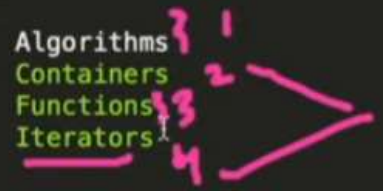


15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48



1	3	2
2	1	2
3	2	3

different algorithms and different
functions that do exist in



1	3	2
2	1	2
3	2	3

```
53
54
55
56
57
58 // Pairs
59 void explainPair() {
60
61     pair<int, int> p = {1, 3};
62
63
64     cout << p.first << " " << p.second;
65
66
67     pair<int, pair<int, int>> p = {1, {3, 4}};
68
69
70     cout << p.first << " " << p.second.second << " " << p.second.first;
71
72
73     pair<int, int> arr[] = { {1, 2}, {2, 5}, {5, 1}};
74
75
76     cout << arr[1].second;
77
78 }
```

functions that do exist in c plus plus
stl so before moving on



1	3	2
2	1	2
3	2	3

```
void explainVector() {
```

$\{ \} \rightarrow \{1\} \rightarrow \{1, 2\}$

```
vector<int> v;  
v.push_back(1);  
v.emplace_back(2);  
vector<pair<int, int>>vec;  
v.push_back({1, 2});  
v.emplace_back(1, 2);
```

$\{1, 2\}$
 $(1, 2)$

```
vector<int> v(5, 100);  
  
vector<int> v(5);  
  
vector<int> v1(5, 20);  
vector<int> v2(v1);  
  
vector<int>::iterator it = v.begin();  
it++;  
cout << *(it) << " ";
```



1	3	2
2	1	2
3	2	3

```
void explainVector() {
```

```
vector<int> v;
```

```
v.push_back(1);  
v.emplace_back(2);
```

```
vector<pair<int, int>>vec;
```

```
v.push_back({1, 2});  
v.emplace_back(1, 2);
```

```
vector<int> v(5, 100);
```

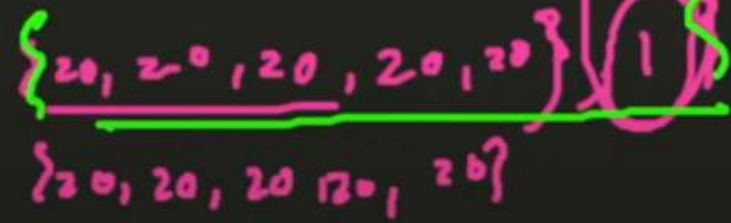
```
vector<int> v(5);
```

```
vector<int> v1(5, 20);  
vector<int> v2(v1);
```

```
vector<int>::iterator it = v.begin();
```

```
it++;
```

```
cout << *(it) << " ";
```




```
110  
117 vector<int>::iterator it = v.begin();
```

```
119  
120 it++;  
121 cout << *(it) << " ";
```

```
123  
124 it = it + 2;  
125 cout << *(it) << " ";
```

```
126  
127 vector<int>::iterator it = v.end();
```

```
128  
129 vector<int>::iterator it = v.rend();
```

```
130  
131 vector<int>::iterator it = v.rbegin();
```

```
132  
133  
134 cout << v[0] << " " << v.at(0);
```

```
135  
136 cout << v.back() << " ";
```

```
137  
138  
139  
140  
141  
142 for (vector<int>::iterator it = v.begin(); it != v.end(); it++) {  
143     cout << *(it) << " ";
```

```
144 }  
145  
146  
147 for (auto it = v.begin(); it != v.end(); it++) {  
148     cout << *(it) << " ";
```

```
149 }  
150
```

V → {20, 10, 15, 5, 7}

0 1 2 3 4

v[1] ← 10

v[3] ← 5

1	3	2
2	1	2
3	2	3

output.txt x



[Finished in 7.3s]

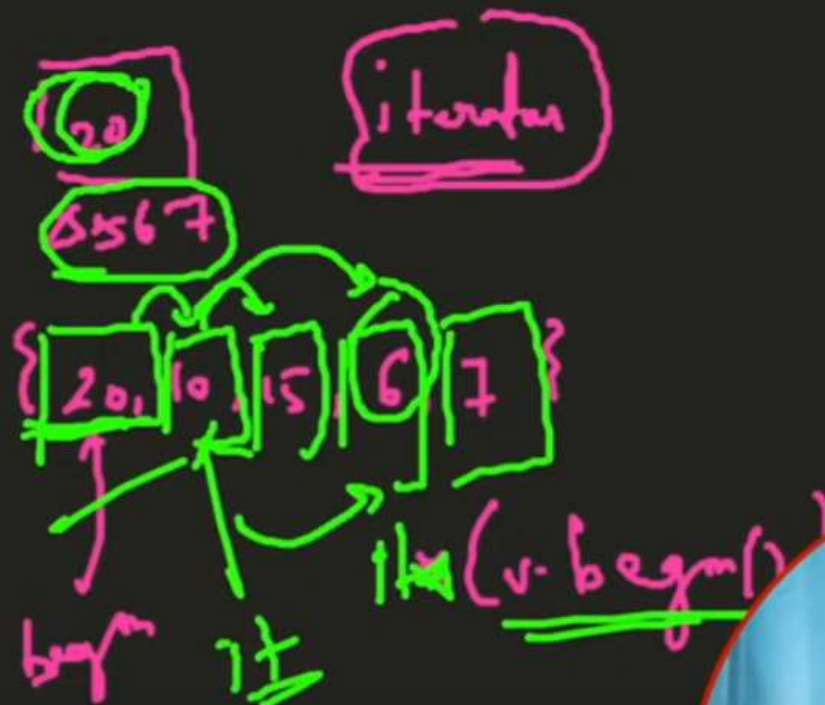
16:17 / 1:07:36 · Vectors >

CC

```

110
117 vector<int>::iterator it = v.begin();
118
119 it++;
120 cout << *(it) << " ";
121
122 it = it + 2;
123 cout << *(it) << " ";
124
125 vector<int>::iterator it = v.end();
126
127 vector<int>::iterator it = v.rend();
128
129 vector<int>::iterator it = v.rbegin();
130
131
132
133
134
135 cout << v[0] << " " << v.at(0);
136
137 cout << v.back() << " ";
138
139
140
141
142 for (vector<int>::iterator it = v.begin(); it != v.end(); it++) {
143     cout << *(it) << " ";
144 }
145
146
147 for (auto it = v.begin(); it != v.end(); it++) {
148     cout << *(it) << " ";
149 }
150

```



1	3	2
2	1	2
3	2	3

output.txt



[Finished in 7.3s]

19:13 / 1:07:36 · Vectors >

CC

```
vector<int>::iterator it = v.begin();
```

```
it++;  
cout << *(it) << " ";
```

```
it = it + 2;  
cout << *(it) << " ";
```

```
vector<int>::iterator it = v.end();
```

```
vector<int>::iterator it = v.rend();
```

```
vector<int>::iterator it = v.rbegin();
```

```
cout << v[0] << " " << v.at(0);
```

```
cout << v.back() << " ";
```

```
for (vector<int>::iterator it = v.begin(); it != v.end(); it++) {  
    cout << *(it) << " ";}
```

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

{10, 20, 30, 40}

1	3	2
2	1	2
3	2	3




```
vector<int>::iterator it = v.begin();
```

```
it++;  
cout << *(it) << " ";
```

```
it = it + 2;  
cout << *(it) << " ";
```

```
vector<int>::iterator it = v.end();
```

```
vector<int>::iterator it = v.end();
```

```
vector<int>::iterator it = v.rbegin();
```

```
cout << v[0] << " " << v.at(0);
```

```
cout << v.back() << " ";
```

```
for (vector<int>::iterator it = v.begin(); it != v.end(); it++) {  
    cout << *(it) << " ";
```

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

{ 10, 20, 30, 40 }



→ it++

{ 40 30 20 10 }



1	3	2
2	1	2
3	2	3

output.txt
1




```
117 vector<int>::iterator it = v.begin();
```

```
118  
119  
120 it++;  
121 cout << *(it) << " ";  
122
```

```
123  
124 it = it + 2;  
125 cout << *(it) << " ";  
126
```

```
127 vector<int>::iterator it = v.end();
```

```
128  
129 vector<int>::iterator it = v.rend();
```

```
130  
131 vector<int>::iterator it = v.rbegin();  
132
```

```
133  
134  
135 cout << v[0] << " " << v.at(0);  
136
```

```
137 cout << v.back() << " ";  
138  
139
```

```
140  
141  
142 for (vector<int>::iterator it = v.begin(); it != v.end(); it++) {  
143     cout << *(it) << " ";  
144 }  
145
```

```
146  
147 for (auto it = v.begin(); it != v.end(); it++) {  
148     cout << *(it) << " ";  
149 }  
150
```

{10, 20, 30}



1	3	2
2	1	2
3	2	3

output.txt
1

```

127 vector<int>::iterator it = v.end();
128 vector<int>::iterator it = v.rend();
129 vector<int>::iterator it = v.rbegin();
130
131
132
133
134
135 cout << v[0] << " " << v.at(0);
136
137 cout << v.back() << " ";
138
139
140
141
142 for (vector<int>::iterator it = v.begin(); it != v.end(); it++) {
143     cout << *(it) << " ";
144 }
145
146
147 for (auto it = v.begin(); it != v.end(); it++) {
148     cout << *(it) << " ";
149 }
150
151 for (auto it : v) {
152     cout << it << " ";
153 }
154
155 // {10, 20, 12, 23}
156 v.erase(v.begin()+1);
157
158 // {10, 20, 12, 23, 35}
159 v.erase(v.begin() + 2, v.begin() + 4); // // {10, 20, 35} [start, end)
160

```



1	3	2
2	1	2
3	2	3

output.txt
1

```

156 v.erase(v.begin()+1);
157
158 // {10, 20, 12, 23, 35}
159 v.erase(v.begin() + 2, v.begin() + 4); // // {10, 20, 35} [start, end)
160
161
162 // Insert function
163
164 vector<int> v(2, 100); // {100, 100}
165 v.insert(v.begin(), 300); // {300, 100, 100};
166 v.insert(v.begin() + 1, 2, 10); // {300, 10, 10, 100, 100}
167
168 vector<int> copy(2, 50); // {50, 50}
169 v.insert(v.begin(), copy.begin(), copy.end()); // {50, 50, 300, 10, 10, 100, 100}
170
171 // {10, 20}
172 cout << v.size(); // 2
173
174 // {10, 20}
175 v.pop_back(); // {10}
176
177 // v1 -> {10, 20} ✓
178 // v2 -> {30, 40} ✓
179 v1.swap(v2); // v1 -> {30, 40} , v2 -> {10, 20}
180
181 v.clear(); // erases the entire vector
182
183 cout << v.empty();
184
185 }
186
187
188 void explainList() {
189     list<int> ls;

```

① → False
② → True

??

1	3	2
2	1	2
3	2	3

output.txt

1



insert

1	3	2
2	1	2
3	2	3

```
203 void explainList() {  
204     list<int> ls;  
205     ls.push_back(2); // {2}  
206     ls.emplace_back(4); // {2, 4}  
207     ls.push_front(5); // {5, 2, 4}  
208     ls.emplace_front(); {2, 4};  
209     // rest functions same as vector  
210     // begin, end, rbegin, rend, clear, insert, size, swap  
211 }  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230 void explainDeque() {  
231     deque<int> ds;
```



output.txt
1


```

227
228
229
230 void explainDeque() {
231
232     deque<int> dq;
233     dq.push_back(1); // {1}
234     dq.emplace_back(2); // {1, 2}
235     dq.push_front(4); // {4, 1, 2}
236     dq.emplace_front(3); // {3, 4, 1, 2}
237
238     dq.pop_back(); // {3, 4, 1}
239     dq.pop_front(); // {4, 1}
240
241     dq.back();
242
243     dq.front();
244
245     // rest functions same as vector
246     // begin, end, rbegin, rend, clear, insert, size, swap
247 }

```

1	3	2
2	1	2
3	2	3

output.txt

1



[Finished in 7.3s]
32:01 / 1:07:36 · DQ >

CC



258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291

```
void explainStack() {  
    stack<int> st;  
    st.push(1); // {1}  
    st.push(2); // {2, 1}  
    st.push(3); // {3, 2, 1}  
    st.push(3); // {3, 3, 2, 1}  
    st.emplace(5); // {5, 3, 3, 2, 1}  
  
    cout << st.top(); // prints 5 "** st[2] is invalid **"  
    st.pop(); // st looks like {3, 3, 2, 1}  
  
    cout << st.top(); // 3  
    cout << st.size(); // 4  
    cout << st.empty();  
  
    stack<int> st1, st2;  
    st1.swap(st2);  
}
```

False



LIFO
Last In First Out
push
pop
top

1	3	2
2	1	2
3	2	3

output.txt
1

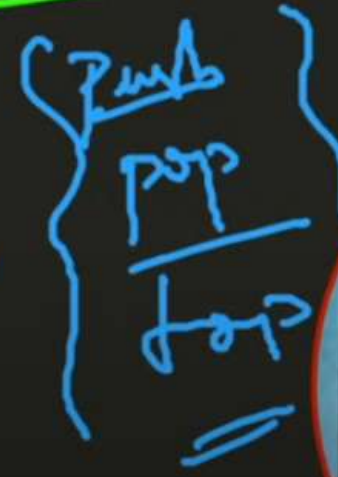


```

258
259
260
261
262 void explainStack() {
263     stack<int> st;
264     st.push(1); // {1}
265     st.push(2); // {2, 1}
266     st.push(3); // {3, 2, 1}
267     st.push(3); // {3, 3, 2, 1}
268     st.emplace(5); // {5, 3, 3, 2, 1}
269
270     cout << st.top(); // prints 5 "** st[2] is invalid **"
271
272     st.pop(); // st looks like {3, 3, 2, 1}
273
274     cout << st.top(); // 3
275
276     cout << st.size(); // 4
277
278     cout << st.empty(); // false
279
280     stack<int> st1, st2;
281     st1.swap(st2);
282
283 }
284
285
286
287
288
289
290
291

```

LIFO
Last In First Out



o(1)

1	3	2
2	1	2
3	2	3



```

288
289
290
291
292
293 void explainQueue() {
294     queue<int> q;
295     q.push(1); // {1}
296     q.push(2); // {1, 2}
297     q.emplace(4); // {1, 2, 4}
298
299     q.back() += 5;
300
301     cout << q.back(); // prints 9
302
303     // Q is {1, 2, 9}
304     cout << q.front(); // prints 1
305
306     q.pop(); // {2, 9}
307
308     cout << q.front(); // prints 2
309
310     // size swap empty same as stack
311 }
312
313
314
315
316
317
318
319
320
321

```

FIFO

First In First out




1	3	2
2	1	2
3	2	3

output.txt

1




```

320
321
322 void explainPQ() {
323     priority_queue<int> pq;
324
325     pq.push(5); // {5}
326     pq.push(2); // {5, 2}
327     pq.push(8); // {8, 5, 2}
328     pq.emplace(10); // {10, 8, 5, 2}
329
330     cout << pq.top(); // prints 10
331
332     pq.pop(); // {8, 5, 2}
333
334     cout << pq.top(); // prints 8
335
336     // size swap empty function same as others
337
338     // Minimum Heap
339     priority_queue<int, vector<int>, greater<int>> pq;
340     pq.push(5); // {5}
341     pq.push(2); // {2, 5}
342     pq.push(8); // {2, 5, 8}
343     pq.emplace(10); // {2, 5, 8, 10}
344
345     cout << pq.top(); // prints 2
346 }
347
348
349
350
351
352
353

```

push
top
pop



1	3	2
2	1	2
3	2	3



output.txt x

GREEN SOUL

CC



[Finished in 7.35s]

38:12 / 1:07:36 • Priority >

```

320
321 void explainPQ() {
322     priority_queue<int> pq;
323
324     pq.push(5); // {5}
325     pq.push(2); // {5, 2}
326     pq.push(8); // {8, 5, 2}
327     pq.emplace(10); // {10, 8, 5, 2}
328
329     cout << pq.top(); // prints 10
330
331     pq.pop(); // {8, 5, 2}
332
333     cout << pq.top(); // prints 8
334
335     // size swap empty function same as others
336
337     // Minimum Heap
338     priority_queue<int, vector<int>, greater<int>> pq;
339     pq.push(5); // {5}
340     pq.push(2); // {2, 5}
341     pq.push(8); // {2, 5, 8}
342     pq.emplace(10); // {2, 5, 8, 10}
343
344     cout << pq.top(); // prints 2
345
346 }
347
348
349
350
351
352
353

```

max heap
push
top
pop

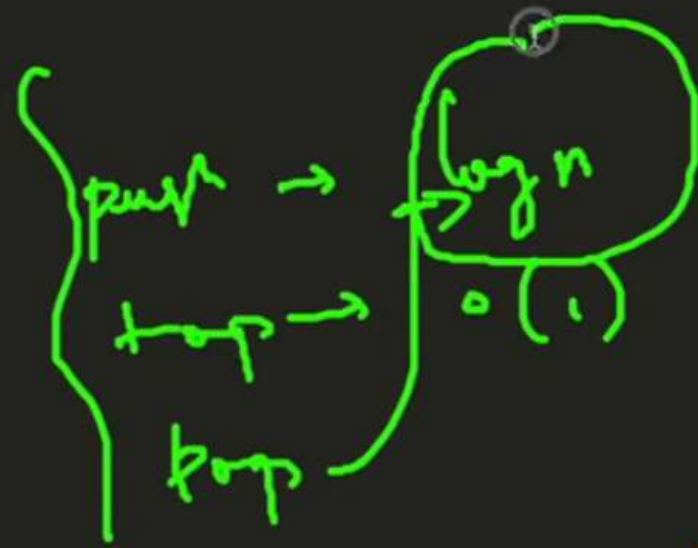


min heap



Snipping Tool

1	3	2
2	1	2
3	2	3



```

317
318
319
320
321
322 void explainPQ() {
323     priority_queue<int> pq;
324
325     pq.push(5); // {5}
326     pq.push(2); // {5, 2}
327     pq.push(8); // {8, 5, 2}
328     pq.emplace(10); // {10, 8, 5, 2}
329
330     cout << pq.top(); // prints 10
331
332     pq.pop(); // {8, 5, 2}
333
334     cout << pq.top(); // prints 8
335
336     // size swap empty function same as others
337
338     // Minimum Heap
339     priority_queue<int, vector<int>, greater<int>> pq;
340     pq.push(5); // {5}
341     pq.push(2); // {2, 5}
342     pq.push(8); // {2, 5, 8}
343     pq.emplace(10); // {2, 5, 8, 10}
344
345     cout << pq.top(); // prints 2
346
347 }
348
349
350

```

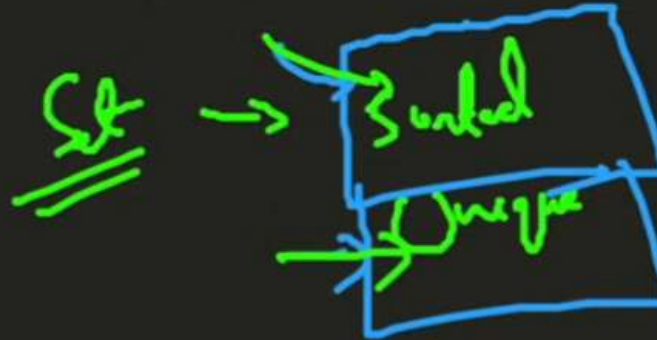


[Finished in 7.3s]


```

350
351
352
353
354 void explainSet() {
355     set<int>st;
356     st.insert(1); // {1}
357     st.emplace(2); // {1, 2}
358     st.insert(2); // {1, 2}
359     st.insert(4); // {1, 2, 4}
360     st.insert(3); // {1, 2, 3, 4}
361
362     // Functionality of insert in vector
363     // can be used also, that only increases
364     // efficiency
365
366     // begin(), end(), rbegin(), rend(), size(),
367     // empty() and swap() are same as those of above
368
369     // {1, 2, 3, 4, 5}
370     auto it = st.find(3); // it is
371
372     // {1, 2, 3, 4, 5}
373     auto it = st.find(6); // st.end()
374
375     // {1, 4, 5}
376     st.erase(5); // erases 5 // takes logarithmic time
377
378     int cnt = st.count(1);
379
380
381     auto it = st.find(3);
382     st.erase(it); // it takes constant time
383

```



Handwritten text: {1, 2, 4, 5}

Handwritten text: st.find(3)

1	3	2
2	1	2
3	2	3



[Finished in 7.35s]

41:24 / 1:07:36 • Set >

CC

Settings