3 2 3
```



2:40 / 1:07:36 • Map >


459
460
461 void explainMap() {
462
463
464 map<int, int> mpp;
465
466 map<int, pair<int, int>> mpp;
467
468 map< pair<int, int>, int> mpp;
469
470
471
472 mpp[1] = 2;
473 mpp.emplace({3, 1});
474
475 mpp.insert({2, 4});
476
477 mpp[{2,3}] = 10;
478
479
480 for(auto it : mpp) {
481 cout << it.first << " " << it.second << endl;
482 }
483
484
485 cout << mpp[1];
486 cout << mpp[5];
487
488 auto it = mpp.find(3);
489 cout << *(it).second;
490
491 auto it = mpp.find(5);
492
1 3 2
2 1 2
3 2 3
[Finished in 7.3s]
:54 / 1:07:36 • Map >

```
494     auto it = mpp.lower_bound(2);
495
496     auto it = mpp.upper_bound(3);
497
498     // erase, swap, size, empty, are same as above
499
500 }
501
502
503 void explainMultimap() {
504     // everything same as map, only it can store multiple keys
505     // only mpp[key] cannot be used here
506
507 }
508
509
510 void explainUnorderedMap() {
511     // same as set and unordered_Set difference.
512 }
513
514 bool comp(pair<int,int>p1, pair<int,int>p2) {
515     if(p1.second < p2.second) {
516         return true;
517     } else if(p1.second == p2.second){
518         if(p1.first>p2.second) return true;
519     }
520     return false;
521 }
522
523 void explainExtra() {
524
525     sort(a+2, a+4);
526
527     sort(a, a+n, greater<int>);
```

duplicate keys

sorted

{1, 2}

{1, 3}

```
2 1 2
3 2 3
```

[Finished in 7.3s]

42 / 1:07:36 · Multimap >

```
494     auto it = mpp.lower_bound(2);
495
496     auto it = mpp.upper_bound(3);
497
498     // erase, swap, size, empty, are same as above
499
500 }
501
502
503 void explainMultimap() {
504     // everything same as map, only it can store multiple keys
505     // only mpp[key] cannot be used here
506
507 }
508
509
510 void explainUnorderedMap() {
511     // same as set and unordered_Set difference.
512 }
513
514 bool comp(pair<int,int>p1, pair<int,int>p2) {
515     if(p1.second < p2.second) {
516         return true;
517     } else if(p1.second == p2.second){
518         if(p1.first>p2.second) return true;
519     }
520     return false;
521 }
522
523 void explainExtra() {
524
525     sort(a+2, a+4);
526
527     sort(a, a+n, greater<int>);
```

duplicate keys

{1, 2}
{1, 3}

1 2
2 1 2
3 2 3

[Finished in 7.3s]

55 / 1:07:36 • Multimap >

```
494     auto it = mpp.lower_bound(2);
495
496     auto it = mpp.upper_bound(3);
497
498     // erase, swap, size, empty, are same as above
499
500 }
501
502
503 void explainMultimap() {
504     // everything same as map, only it can store multiple keys
505     // only mpp[key] cannot be used here
506
507 }
508
509
510 void explainUnorderedMap() {
511     // same as set and unordered_set difference.
512 }
513
514 bool comp(pair<int,int>p1, pair<int,int>p2) {
515     if(p1.second < p2.second) {
516         return true;
517     } else if(p1.second == p2.second){
518         if(p1.first>p2.second) return true;
519     }
520     return false;
521 }
522
523 void explainExtra() {
524
525     sort(a+2, a+4);
526
527     sort(a, a+n, greater<int>);
```

```
1  3 2
2  1 2
3  2 3
```

[Finished in 7.3s]

22 / 1:07:36 • Multimap >

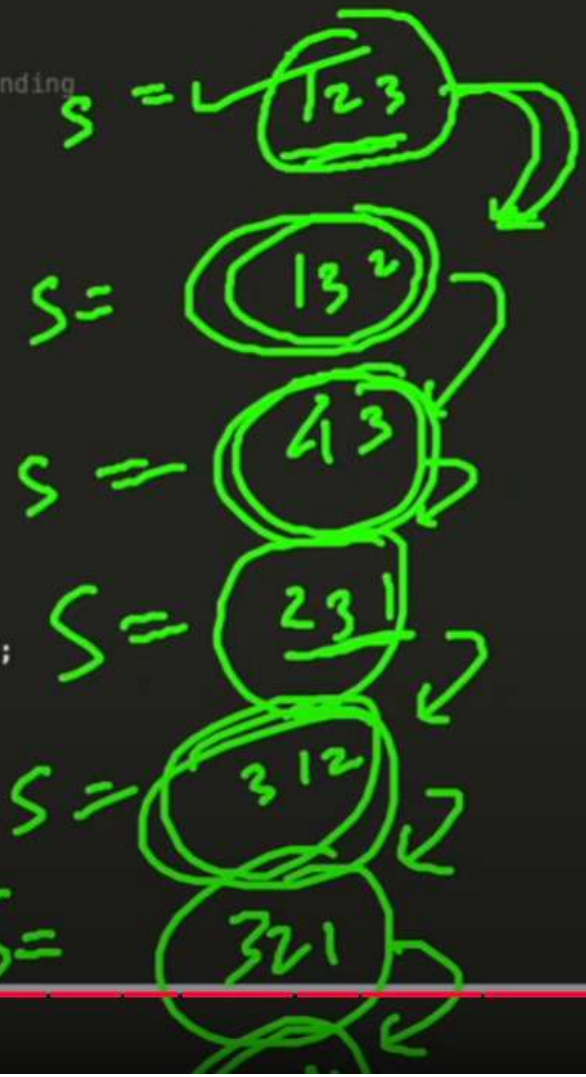
```cpp
bool comp(pair<int,int> p1, pair<int,int> p2) {
    if(p1.second < p2.second) return true;
    if(p1.second > p2.second) return false;
    // they are same

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

void explainExtra() {


    sort(a, a + n);
    sort(v.begin(), v.end());

    sort(a+2, a+4);

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

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

    // sort it according to second element
    // if second element is same, then sort
    // it according to first element but in descending

    sort(a, a+n ,comp);

    // {4,1}, {2, 1}, {1, 2}};


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

```
3 2
1 2
2 3
```

```
[Finished in 7.3s]
```

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

    // sort it according to second element
    // if second element is same, then sort
    // it according to first element but in descending

    sort(a, a+n ,comp);

    // {4,1}, {2, 1}, {1, 2}};


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

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


    string s = "123";

    do {
        cout << s << endl;
    } while(next_permutation(s.begin(), s.end()));

    int maxi = *max_element(a,a+n);
}


int main() {

    return 0;
}

[Finished in 7.3s]
```

```
1  3 2
2  1 2
3  2 3
```



05:38 / 1:07:36 · Sorting in descending order >

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

    // sort it according to second element
    // if second element is same, then sort
    // it according to first element but in descending

    sort(a, a+n ,comp);

    // {4,1}, {2, 1}, {1, 2}};


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

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


    string s = "123";
    sort(s.begin(), s.end());

    do {
        cout << s << endl;
    } while(next_permutation(s.begin(), s.end()));

    int maxi = *max_element(a,a+n);
}


int main() {

    return 0;
}
```

```
1  3 2
2  1 2
3  2 3
```

```
[Finished in 7.3s]
```

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

    // sort it according to second element
    // if second element is same, then sort
    // it according to first element but in descending

    sort(a, a+n ,comp);

    // {4,1}, {2, 1}, {1, 2}};


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

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


    string s = "123";
    sort(s.begin(), s.end());

    do {
        cout << s << endl;
    } while(next_permutation(s.begin(), s.end()));

    int maxi = *max_element(a,a+n);
}


int main() {

    return 0;
}
```

```
1  3 2
2  1 2
3  2 3
```

```
[Finished in 7.3s]
```