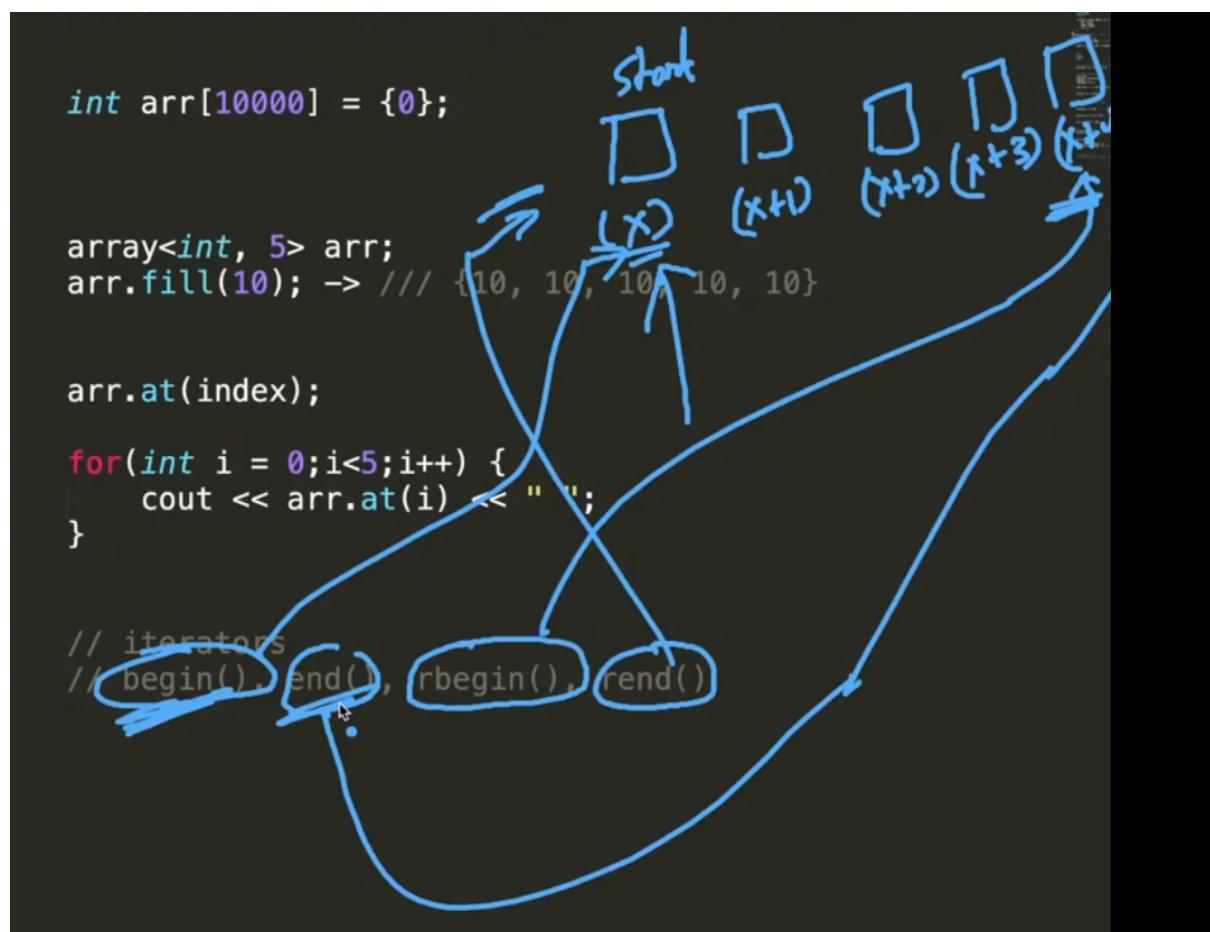


```

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

```



```
55  
56     int arr[10000] = {0};  
57  
58  
59  
60     array<int, 5> arr;  
61     arr.fill(10); -> // {10, 10, 10, 10, 10}  
62  
63  
64     arr.at(index);  
65  
66     for(int i = 0;i<5;i++) {  
67         cout << arr.at(i) << " ";  
68     }  
69  
70  
71     // iterators  
72     // begin(), end(), rbegin(), rend()  
73  
74     //  
75  
76     array<int, 5> arr = {1, 3, 4, 5, 6};  
77     for(auto it: arr.begin(); it!=arr.end();it++) {  
78         cout << *it << " ";  
79     }  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90     // for each loop |  
91     for(auto it: arr) {  
92         cout << it << " ";  
93     }  
94  
95
```

```
96     string s = "xhegcwe";
97     // x h e g c w e
98     for(auto c:s) {
99         cout << c << " ";
100    }
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];
```

```
128 // VECTOR
129
130 int arr[50];
131
132 // segmentation fault if you push_back 10^7 times
133
134 vector<int> arr; // -> {}
135 cout << arr.size() << endl; // -> print 0
136 arr.push_back(0); // {0}
137 arr.push_back(2); // {0,2}
138 cout << arr.size() << endl; // -> print 2
139 arr.pop_back(); // {0}
140 cout << arr.size() << endl; // print 1
141
142 arr.push_back(0); // {0,0}
143 arr.push_back(2); // {0,0,2}
144
145
146
147 vec.clear(); // --> erase all elements at once {}|
```



```
150 vector<int> vec1(4, 0); // -> {0,0,0,0}
151 vector<int> vec2(4, 10); // -> {10,10,10,10}
152
153 // copy the entire vec2 into vec3
154 vector<int> vec3(vec2.begin(), vec2.end()); // -> []
155 vector<int> vec3(vec2);
156
157
158
159 vector<int> raj;
160 raj.push_back(1); // raj.emplace_back(1); // emplace_back
161 raj.push_back(3);
162 raj.push_back(2);
163 raj.push_back(5); // -> {1, 3, 2, 5}
164
165 vector<int> raj1(raj.begin(), raj.begin() + 2); // -> {1,
```

```

152 // copy the entire vec2 into vec3
153 vector<int> vec3(vec2.begin(), vec2.end()); // -> []
154 vector<int> vec3(vec2);
155
156
157
158
159 vector<int> raj;
160 raj.push_back(1);
161 raj.push_back(3);
162 raj.push_back(2);
163 raj.push_back(5); // -> {1, 3, 2, 5} .
164
165 vector<int> raj1(raj.begin(), raj.begin() + 2); // -> {1, 3
166
167
168
169
170

```

The diagram shows a horizontal sequence of four boxes containing the numbers 1, 3, 2, and 5. A blue bracket labeled "Sitterwurz" spans from the first box to the third box. An arrow points from the center of this bracket to the text "raj.begin()". Another blue bracket labeled "Endwurz" spans from the third box to the fourth box. An arrow points from the center of this bracket to the text "raj.begin() + 2".

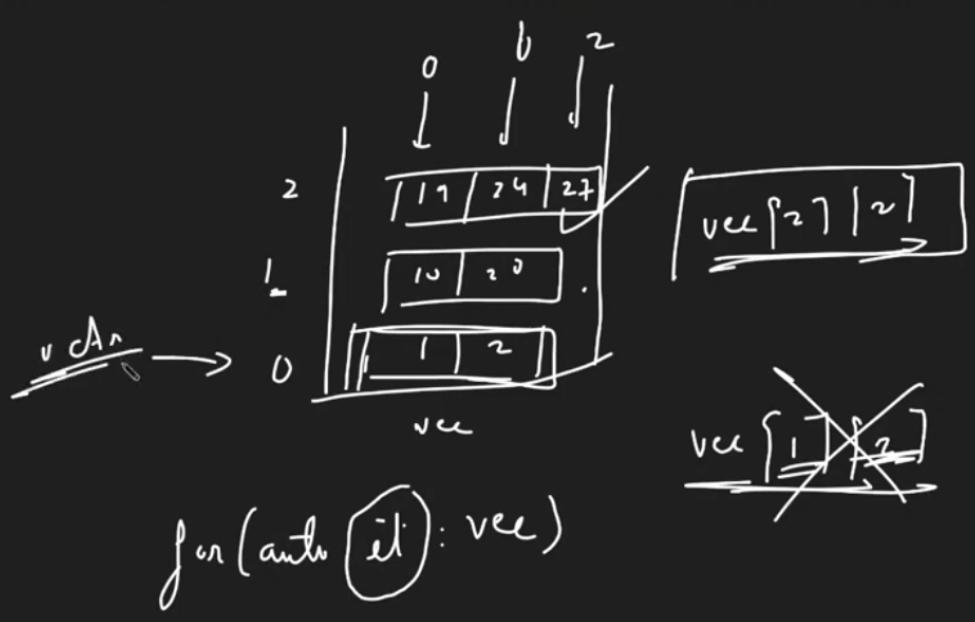
```

167 // lower bound , upper bound
168
169 // swap swap(v1, v2)
170 // begin(), end(), rbegin(), rend()
171
172
173
174 // to defining 2d vectors
175
176 vector<vector<int>> vec;
177
178 vector<int> raj1;
179 raj1.push_back(1);
180 raj1.push_back(2);
181
182 vector<int> raj2;
183 raj2.push_back(10);
184 raj2.push_back(20);
185
186 vec.push_back(raj1);
187 vec.push_back(raj2);
188
189
190
191
192
193
194
195

```

The diagram shows a 2D vector structure represented as a grid of four boxes. The top row contains two boxes labeled 10 and 20. The bottom row contains two boxes labeled 1 and 2. To the right of the grid, there are two small circles, each with an arrow pointing to it from one of the grid boxes. This illustrates how a 2D vector is stored as multiple 1D vectors.

```
176     vector<vector<int>> vec;
177
178     vector<int> raj1;
179     raj1.push_back(1);
180     raj1.push_back(2);
181
182     vector<int> raj2;
183     raj2.push_back(10);
184     raj2.push_back(20);
185
186     vector<int> raj3;
187     raj3.push_back(19);
188     raj3.push_back(24);
189     raj3.push_back(27);
190
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
```



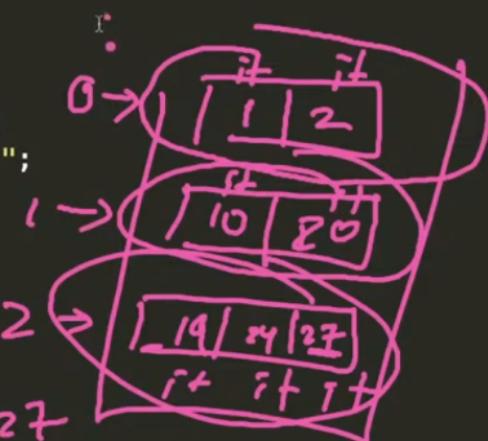
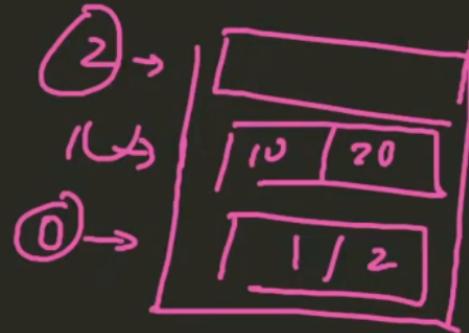
1:35:42



```

178
179     raj1.push_back(1);
180     raj1.push_back(2);
181
182     vector<int> raj2;
183     raj2.push_back(10);
184     raj2.push_back(20);
185
186     vector<int> raj3;
187     raj3.push_back(19);
188     raj3.push_back(24);
189     raj3.push_back(27);
190
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
205
206
207
208

```



`rec [2] P2]`

`27`

```

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

```

Vector <int> vec(10)

[10|10|10] ID



```

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     arr[1].pb(2);
219
220
221

```

```

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>>(20, vector<int>(30, 0)));
226
227
228

```

```

96
97     for(auto it = arr.rbegin(); it>arr.rend();it++) {
98         cout << *it << " ";
99     }
100    // set map
101
102    // set
103
104    // given n elements, tell me the number of unique elements
105    arr[] = {2, 5, 2, 1, 5} // 3 unique elements -> {1, 2, 5}
106
107    set<int> st;
108    int n;
109    cin >> n;
110    for(int i = 0;i<n;i++) {
111        int x;
112        cin >> x;
113        st.insert(x);
114    }
115
116    cout << *st.begin();
117
118    cout << endl;
119
120    cout << st.size();
121
122    cout << endl;
123
124    cout << st;
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161

```

```

126
127    // given n elements, tell me the number of unique elements
128    arr[] = {2, 5, 2, 1, 5} // 3 unique elements -> {1, 2, 5}
129
130    set<int> st;
131    int n;
132    cin >> n;
133    for(int i = 0;i<n;i++) {
134        int x;
135        cin >> x;
136        st.insert(x);
137    }
138
139    // st -> {1, 2, 5}
140    // erase functionality
141    st.erase(st.begin()); // st.erase(iterator) // st -> {2, 5}
142
143    st.erase(st.begin(), st.begin() + 2); // -> []
144    // st.erase(startIterator, endIterator)
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161

```

```
248 // st -> {1, 2, 5}
249 // erase functionality
250 // log n
251 st.erase(st.begin()); // st.erase(iterator) // st -> {2, 5}
252
253 // log n
254 // 77777777777
255 st.erase(st.begin(), st.begin() + 2); // -> []
256 // st.erase(startIterator, endIterator)
257
258 st.erase(5) // st.erase(key) // delete the 5
259
260
261 set<int> st = {1, 5, 7, 8};
262
263 auto it = st.find(7); // log n
264
265 auto it = st.find(9); // it = st.end();
266
267
268 // given n elements, tell me the number of unique elements
269 arr[] = {2, 5, 2, 1, 5} // 3 unique elements -> {1, 2, 5}
270
271
272 set<int> st;
273 int n;
274 cin >> n;
275 for(int i = 0;i<n;i++) {
276     int x;
277     cin >> x;
278     st.insert(x);
279 }
280
281 // st -> {1, 2, 5}
282 // erase functionality
283 // log n
284 st.erase(st.begin()); // st.erase(iterator) // st -> {2, 5}
285
286 // log n
287 // 77777777777
288 st.erase(st.begin(), st.begin() + 2); // -> []
289 // st.erase(startIterator, endIterator)
290
291 st.erase(5) // st.erase(key) // delete the 5 -> {1, 2}
292
293
294 set<int> st = {1, 5, 7, 8};
295
296 auto it = st.find(7); // log n // it will be iterator to 7
297
298 auto it = st.find(9); // it = st.end();
299
300 st.emplace(6); // st.insert(6)
301
302 cout << st.size() << endl;
```

```
272
273     set<int> st;
274     st.insert(5); // -> {5}
275     st.insert(5); // -> {5}
276
277     for(auto it=st.begin(); it!=st.end();it++) {
278         cout << *it << " ";
279     }
280
281     for(auto it : st) {
282         cout << it << endl;
283     }
284
285     // delete the entire set
286     st.erase(st.begin(), st.end()); // makes sure the entire set is de
287
288
289
290     unordered_set<int> st;
291
292     st.insert(2);
293     st.insert(3);
294     st.insert(1);
295
296     // average time complexity is O(1)
297     // tle -> switch to set
298     // but the worst case is linear in nature, O(set size)
299 }
300
327
328     ms.erase(ms.find(2));
329     ms.erase(ms.find(2), ms.find(2) + 2);
330
300     multiset<int> ms;
301
302     ms.insert(1);
303     ms.insert(1);
304     ms.insert(2);
305     ms.insert(2);
306     ms.insert(3); // ms.emplace(3)
307     // st -> {1, 1, 2, 2, 3}
308
309     ms.erase(2); // all the instances will be erased
310
311     auto it = ms.find(2); // returns an iterator pointing to the first
312     ms.clear(); // deleted the entire set
313     ms.erase(ms.begin(), ms.end()); // deletes the entire set
314     // log n in size
315
316
317     for(auto it=st.begin(); it!=st.end();it++) {
318         cout << *it << " ";
319     }
320
321     for(auto it : st) {
322         cout << it << endl;
323     }
324
325
326
```

```

333
334
335
336    // Key Value
337    // raj -> 27
338    // hima -> 31
339    // sandeep -> 67
340    // tank -> 89
341    map<string, int> mpp;
342    mpp["raj"] = 27;
343    mpp["hima"] = 31;
344    mpp["sandeep"] = 67;
345    mpp["tank"] = 89;
346
347
348
349

```

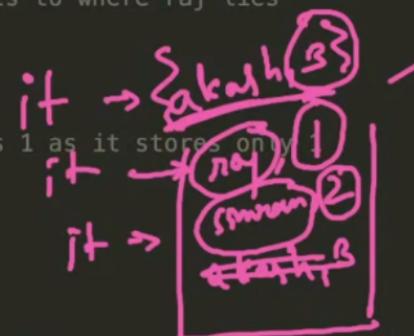
(hima, 31)
raj,	27
sandeep,	67
tank,	89

```

341    // map only stores unique keys
342    // log n is the tc of map
343    map<string, int> mpp;
344    mpp["raj"] = 27;
345    mpp["hima"] = 31;
346    mpp["praveer"] = 31;
347    mpp["sandeep"] = 67;
348    mpp["tank"] = 89;
349    mpp["raj"] = 29;
350    mpp.emplace("raj", 45);
351    mpp.erase("raj"); // mpp.erase(key)
352    mpp.erase(mpp.begin()); // mpp.erase(iterator)
353    mpp.clear(); // entire map is cleaned up
354    mpp.erase(mpp.begin(), mpp.begin() + 2); // cleans up a given range
355    auto it = mpp.find("raj"); // points to where raj lies
356
357    if(mpp.empty()) {
358        cout << "Yes it is empty";
359    }
360    mpp.count("raj"); // always returns 1 as it stores only
361    // instance of raj
362
363    pair<int,int> pr;
364    pr.first = 1;
365    pr.second = 10;
366
367    // printing map
368    for(auto it: mpp) {
369        cout << it.first << " " << it.second << endl;
370    }
371
372
373

```

$\text{pair} \langle \text{int}, \text{int} \rangle \text{ fn} = \{1, 2\}$



for(it: mpp) {

```
367
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; xxxxxxx
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 
```

```
397  
398     multimap<string, int> mpp;  
399     mpp.emplace("raj", 2);  
400     mpp.emplace("raj", 5);  
401  
402  
403  
404  
405 // Stack and Queue  
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     References:  
419     cout << prime.cpp:413 // prints 2  
420     st.pop() prime.cpp:414 removes the last entered element  
421     cout << prime.cpp:415 // prints 3  
422     st.pop();  
423     cout << st.top(); → 4  
424  
425
```

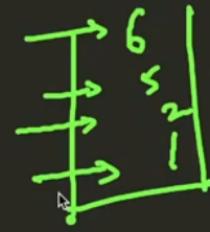


```
406     stack<int> st; // lifo ds
407     // pop
408     // top
409     // size
410     // empty
411     // push and emplace
412
413     st.push(2);
414     st.push(4);
415     st.push(3);
416     st.push(1);
417
418
419     cout << st.top() // prints 2
420     st.pop(); // deletes the last entered 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 false
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
439
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); } // 6
472     pq.push(5); } // 5
473     pq.push(2); } // 2
474     pq.push(6); } // 1
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); } // 1, 5
482     pq.push(1, 6); } // 1, 6
483     pq.push(1, 7); } // 1, 7
484
485
486
487
488
489
490
491
492
493
494

```



```

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; } // -1, -2, -5, -6
486     pq.push(-1);
487     pq.push(-5); } // -1, -2, -5, -6
488     pq.push(-2);
489     pq.push(-6); } // -1, -2, -5, -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

```

```

506     dequeue<int> dq;
507     // push_front()
508     // push_back()
509     // pop_front()
510     // pop_back()
511     // begin, end, rbegin, rend
512     // size
513     // clear
514     // empty
515     // at
516
517
518     list<int> ls;
519     // push_front()
520     // push_back()
521     // pop_front()
522     // pop_back()
523     // begin, end, rbegin, rend
524     // size
525     // clear
526     // empty
527     // at
528     // remove -> O(1)
529     ls.push_front(1);
530     ls.push_front(2);
531     ls.push_front(3);
532     ls.remove(2); -> // O(1) operation
533
534
535
536
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

```

```

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 // output
543 // 3
544
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 << maxi << endl;
560
561
562
563
564
565
566
567
568
569

```

Handwritten annotations:

- mpp[1] = 1**
- maxi** (circled)
- (3, 2)**
- (1, 1)**
- mpp[0]**
- max**
- mpp[3] 2**
- mpp[1] 1**
- D**
- mpp[1] ++**

```

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
562
563
564
565
566
567

```

Handwritten annotations:

- TLF** (Time Limit Exceeded) is circled in red.
- map** is circled in green.
- $O(N \times \log N)$ is written next to the **map** annotation.
- $O(N)$ is written below the first **for** loop.
- $O(N \times 1)$ is written above the **if** condition.
- $O(N \times N)$ is written below the **if** condition.
- TC** (Time Complexity) is circled in red.
- TL** (Time Limit) is circled in red.

```

562
563 // given N elements, print all elements in sorted order
564 // input
565 // n = 6
566 // 6 6 3 2 3 5
567
568 // output
569 // 2 3 3 5 6 6
570
571 int n;
572 cin >> n;
573 multiset<int> ms;
574 for(int i=0;i<n;i++) {
575     int x;
576     cin >> x;
577     ms.insert(x);
578 }
579
580 for(auto it : ms) {
581     cout << it << endl;
582 }
583
584
585
586

```

```
586      // Day 3
587      // Bitset
588      // int -> 16 bits
589      // char -> 8 bits
590
591      int a[100];
592      char a[100];
593
594      // bitset -> 1 bit
595
596      bitset<5> bt; // stores 1 or 0
597      cin >> bt; // 10111
598
599      // all
600      // true // false
601      cout << bt.all(); // returns a true or a false
602
603      // any
604      // true
605      cout << bt.any(); // bt -> 10011
606
607      // false
608      cout << bt.any(); // bt -> 00000
609
610      // count
611      // for bt -> 10100
612      // prints 2
613      cout << bt.count(); // print the number of set bits|    I
614      // flip
615
616      // flip
617      // bt -> 10100
618      bt.flip(2); // bt will become 10000
619
620      bt.flip();
621
622      // none
623      // if none is set, then true, else false
624      // bt -> 10000
625      cout << bt.none(); // false
626
627      // bt -> 00000
628      cout << bt.none(); //true
629
630      // set
631      bt.set(); // 11111
632      bt.set(2); // sets the 2nd index
633
634      bt.set(2, 0);
635
636
637      // reset
638      bt.reset() // turn all indexes to 0
639
640      bt.reset(2); // turn the 2nd index to 0
641
642      // size
643      cout << bt.size(); // prints 5
644
645      // test
646
647      cout << bt.test(1); // check if the bit is set or not at index 1
648
```

```
649 //·Algorithms·
650 //·sorting·
651 //·array,·vector·
652 int n;
653 cin >> n;
654 int arr[n];
655 for(int i = 0;i<n;i++) cin >> arr[i];
656 // takes n log n
657 sort(arr, arr+n); // in increasing order
658 // sort from 1 to 3
659 sort(arr + 1, arr + 3);
660
661 vector<int> vec(5, 0);
662 for(int i = 0;i<n;i++) {
663     cin >> vec[i];
664 }
665
666 sort(vec.begin(), vec.end()); // []
667
668 // vec -> {1, 6, 2, 7, 4}
669 //          0 1 2 3 4
670 // sort it so that only indexes from 1 to 3
671 // final vec -> {1, 2, 6, 7, 4}
672 sort(vec.begin() + 1, vec.begin() + 4); // [1, 4]
673
674
675
676
677
678
679
680
681
682
683
684
685
686
```

```
// 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());
```

```
708  
709     // I give you a range and I want you to find the sum in that range  
710     // i - j, tell me the sum in that range i to j  
711     int sum = 0;  
712     for(int k = i;k<=j;k++) {  
713         sum += arr[k];  
714     }  
715  
716     // accumulate(startIterator, endIterator, initialSum);  
717     int sum = accumulate(arr, arr+n, 0);  
718     int sum = accumulate(vec.begin(), vec.end(), 0);  
719  
720
```

```
720  
721     // arr[] -> [1, 6, 7, 1, 2, 1, 3]  
722     // x = 1  
723     // tell me how many times the element 1 occurs in the array  
724     int cnt = 0;  
725     // O(N)  
726     for(int i = 0;i<n;i++) {  
727         if(arr[i] == x) {  
728             cnt++;  
729         }  
730     }  
731     cout << cnt;  
732  
733     /// count(firstIterator, lastIterator, x)  
734     int cnt = count(arr, arr+n, 1);  
735  
736
```

```

737 // arr[] -> {1, 2, 5, 1, 2, 4, 4}
738 // i want you to find the first occurrence of 2
739 // it is in the index 1
740
741 int ind = -1;
742 for(int i = 0; i < n; i++) {
743     if(arr[i] == x) {
744         ind = i;
745         break;
746     }
747 }
748 cout << ind;
749
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
758
759

```

```

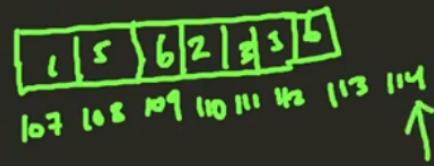
737 // arr[] -> {1, 2, 5, 1, 2, 4, 4}
738 // i want you to find the first occurrence of 2
739 // it is in the index 1
740
741 int ind = -1;
742 for(int i = 0; i < n; i++) {
743     if(arr[i] == x) {
744         ind = i;
745         break;
746     }
747 }
748 cout << ind;
749
750
751
752 // arr[] -> {1, 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
758
759

```

```

744     if(arr[i] == x) {
745         ind = i;
746         break;
747     }
748     cout << ind;
749
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 arrays
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

```

```
791 // lower_bound function
792 // returns an iterator pointing to the first
793 // element which is not less than x
794 // arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
795 // x = 10
796 // 
797 // x < 10
```

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

```
791 // lower bound function
792 // returns an iterator pointing to the first
793 // element which is not less than x
794 // arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
795 // x = 10
796 // x = 6
797 // x = 13
```

```
791 // lower_bound function
792 // returns an iterator pointing to the first
793 // element which is not less than x
794 // arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
795 // x = 10
796 // x = 6
797 // x = 13
798 // this works in log N
799
800 auto it = lower_bound(arr, arr+n, x);
801 ind = it - arr;
802
803 auto it = lower_bound(vec.begin(), vec.end(), x);
804 int ind = it - vec.begin();
805
806 int ind = lower_bound(vec.begin(), vec.end(), x) - vec.begin()
807
808
809
810
811 // upper bound
812 // returns an iterator which points to an element which is
813 // just greater than x
814 // arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
815 // x = 7
816 // x < 7
```

```
810 // upper bound
811 // returns an iterator which points to an element which is
812 // just greater than x
813 // arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
814 // x = 7
815 // x = 6
816
817
818
819

820 // upper bound
821 // returns an iterator which points to an element which is
822 // just greater than x
823 // arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
824 // x = 7
825 // x = 6
826 // x = 12 -> end() iterator
827 // x = 15 -> end() iterator
828

829 // Q1. find me the first index where the element X lies
830 // find function can be used but that takes O(N) times
831 // the array is sorted..
832
833
834 int n;
835 cin >> n;
836 int arr[n];
837 for(int i = 0;i<n;i++) {
838     cin >> arr[i];
839 }
840
841 int x;
842 cin >> x;
843
844 |
845
846
```

```

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     ///////////////////0 1 2 3 4 5 6 7 8 9 10
864     // arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
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 occurrence 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

```

```

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

```

lb ub
 ↓ ↓

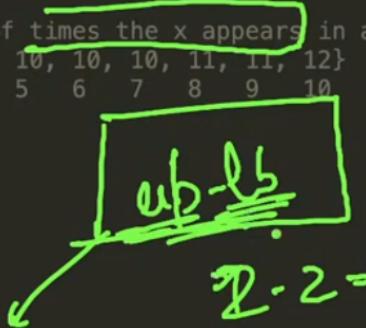
$$\frac{ub-lb}{8-5} = 3$$

```
892  
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
```

$$n=6.$$

ub-lb

$$\Theta(2 \times \log N)$$



$$2-2=0$$

```
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
```

abc
acb
bac
bca
cab
cba

→ sorted

arr[1] → {2, 1, 3}

1, 2, 3
1, 3, 2

```
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 = "cab"  
921  
922  
923  
924  
925  
926
```

bca → true
"bca"
S = "cab"

if it gets
return true

```
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 = "ba" → false  
921     bool res = next_permutation(s.begin(), s.end());  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945
```

~~-it did not get~~
~~other far~~

```
922     // if I give you any random string s = "bca"  
923     // i want you to print all the permutations  
924  
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  
945
```

```
980
981     // COMPARATOR
982     sort(arr, arr+n); // sorts everything in ascending order
983     sort(arr, arr+n, comp);
984
985
986     // descending
987     sort(arr, arr+n, comp);
988     // greater<int> is an inbuilt comparator
989     // which works only if you wanna do this in descending
990     sort(arr, arr+n, greater<int>);
991
992
993
994
995
```

```
15     }
16 }
17 array<int, 3> arr; // -> {0, 0, 0}
18
19 // max size of 10^7 -> int, double, char
20 int arr[10000000];
21
22 // max size of 10^8 -> bool
23 bool arr[100000000];
24
25 bool comp(int el1, int el2) {
26     if(el1 <= el2) {
27         return true;
28     }
29     return false;
30 }
31
32 // arr
33 // pair<int,int> arr[] = {{1, 4}, {5, 2}, {5, 9}},  

34 // i want you to sort this in such a way
35 // that the element who have first element in pair smaller
36 // appears first, and if first is equal then sort according
37 // to second and keep the larger second
38
39
40
```

```
41
42     bool comp(pair<int,int> el1, pair<int,int> el2) {
43         if(el1.first < el2.first) {
44             return true;
45         }
46         if(el1.first == el2.first) {
47             if(el1.second > el2.second) {
48                 return true;
49             }
50         }
51     }
52     return false;
53 }
54
```

w
(1,4)
(5,9)
(5,2)