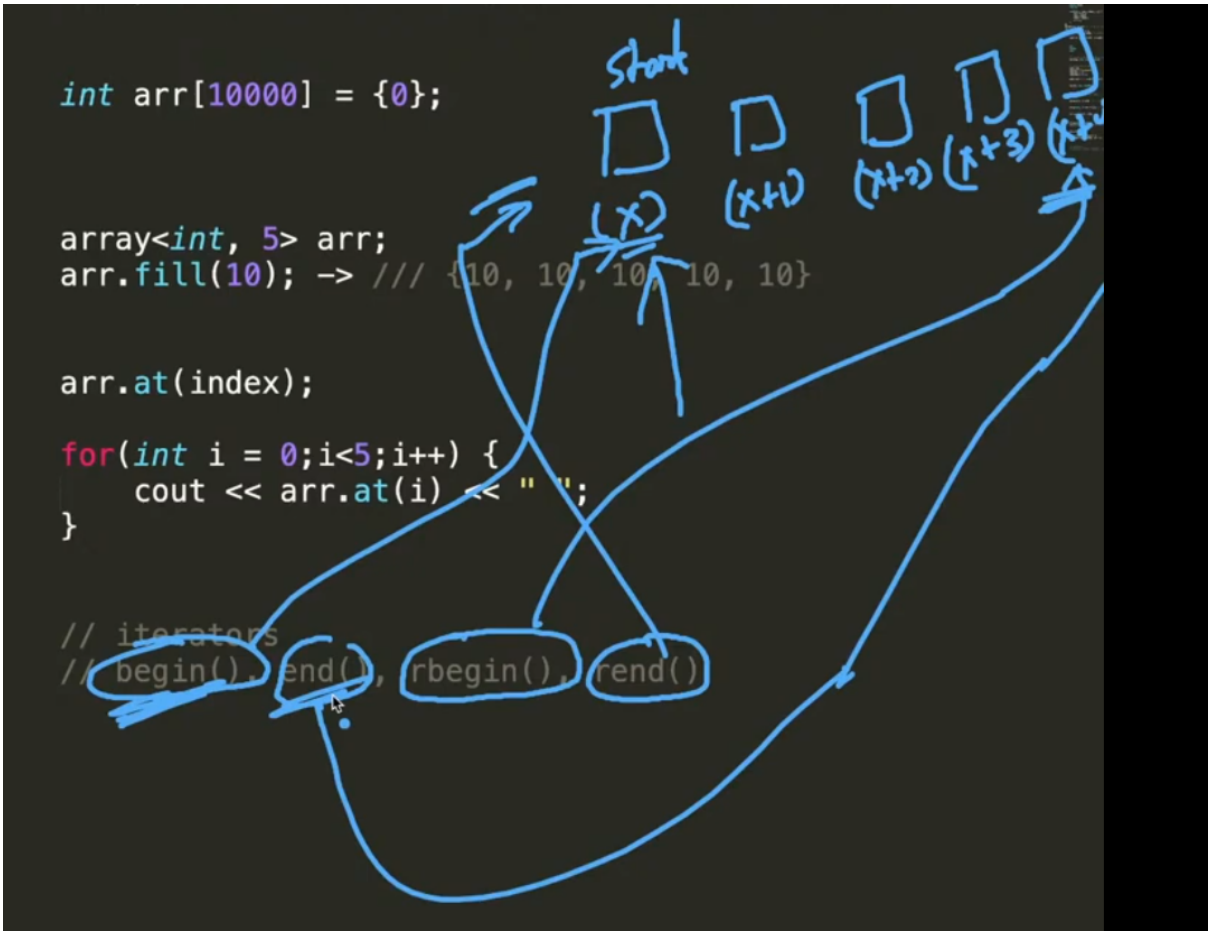
```cpp
47
48      // Arrays -> int arr[100];
49
50
51      array<int, 3> arr;  // -> {?, ?, ?}
52
53
54      array<int, 5> arr = {1}; // -> {1, 0, 0, 0, 0}
55
56
57      int arr[10000] = {0};
58
59
60
61      array<int, 5> arr;
62      arr.fill(10); -> /// {10, 10, 10, 10, 10}
63
64
65      arr.at(index);
66
67      for(int i = 0;i<5;i++) {
68          cout << arr.at(i) << " ";
69      }
70
71
```



```cpp
int arr[10000] = {0};

array<int, 5> arr;
arr.fill(10); -> /// {10, 10, 10, 10, 10}

arr.at(index);

for(int i = 0;i<5;i++) {
    cout << arr.at(i) << " ";
}

// iterators
// begin(), end(), rbegin(), rend()

//

array<int, 5> arr = {1, 3, 4, 5, 6};
for(auto it: arr.begin(); it!=arr.end();it++) {
    cout << *it << " ";
}
```

```cpp
// iterators
// begin(), end(), rbegin(), rend()

//

array<int, 5> arr = {1, 3, 4, 5, 6};
for(auto it: arr.begin(); it!=arr.end();it++) {
    cout << *it << " ";
}

for(auto it: arr.rbegin(); it>arr.rend();it++) {
    cout << *it << " ";
}

for(auto it: arr.end() - 1; it>=arr.begin();it--) {
    cout << *it << " ";
}

// for each loop
for(auto it: arr) {
    cout << it << " ";
}
```

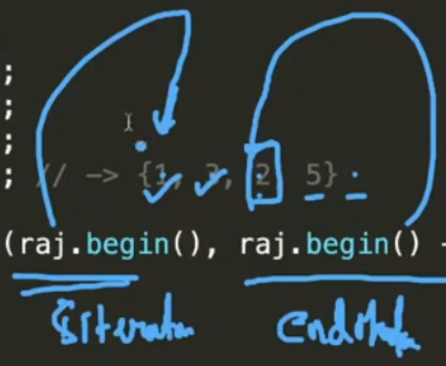
```
96
97      string s = "xhegcwe";
98      // x h e g c w e
99      for(auto c:s) {
100         cout << c << " ";
101     }
102
103
104     // size
105     cout << arr.size();
106
107     // front
108     cout << arr.front(); // arr.at(0);
109
110     // back
111     cout << arr.back(); // arr.at(arr.size() - 1);
112
113
```

```
// max size of 10^7 -> int, double, char
int arr[10000000];

// max size of 10^8 -> bool
    bool arr[100000000];
int main() {

    // max size of 10^6 -> int, double, char
    int arr[1000000];


    // max size of 10^7 -> bool
    bool arr[10000000];
```

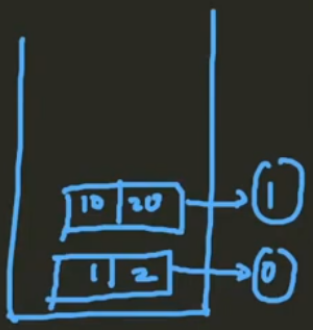
```cpp
// VECTOR

int arr[50];

// segmentation fault if you push_back 10^7 times

vector<int> arr; // -> {}
cout << arr.size() << endl; // -> print 0
arr.push_back(0); // {0}
arr.push_back(2); // {0,2}
cout << arr.size() << endl; // -> print 2
arr.pop_back(); // {0}
cout << arr.size() << endl; // print 1

arr.push_back(0); // {0,0}
arr.push_back(2); // {0,0,2}


vec.clear(); // --> erase all elements at once {}
```

```cpp
vector<int> vec1(4, 0); // -> {0,0,0,0}
vector<int> vec2(4, 10); // -> {10,10,10,10}

// copy the entire vec2 into vec3
vector<int> vec3(vec2.begin(), vec2.end()); // -> []
vector<int> vec3(vec2);



vector<int> raj;
raj.push_back(1); // raj.emplace_back(1); // emplace_back
raj.push_back(3);
raj.push_back(2);
raj.push_back(5); // -> {1, 3, 2, 5}

vector<int> raj1(raj.begin(), raj.begin() + 2);  // -> {1,
```

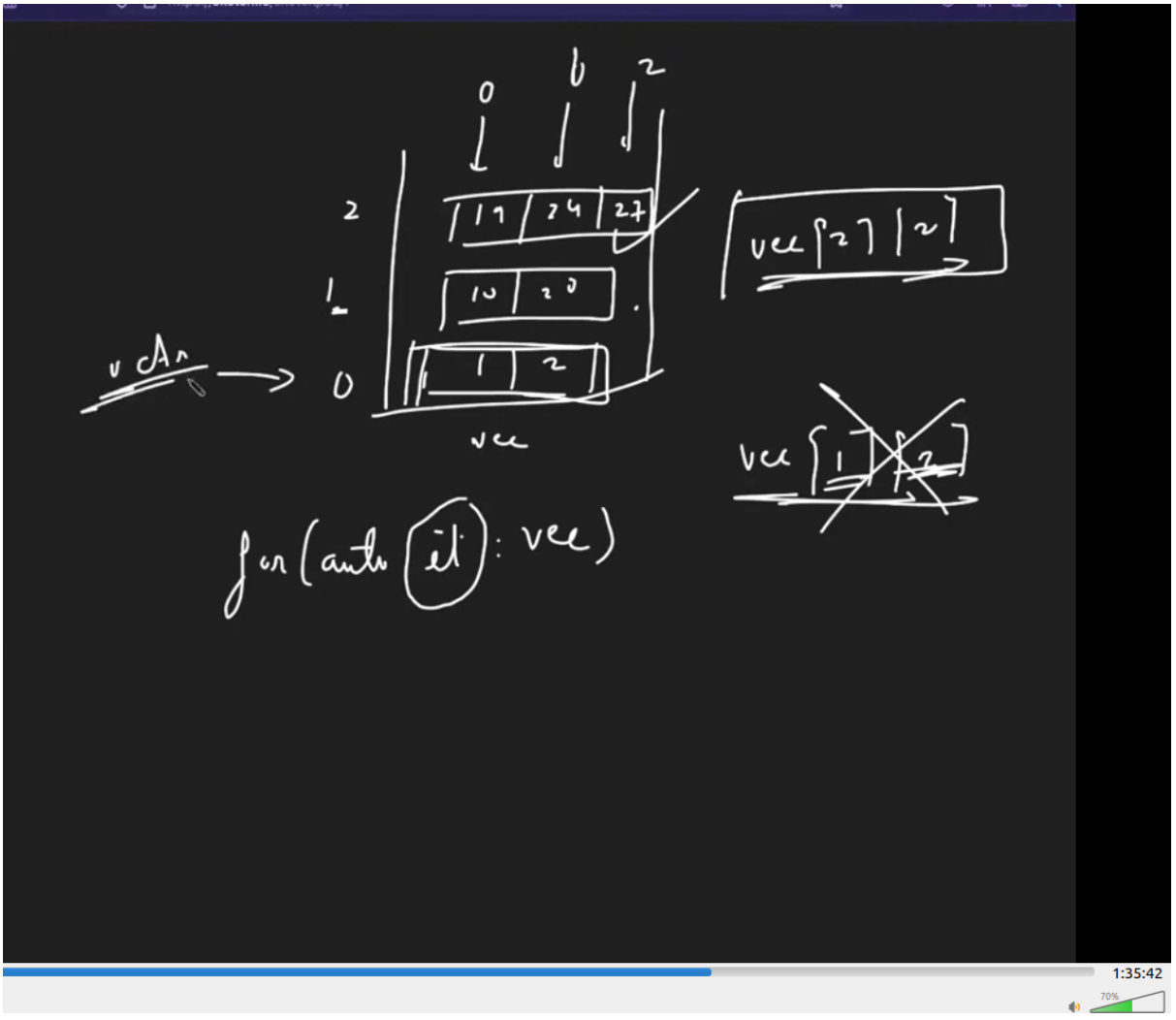
```cpp
// copy the entire vec2 into vec3
vector<int> vec3(vec2.begin(), vec2.end()); // -> []
vector<int> vec3(vec2);



vector<int> raj;
raj.push_back(1);
raj.push_back(3);
raj.push_back(2);
raj.push_back(5); // -> {1, 3, 2, 5}

vector<int> raj1(raj.begin(), raj.begin() + 2);  // -> {1, 3
```

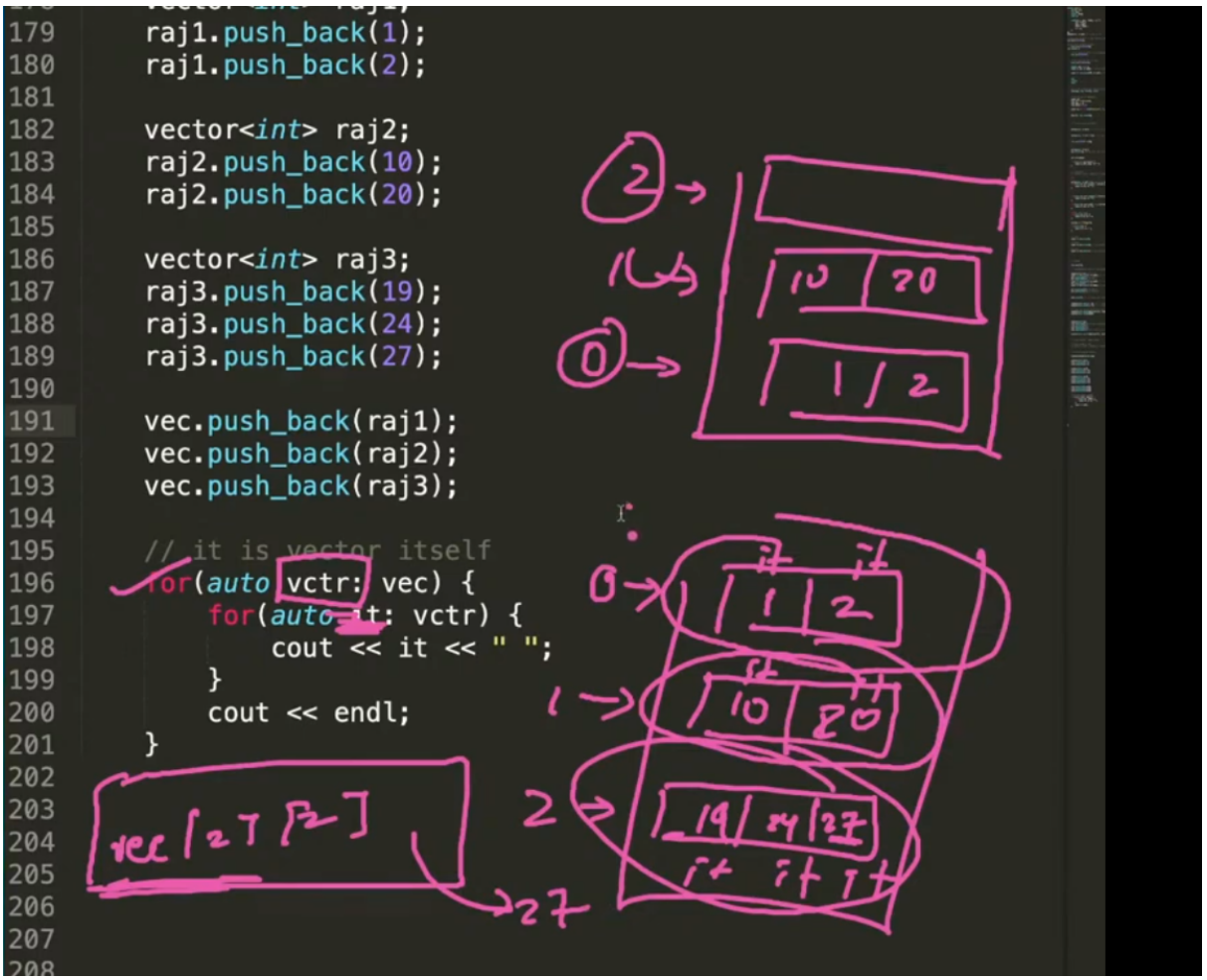


```cpp
// lower bound , upper bound

// swap swap(v1, v2)
// begin(), end(), rbegin(), rend()


// to defining 2d vectors

vector<vector<int>> vec;

vector<int> raj1;
raj1.push_back(1);
raj1.push_back(2);

vector<int> raj2;
raj2.push_back(10);
raj2.push_back(20);

vec.push_back(raj1);
vec.push_back(raj2);
```

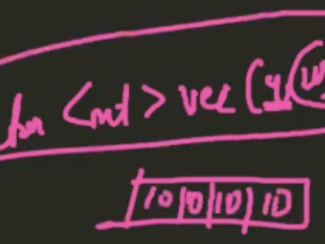


```cpp
    vector<vector<int>> vec;

    vector<int> raj1;
    raj1.push_back(1);
    raj1.push_back(2);

    vector<int> raj2;
    raj2.push_back(10);
    raj2.push_back(20);

    vector<int> raj3;
    raj3.push_back(19);
    raj3.push_back(24);
    raj3.push_back(27);

    vec.push_back(raj1);
    vec.push_back(raj2);
    vec.push_back(raj3);

    // it is vector itself
    for(auto vctr: vec) {
        for(auto it: vctr) {
            cout << it << " ";
        }
        cout << endl;
    }
```

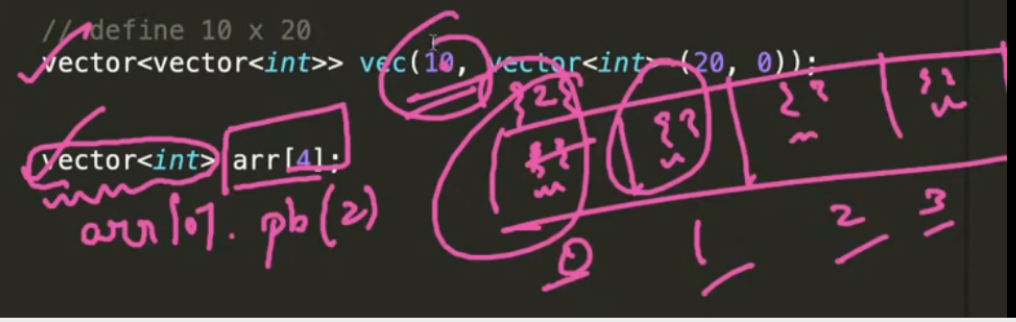

0
1
2
2
1
0
17
24
27
10
20
1
2
vec
vec[2][2]
vec[1][2]
fun(auto it : vec)


raj1.push_back(1);
raj1.push_back(2);
vector<int> raj2;
raj2.push_back(10);
raj2.push_back(20);
vector<int> raj3;
raj3.push_back(19);
raj3.push_back(24);
raj3.push_back(27);
vec.push_back(raj1);
vec.push_back(raj2);
vec.push_back(raj3);
// it is vector itself
for(auto vctr: vec) {
for(auto it: vctr) {
cout << it << " ";
}
cout << endl;
}

```
191     vec.push_back(raj1);
192     vec.push_back(raj2);
193     vec.push_back(raj3);
194
195     // it is vector itself
196     for(auto vctr: vec) {
197         for(auto it: vctr) {
198             cout << it << " ";
199         }
200         cout << endl;
201     }
202
203
204     for(int i = 0;i<vec.size();i++) {
205         for(int j = 0;j<vec[i].size();j++) {
206             cout <<  vec[i][j] << " ";
207         }
208         cout << endl;
209     }
210
211
212     // define 10 x 20
213     vector<vector<int>> vec(10, )
214
215
216
217
218
```



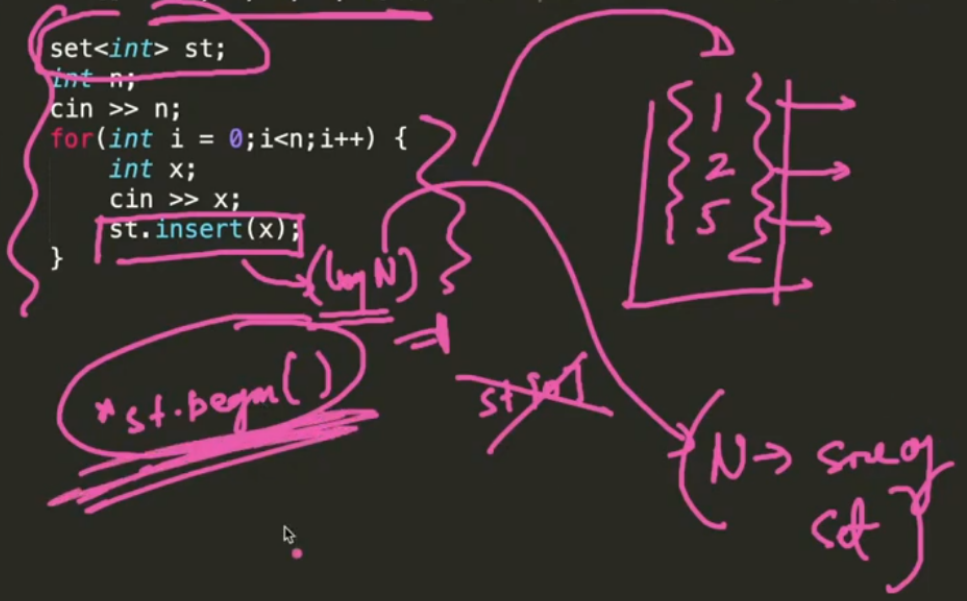

```
211
212     // define 10 x 20
213     vector<vector<int>> vec(10, vector<int> (20, 0));
214
215
216     vector<int> arr[4];
217
218
219
220
```



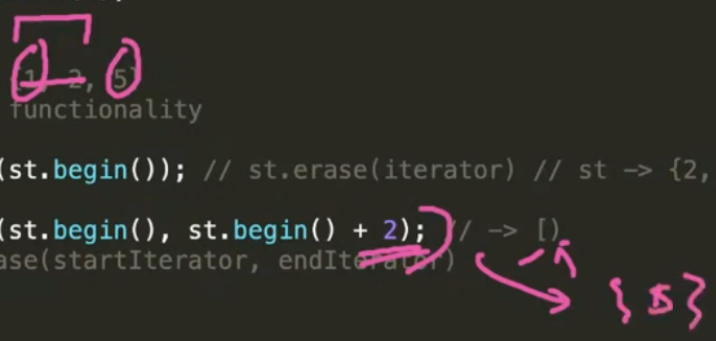

```
211
212     // define 10 x 20
213     vector<vector<int>> vec(10, vector<int> (20, 0));
214     vec.push_back(vector<int>(20, 0));
215     cout << vec.size() << endl; // 11 prints
216
217     vec[2].push_back(1);
218
219
220     vector<int> arr[4];
221     arr[1].push_back(0);
222
223
224     // 10 x 20 x 30 // int arr[10][20][30]
225     vector<vector<vector<int>>> vec(10, vector<vector<int>> ve
226
227
228
```

```cpp
for(auto it = arr.rbegin(); it>arr.rend();it++) {
    cout << *it << " ";
}
```

```cpp
// set

// given n elements, tell me the number of unique elements
arr[] = {2, 5, 2, 1, 5} // 3 unique elements -> {1. 2. 5}

set<int> st;
int n;
cin >> n;
for(int i = 0;i<n;i++) {
    int x;
    cin >> x;
    st.insert(x);
}
```




```cpp
// given n elements, tell me the number of unique elements
arr[] = {2, 5, 2, 1, 5} // 3 unique elements -> {1. 2. 5}

set<int> st;
int n;
cin >> n;
for(int i = 0;i<n;i++) {
    int x;
    cin >> x;
    st.insert(x);
}

// st -> {1, 2, 5}
// erase functionality

st.erase(st.begin()); // st.erase(iterator) // st -> {2, 5}

st.erase(st.begin(), st.begin() + 2); // -> []
// st.erase(startIterator, endIterator)
```



```
248
249     // st -> {1, 2, 5}
250     // erase functionality
251     // log n
252     st.erase(st.begin()); // st.erase(iterator) // st -> {2, 5}
253
254     // log n
255     // 77777777777
256     st.erase(st.begin(), st.begin() + 2); // -> []
257     // st.erase(startIterator, endIterator)
258
259     st.erase(5) // st.erase(key) // delete the 5
260
261
262     set<int> st = {1, 5, 7, 8};
263
264     auto it = st.find(7); // log n
265
266     auto it = st.find(9); // it = st.end();
267
```

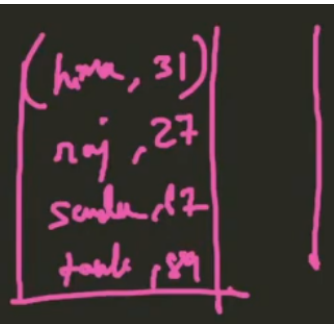
```
237     // given n elements, tell me the number of unique elements
238     arr[] = {2, 5, 2, 1, 5} // 3 unique elements -> {1. 2. 5}
239
240     set<int> st;
241     int n;
242     cin >> n;
243     for(int i = 0;i<n;i++) {
244         int x;
245         cin >> x;
246         st.insert(x);
247     }
248
249     // st -> {1, 2, 5}
250     // erase functionality
251     // log n
252     st.erase(st.begin()); // st.erase(iterator) // st -> {2, 5}
253
254     // log n
255     // 77777777777
256     st.erase(st.begin(), st.begin() + 2); // -> []
257     // st.erase(startIterator, endIterator)
258
259     st.erase(5) // st.erase(key) // delete the 5 -> {1, 2}
260
261
262     set<int> st = {1, 5, 7, 8};
263
264     auto it = st.find(7); // log n // it will be iterator to 7
265
266     auto it = st.find(9); // it = st.end();
267
268     st.emplace(6); // st.insert(6)
269
270     cout << st.size() << endl;
```

```
    set<int> st;
    st.insert(5); // -> {5}
    st.insert(5); // -> {5}

    for(auto it=st.begin(); it!=st.end();it++) {
        cout << *it << " ";
    }

    for(auto it : st) {
        cout << it << endl;
    }

    // delete the entire set
    st.erase(st.begin(), st.end()); // makes sure the entire set is de


    unordered_set<int> st;

    st.insert(2);
    st.insert(3);
    st.insert(1);

    // average time complexity is O(1)
    // tle -> switch to set
    // but the worst case is linear in nature, O(set size)
}

```

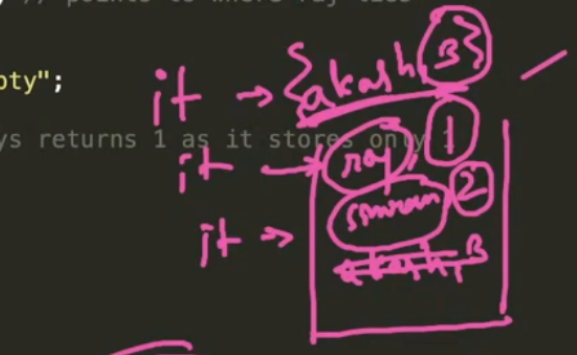
```
    ms.erase(ms.find(2));
    ms.erase(ms.find(2), ms.find(2) + 2);

```

```
    multiset<int> ms;

    ms.insert(1);
    ms.insert(1);
    ms.insert(2);
    ms.insert(2);
    ms.insert(3); // ms.emplace(3)
    // st -> {1, 1, 2, 2, 3}

    ms.erase(2); // all the instances will be erased

    auto it = ms.find(2); // returns an iterator pointing to the first
    ms.clear(); // deleted the entire set
    ms.erase(ms.begin(), ms.end()); // deletes the entire set
    // log n in size


    for(auto it=st.begin(); it!=st.end();it++) {
        cout << *it << " ";
    }

    for(auto it : st) {
        cout << it << endl;
    }


```

```cpp
// Key Value
// raj -> 27
// hima -> 31
// sandeep -> 67
// tank -> 89
map<string, int> mpp;
mpp["raj"] = 27;
mpp["hima"] = 31;
mpp["sandeep"] = 67;
mpp["tank"] = 89;
```




```cpp
// map only stores unique keys
// log n is the tc of map
map<string, int> mpp;
mpp["raj"] = 27;
mpp["hima"] = 31;
mpp["praveer"] = 31;
mpp["sandeep"] = 67;
mpp["tank"] = 89;
mpp["raj"] = 29;
mpp.emplace("raj", 45);
mpp.erase("raj"); // mpp.erase(key)
mpp.erase(mpp.begin()); // mpp.erase(iterator)
mpp.clear(); // entire map is cleaned up
mpp.erase(mpp.begin(), mpp.begin()+2); // cleans up a given range
auto it = mpp.find("raj"); // points to where raj lies

if(mpp.empty()) {
    cout << "Yes it is empty";
}
mpp.count("raj"); // always returns 1 as it stores only 1
// instance of raj

pair<int,int> pr;
pr.first = 1;
pr.second = 10;

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



```
368     // printing map
369     for(auto it: mpp) {
370         cout << it.first << " " << it.second << endl;
371     }
372
373     for(auto it = mpp.begin(); it!=mpp.end();it++) {
374         cout << it->first << " " << it->second << endl;
375     }
376
377     // does not stores in any order
378     unordered_map<int,int> mpp;
379     // o(1) in almost all cases
380     // o(n) in the worst case, where n is the container size
381
```

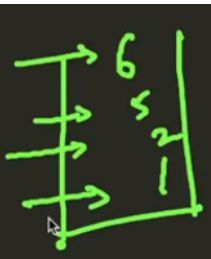
```
377     // does not stores in any order
378     unordered_map<int,int> mpp;
379     // unordered_map<pair<int,int>,int> mpp; xxxxxx
380     // o(1) in almost all cases
381     // o(n) in the worst case, where n is the container size
382
383
384     // Pair class
385     pair<int,int> pr = {1,2};
386     pair< pair<int,int>, int> pr = {{1,2}, 2};
387     cout << pr.first.second << endl;
388     pair<pair<int,int>, pair<int,int>> pr = {{1,2},{2, 4}};
389     cout << pr.first.first; -> 1
390     cout << pr.second.second; -> 4
391
392     vector<pair<int,int>> vec;
393     set<pair<int,int>> st;
394     map< pair<int,int>, int> mpp;
395
```


multimap<string, int> mpp;
mpp.emplace("raj", 2);
mpp.emplace("raj", 5);
// Stack and Queue
stack<int> st; // lifo ds
// pop
// top
// size
// empty
// push and emplace
st.push(2);
st.push(4);
st.push(3);
st.push(1);
References:
prime.cpp:413
prime.cpp:414
prime.cpp:415
cout <
// prints
st.pop
es the last entered element
cout <<
// prints 3
st.pop();
cout << st.top();

```
406     stack<int> st; // lifo ds
407     // pop
408     // top
409     // size
410     // empty
411     // push and emplace
412
413     st.push(2);
414     st.push(4);
415     st.push(3);
416     st.push(1);
417
418
419     cout << st.top() // prints 2
420     st.pop(); // deletes the last entered element
421     cout << st.top(); // prints 3
422     st.pop();
423     cout << st.top();
424
425     bool flag = st.empty(); // returns true if stack is empty, or fals
426
427     // deleted the entire stack
428     while(!st.empty()) {
429         st.pop();
430     }
431
432     cout << st.size() << endl; // number of elements in the stack
433
434     stack<int> st;
435     if(!st.empty()) {
436         cout << st.top() << endl; // throw error
437     }
438
```

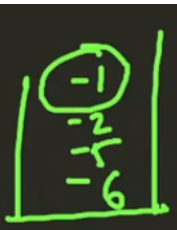
```
440     // queue // fifo operation ds
441     // push
442     // front
443     // pop
444     // size
445     // empty
446
447     queue<int> q;
448     q.push(1);
449     q.push(5);
450     q.push(3);
451     q.push(6);
452
453     cout << q.front();; // prints 1
454     q.pop();
455     cout << q.front(); // prints 5
456
457     // linear time
458     while(!q.empty()) {
459         q.pop();
460     }
461
462
463
```

```
463
464
465
466     // priority_queue
467     // push
468     // size
469     // top pop empty
470     priority_queue<int> pq;
471     pq.push(1);
472     pq.push(5);
473     pq.push(2);
474     pq.push(6);
475
476
477
```




```
471     pq.push(1);
472     pq.push(5);
473     pq.push(2);
474     pq.push(6);
475
476     cout << pq.top(); // print 6
477     pq.pop();
478     cout << pq.top(); // print 5
479
480     priority_queue<pair<int,int>> pq;
481     pq.push(1, 5);
482     pq.push(1, 6);
483     pq.push(1, 7);
484
485
```

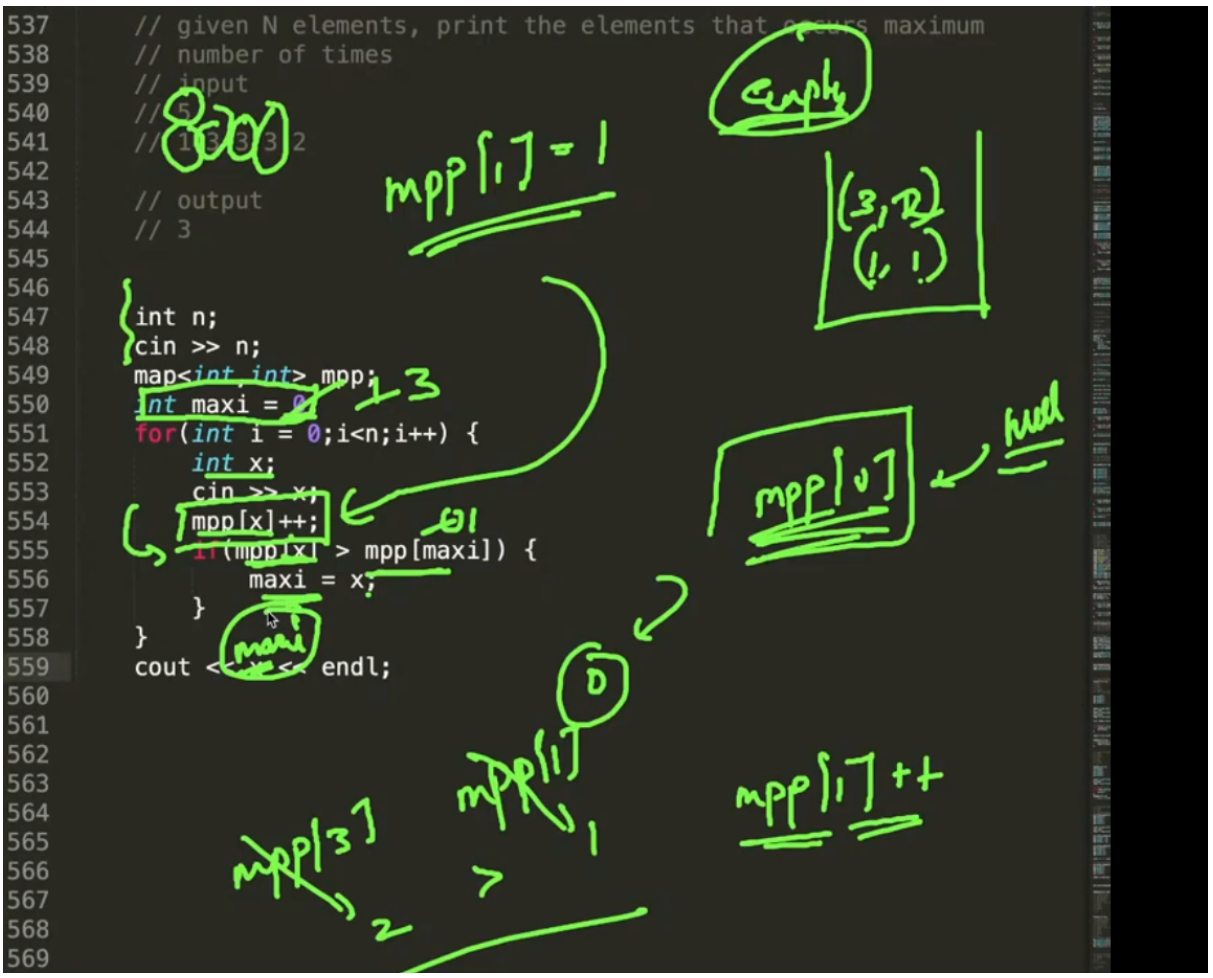
```
480     priority_queue<pair<int,int>> pq;
481     pq.push(1, 5);
482     pq.push(1, 6);
483     pq.push(1, 7);
484
485     priority_queue<int> pq;
486     pq.push(-1);
487     pq.push(-5);
488     pq.push(-2);
489     pq.push(-6);
490
491     cout << -1 * pq.top() << endl; // prints 1
492
493
494
```



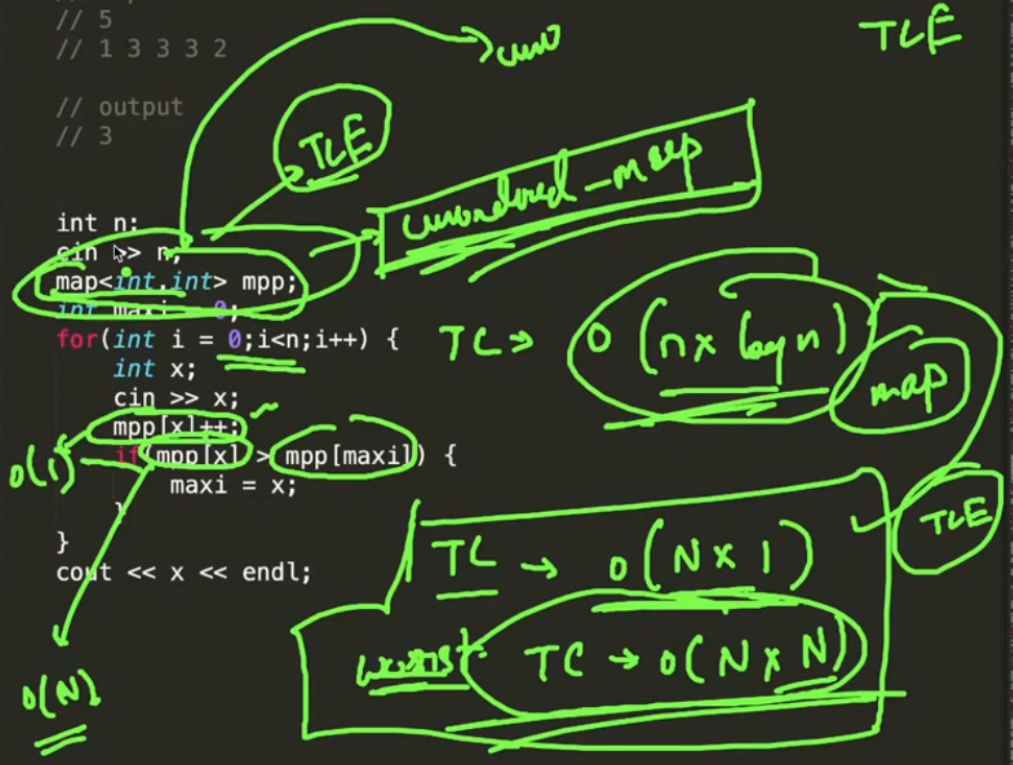

```
492
493     // min priority queue is
494     priority_queue<int, vector<int>, greater<int>> pq;
495     pq.push(1);
496     pq.push(5);
497     pq.push(2);
498     pq.push(6);
499
500     cout << pq.top() << endl; // prints 1
501
502
503
504     priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<
505
```

```
506
507     dequeue<int> dq;
508     // push_front()
509     // push_back()
510     // pop_front()
511     // pop_back()
512     // begin, end, rbegin, rend
513     // size
514     // clear
515     // empty
516     // at
517
518
519     list<int> ls;
520     // push_front()
521     // push_back()
522     // pop_front()
523     // pop_back()
524     // begin, end, rbegin, rend
525     // size
526     // clear
527     // empty
528     // at
529     // remove -> O(1)
530     ls.push_front(1);
531     ls.push_front(2);
532     ls.push_front(3);
533     ls.remove(2); -> // o(1) operation
534
```

```
537     // given N elements, print the elements that occurs maximum
538     // number of times
539     // input
540     // 5
541     // 1 3 3 3 2
542
543     // output
544     // 3
545
546
547     int n;
548     cin >> n;
549     map<int,int> mpp;
550     int maxi = 0;
551     for(int i = 0;i<n;i++) {
552         int x;
553         cin >> x;
554         mpp[x]++;
555         if(mpp[x] > mpp[maxi]) {
556             maxi = x;
557         }
558     }
559     cout << x << endl;
560
561
```


// given N elements, print the elements that occurs maximum
// number of times
// input
// 5
// 1 3 3 3 2
// output
// 3
int n;
cin >> n;
map<int,int> mpp;
int maxi = 0;
for(int i = 0;i<n;i++) {
int x;
cin >> x;
mpp[x]++;
if(mpp[x] > mpp[maxi]) {
maxi = x;
}
}
cout << maxi << endl;
mpp[i] = 1
mpp[v]
mpp[i]++

```cpp
// given N elements, print the elements that occurs maximum
// number of times
// input
// 5
// 1 3 3 3 2

// output
// 3


int n;
cin >> n;
map<int,int> mpp;
int maxi = 0;
for(int i = 0;i<n;i++) {
    int x;
    cin >> x;
    mpp[x]++;
    if(mpp[x] > mpp[maxi]) {
        maxi = x;
    }
}
cout << x << endl;
```




```cpp
// given N elements, print all elements in sorted order
// input
// n = 6
/// 6 6 3 2 3 5

// output
// 2 3 3 5 6 6

int n;
cin >> n;
multiset<int> ms;
for(int i=0;i<n;i++) {
    int x;
    cin >> x;
    ms.insert(x);
}

for(auto it : ms) {
    cout << it << endl;
}
```

```cpp
    // Day 3
    // Bitset
    // int -> 16 bits
    // char -> 8 bits
    int a[100];
    char a[100];

    // bitset -> 1 bit

    bitset<5> bt; // stores 1 or 0
    cin >> bt; // 10111

    // all
    // true // false
    cout << bt.all(); // returns a true or a false

    // any
    // true
    cout << bt.any(); // bt -> 10011

    // false
    cout << bt.any(); // bt -> 00000

    // count
    // for bt -> 10100
    // prints 2
    cout << bt.count(); // print the number of set bits
    // flip
    // bt -> 10100
    bt.flip(2); // bt will become 10000

    bt.flip();

    // none
    // if none is set, then true, else false
    // bt -> 10000
    cout << bt.none(); // false

    // bt -> 00000
    cout << bt.none(); //true

    // set
    bt.set(); // 11111

    bt.set(2); // sets the 2nd index

    bt.set(2, 0);


    // reset
    bt.reset() // turn all indexes to 0

    bt.reset(2); // turn the 2nd index to 0

    // size
    cout << bt.size(); // prints 5

    // test

    cout << bt.test(1); // check if the bit is set or not at index  1
```

```
649
650     //·Algorithms·
651     //·sorting·
652     //·array,·vector·
653     int n;
654     cin >> n;
655     int arr[n];
656     for(int i = 0;i<n;i++) cin >> arr[i];
657     // takes n log n
658     sort(arr, arr+n); // in increasing order
659     // sort from 1 to 3
660     sort(arr + 1, arr + 3);
661
662     vector<int> vec(5, 0);
663     for(int i = 0;i<n;i++) {
664         cin >> vec[i];
665     }
666
667     sort(vec.begin(), vec.end()); // []
668
669     // vec -> {1, 6, 2, 7, 4}
670     //          0  1  2  3  4
671     // sort it so that only indexes from 1 to 3
672     // final vec -> {1, 2, 6, 7, 4}
673     sort(vec.begin() + 1, vec.begin() + 4); // [1, 4)
674
675
676
677
678
```

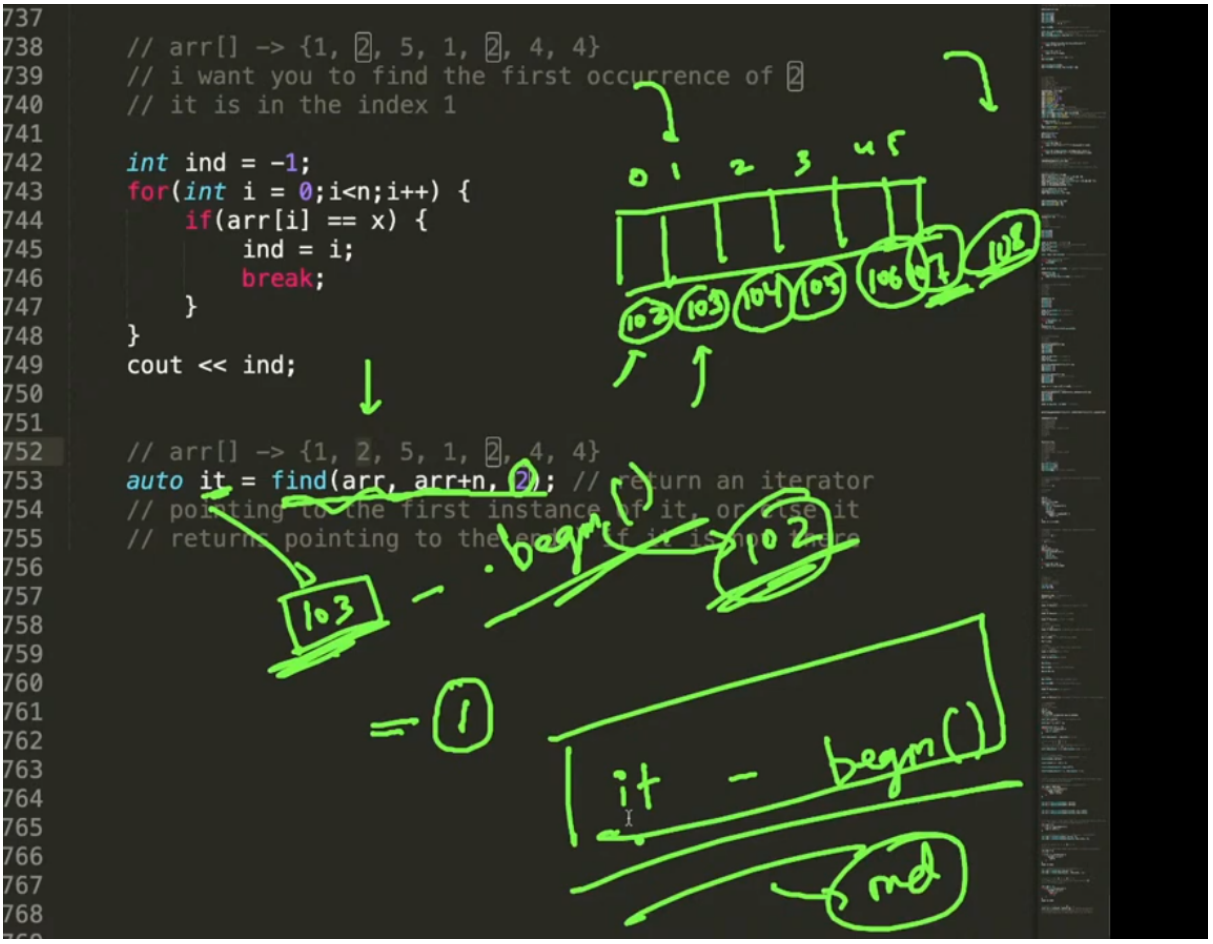
```
675
676     // If I wanna reverse
677     // reverse(startIterator, endIterator) -> []
678     reverse(arr, arr+n);
679
680     reverse(arr + 1, arr + 4);
681
682     reverse(vec.begin(), vec.end());
683
684     reverse(vec.begin() + 1, vec.begin() + 4);
685
686
```

```cpp
// If i want to fine the maximum elements in any index range
// i to j give me the maximum

// *max_element(firstIterator, lastIterator);
int maxi = INT_MIN;
for(int k = i;k<=j;k++) {
    if(a[k] > maxi) {
        maxi = a[k];
    }
}


int el = *max_element(arr, arr+n);
int el = *min_element(arr, arr+n);


int el = *max_element(vec.begin(), vec.end());
int el = *min_element(vec.begin(), vec.end());
```

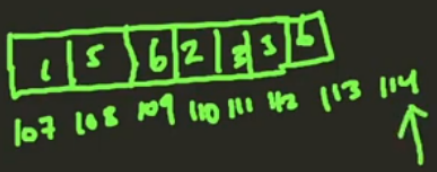
```cpp
    // I give you a range and I want you to find the sum in that ra
    // i - j, tell me the sum in that range i to j
    int sum = 0;
    for(int k = i;k<=j;k++) {
        sum += arr[k];
    }

    // accumulate(startIterator, endIterator, initialSum);
    int sum = accumulate(arr, arr+n, 0);
    int sum = accumulate(vec.begin(), vec.end(), 0);
```

```cpp

    // arr[] -> [1, 6, 7, 1, 2, 1, 3]
    // x = 1
    // tell me how many times the element 1 occurs in the array
    int cnt = 0;
    // O(N)
    for(int i = 0;i<n;i++) {
        if(arr[i] == x) {
            cnt++;
        }
    }
    cout << cnt;

    /// count(firstIterator, lastIterator, x)
    int cnt = count(arr, arr+n, 1);
```

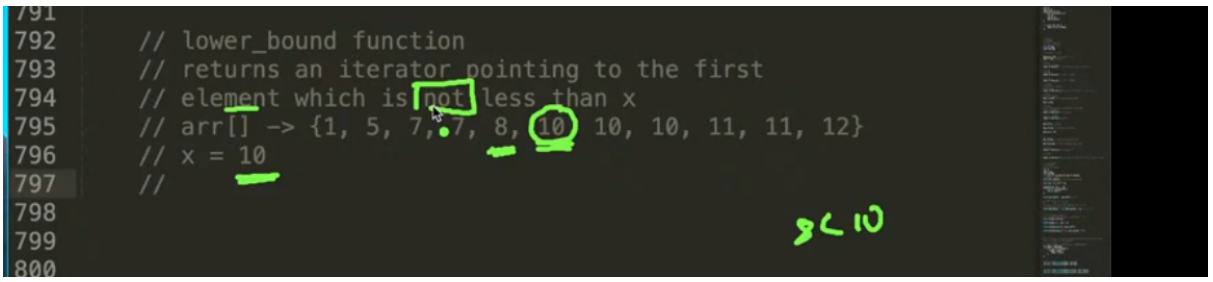
```cpp
// arr[] -> {1, 2, 5, 1, 2, 4, 4}
// i want you to find the first occurrence of 2
// it is in the index 1

int ind = -1;
for(int i = 0;i<n;i++) {
    if(arr[i] == x) {
        ind = i;
        break;
    }
}
cout << ind;


// arr[] -> {1, 2, 5, 1, 2, 4, 4}
auto it = find(arr, arr+n, 2); // return an iterator
// pointing to the first instance of it, or else it
// returns pointing to the end() if it is not there
```



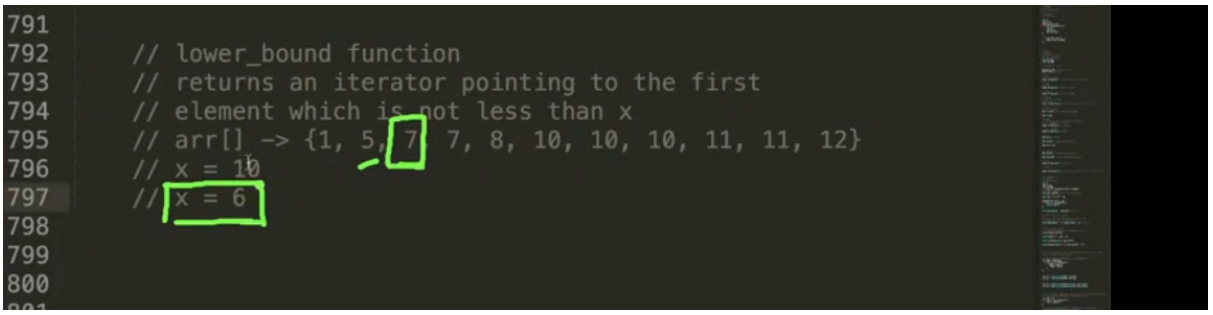
```
744         if(arr[i] == x) {
745             ind = i;
746             break;
747         }
748     }
749     cout << ind;
750
751
752     // arr[] -> {1, 2, 5, 1, 2, 4, 4}
753     auto it = find(arr, arr+n, 2); // return an iterator
754     // pointing to the first instance of it, or else it
755     // returns pointing to the end() if it is not there
756
757     int ind = it - arr;
758
759
760     auto it = find(vec.begin(), vec.end(), 2);
761     int ind = it - vec.begin();
762
763
764     // arr[] -> {1, 5, 6, 2, 3, 5, 6}
765     // x = 4
766     auto it = find(vec.begin(), vec.end(), 4)
767
```



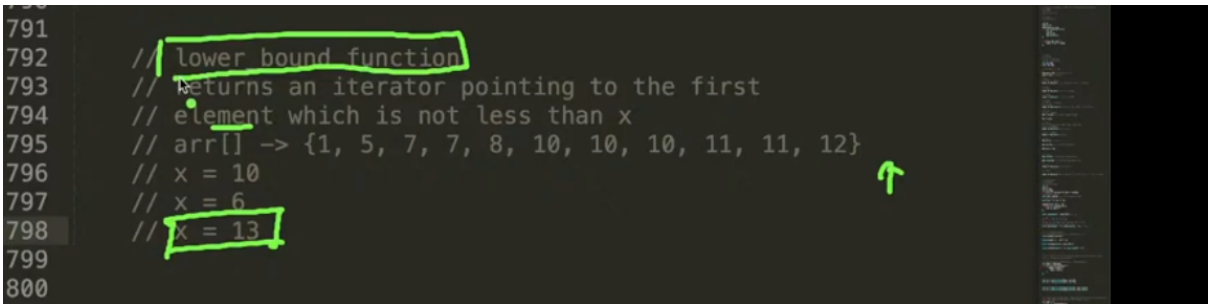

```
764     // arr[] -> {1, 5, 6, 2, 3, 5, 6}
765     // x = 4
766     auto it = find(vec.begin(), vec.end(), 4);
767     if(it == vec.end()) {
768         cout << "element is not present";
769     }
770     else {
771         cout << "Element is first present at: " << it - vec.begin()
772     }
773
774
775     // binary search
776     // this stl only works on sorted arrrays
777     // arr[] -> {1, 5, 7, 9, 10}
778     // x = 9
779     // true -> 9 exists in my arr
780     // x = 8
781     // false -> 8 does not exist in my arr
782
783
784     // binary_search(firstIterator, lastIterator, x)
785     // returns a true or returns a false
786     bool res = binary_search(arr, arr+n, 8);
787     bool res = binary_search(vec.begin(), vec.end(), 8);
788
789
790
791
```



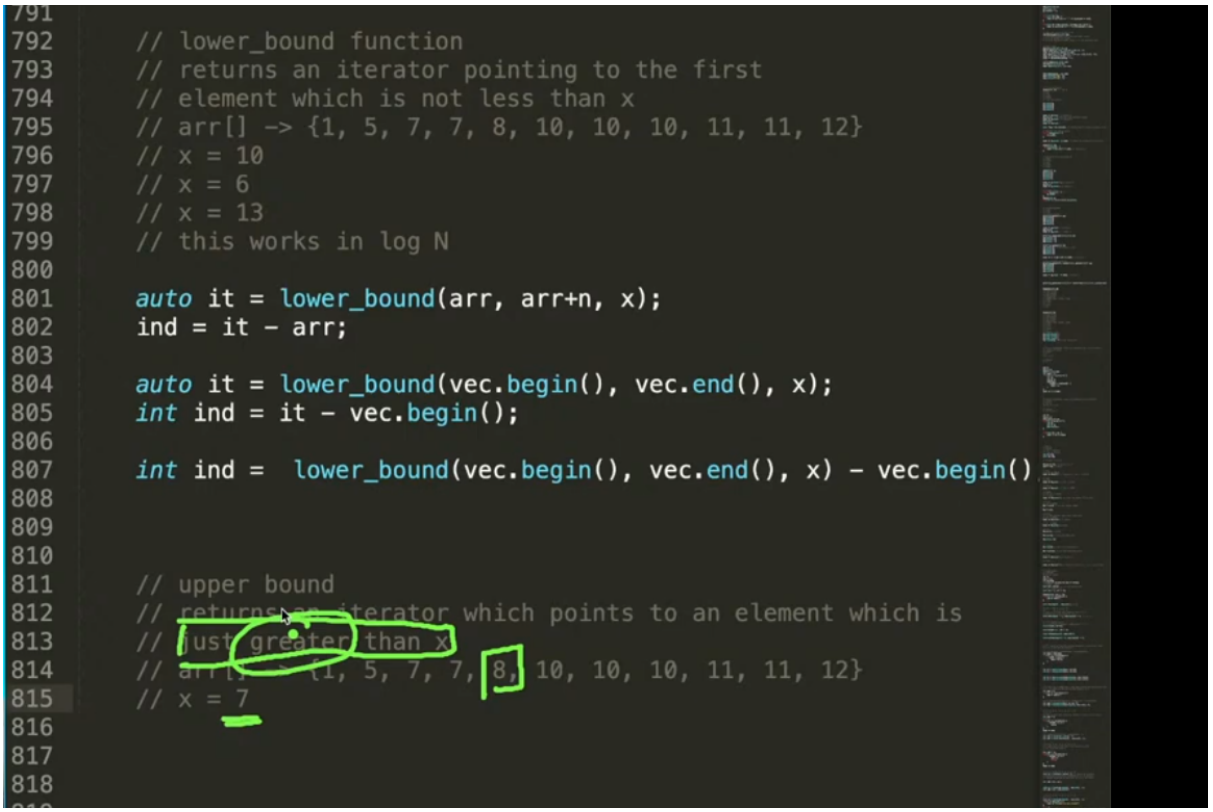
```
// lower_bound function
// returns an iterator pointing to the first
// element which is not less than x
// arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
// x = 10
//
```



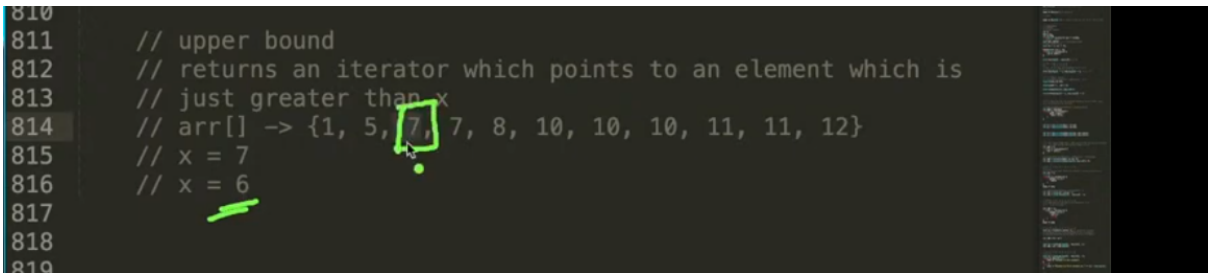
```
// lower_bound function
// returns an iterator pointing to the first
// element which is not less than x
// arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
// x = 10
// x = 6
```



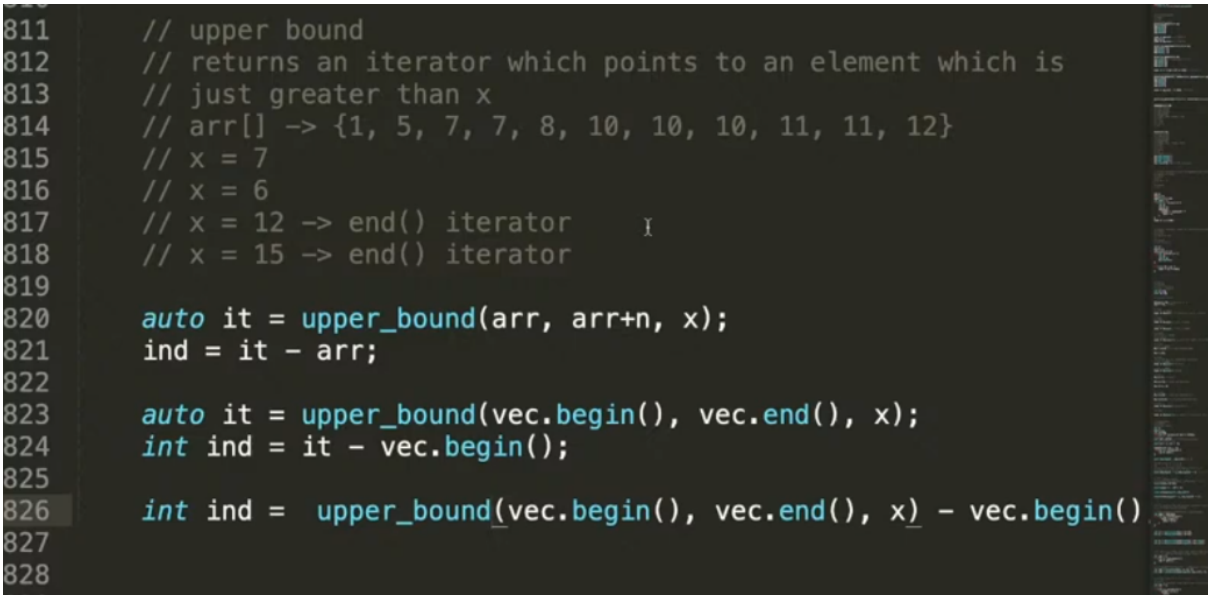
```
// lower_bound function
// returns an iterator pointing to the first
// element which is not less than x
// arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
// x = 10
// x = 6
// x = 13
```



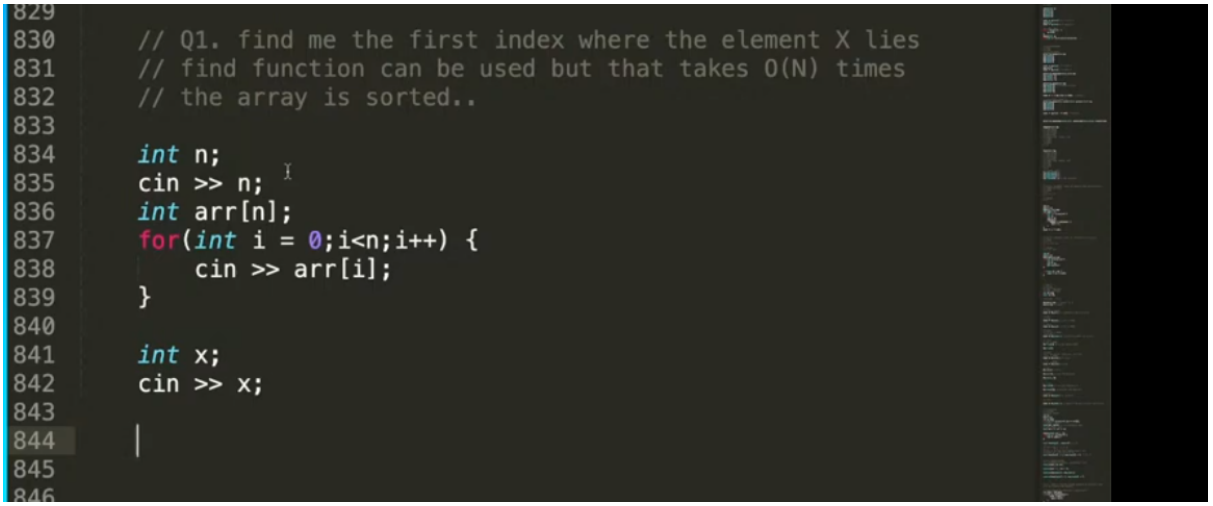
```
// lower_bound function
// returns an iterator pointing to the first
// element which is not less than x
// arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
// x = 10
// x = 6
// x = 13
// this works in log N

auto it = lower_bound(arr, arr+n, x);
ind = it - arr;

auto it = lower_bound(vec.begin(), vec.end(), x);
int ind = it - vec.begin();

int ind =  lower_bound(vec.begin(), vec.end(), x) - vec.begin()



// upper bound
// returns an iterator which points to an element which is
// just greater than x
// arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
// x = 7
```



```
// upper bound
// returns an iterator which points to an element which is
// just greater than x
// arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
// x = 7
// x = 6
```



```
// upper bound
// returns an iterator which points to an element which is
// just greater than x
// arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
// x = 7
// x = 6
// x = 12 -> end() iterator
// x = 15 -> end() iterator

auto it = upper_bound(arr, arr+n, x);
ind = it - arr;

auto it = upper_bound(vec.begin(), vec.end(), x);
int ind = it - vec.begin();

int ind =  upper_bound(vec.begin(), vec.end(), x) - vec.begin()
```



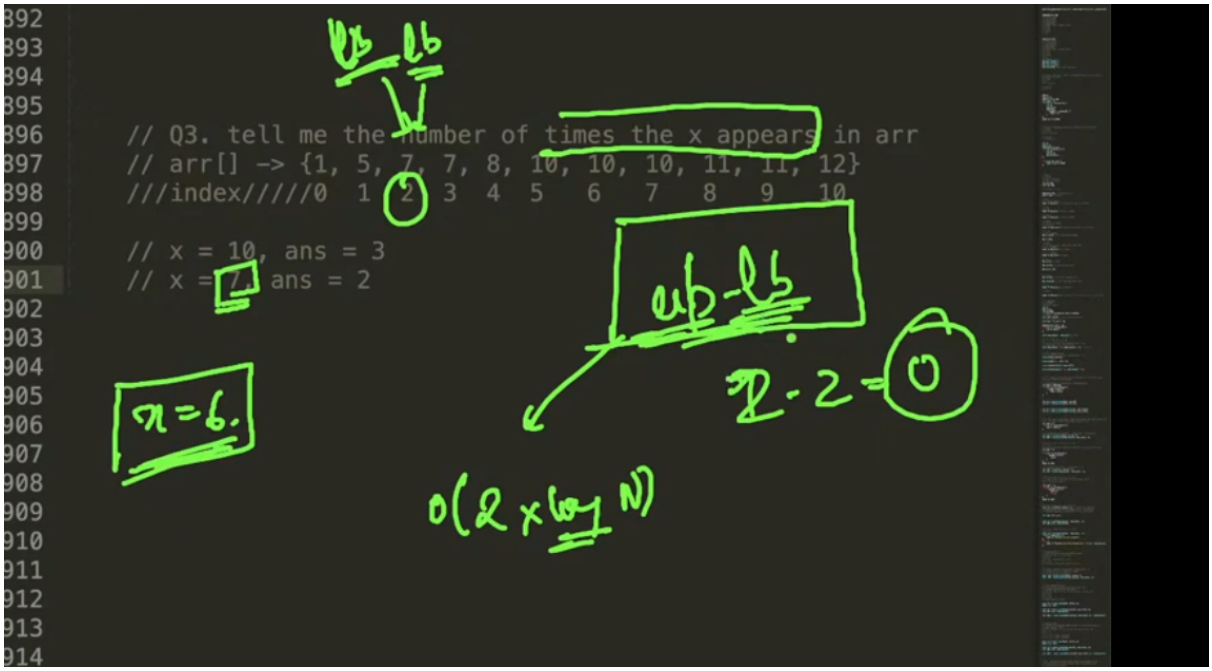
```
// Q1. find me the first index where the element X lies
// find function can be used but that takes O(N) times
// the array is sorted..

int n;
cin >> n;
int arr[n];
for(int i = 0;i<n;i++) {
    cin >> arr[i];
}

int x;
cin >> x;
```

```
853     // There are couple of ways to do it
854     // 1st way
855     if(binary_search(arr, arr+n, x) == true) {
856         cout << lower_bound(arr, arr+n, x) - arr;
857     }
858     else cout << "does not exists";
859
860
861     // 2nd way
862     int ind = lower_bound(arr, arr+n, x) - arr;
863     // arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
864     //////////////0  1  2  3  4  5   6   7   8   9   1
865     // find x = 13 -> ind = 11, which is out of bound
866     // hence arr[11] will give you runtime error
867     if(ind != n && arr[ind] == x) {
868         cout << "Found at: " << ind;
869     }
870     else {
871         cout << "Not found";
872     }
873
874
```

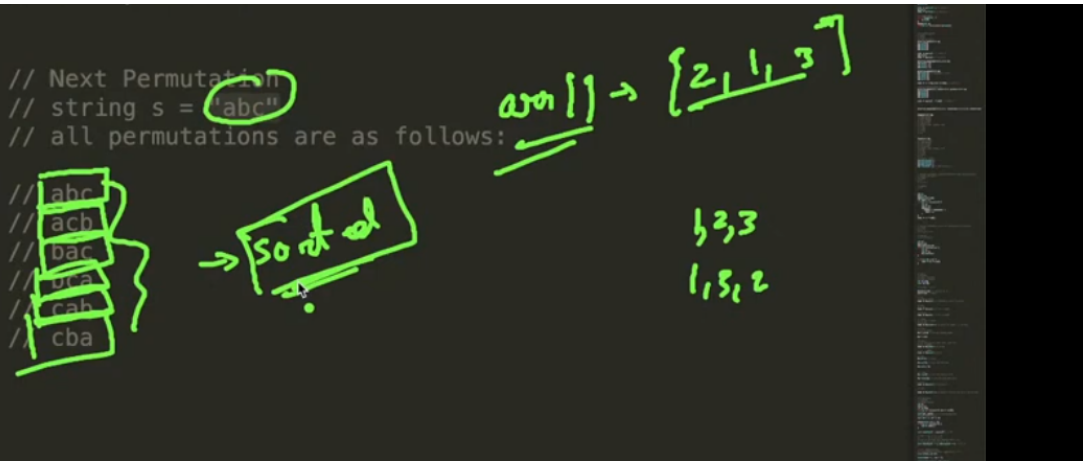
```
875
876     // Find me the last occurrence of x in an arr
877     // arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
878     ///index/////0  1  2  3  4  5   6   7   8   9   10
879
880     // last occurrence of x = 10, ans = 7th index
881     // last occurrence of x = 6, ans = does not exists
882     // last occurrence of x = 0,
883     // last occurence of x = 13
884     int ind = upper_bound(arr, arr+n, x) - arr;
885     ind -= 1;
886     if(ind>=0 && arr[ind] == x) {
887         cout << "last occurrence: " << ind;
888     }
889     else {
890         cout << "Does not exists";
891     }
892
893
894
```

```
891     }
892                                  lb          ub
893                                  ↓           ↓
894
895
896     // Q3. tell me the number of times the x appears in arr
897     // arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
898     ///index/////0  1  2  3  4  5   6   7   8   9   10
899
900     // x = 10, ans = 3
901     // x = 7, ans = 2
902
903                                  ub - lb
904                                  8 - 5 = 3
905
906
907
908
909
```

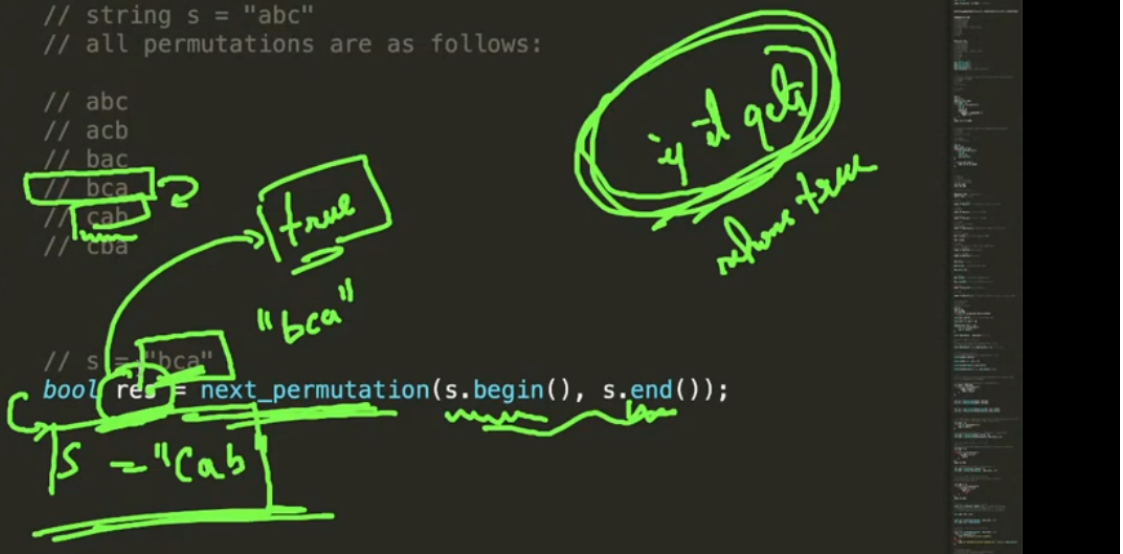
```
// Q3. tell me the number of times the x appears in arr
// arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
///index/////0  1  2  3  4  5   6   7   8   9   10

// x = 10, ans = 3
// x =  7, ans = 2
```



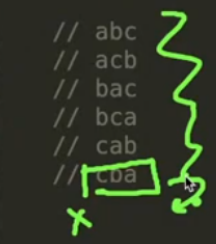

```
// Next Permutation
// string s = "abc"
// all permutations are as follows:

// abc
// acb
// bac
// bca
// cab
// cba
```

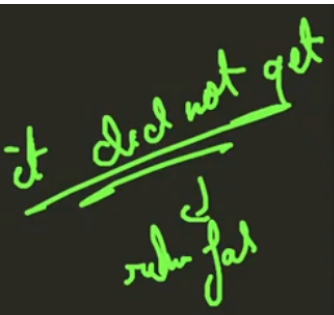


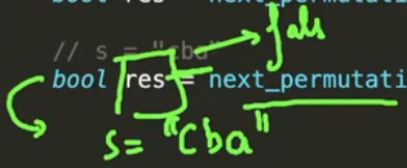

```
// Next Permutation
// string s = "abc"
// all permutations are as follows:

// abc
// acb
// bac
// bca
// cab
// cba

// s = "bca"
bool res = next_permutation(s.begin(), s.end());
```



```
902
903
904     // Next Permutation
905     // string s = "abc"
906     // all permutations are as follows:
907
908     // abc
909     // acb
910     // bac
911     // bca
912     // cab
913     // cba
914
915
916
917     // s = "bca"
918     bool res = next_permutation(s.begin(), s.end());
919
920     // s = "cba"
921     bool res = next_permutation(s.begin(), s.end());
922
923
924
925
926
927
928
```



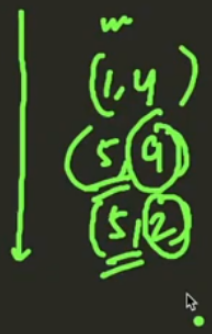




```
922
923     // if I give you any random string s = "bca"
924     // i want you to print all the permutations
925
926     string s = "bca";
927     sort(s.begin(), s.end()); // this makes the string as "abc"
928     do {
929         cout << s << endl;
930     } while(next_permutation(s.begin(), s.end()));
931
932
933     int arr[] = {1, 6, 5};
934     int n = 3;
935     sort(arr, arr + n); // this makes the array as {1, 5, 6}
936     do {
937         for(int i = 0;i<n;i++) cout << arr[i] << " ";
938         cout << endl;
939     } while(next_permutation(arr, arr+n)) ;
940
941
942     // prev permutation
943     bool res = prev_permutation(s.begin(), s.end());
944
```


// COMPARATOR
sort(arr, arr+n); // sorts everything in ascending order
sort(arr, arr+n, comp);
// descending
sort(arr, arr+n, comp);
// greater<int> is an inbuilt comparator
// which works only if you wanna do this in descending
sort(arr, arr+n, greater<int>);

```cpp
};
array<int, 3> arr; // -> {0, 0, 0}

// max size of 10^7 -> int, double, char
int arr[10000000];

// max size of 10^8 -> bool
bool arr[100000000];

bool comp(int el1, int el2) {
    if(el1 <= el2) {
        return true;
    }
    return false;
}

// arr
// pair<int,int> arr[] = {{1, 4}, {5, 2}, {5, 9}};
// i want you to sort this in such a way
// that the element who have first element in pair smaller
// appears first, and if first is equal then sort according
// to second and keep the larger second
```




```cpp
bool comp(int el1, int el2) {
    if(el1 <= el2) {
        return true;
    }
    return false;
}

bool comp(pair<int,int> el1, pair<int,int> el2) {
    if(el1.first < el2.first) {
        return true;
    }
    if(el1.first == el2.first) {
        if(el1.second > el2.second) {
            return true;
        }
    }

    return false;
}
```