

```

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);
371
372 // {1, 2, 3, 4, 5}
373 auto it = st.find(6);
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
384 // {1, 2, 3, 4, 5}
385 auto it1 = st.find(2);
386 auto it2 = st.find(4);
387 st.erase(it1, it2); // after erase {1, 4, 5} [first, last)
388
389 // lower_bound() and upper_bound() function works in the same way
390

```

Set \Rightarrow unique
sorted

erase(el)
erase(iterator)

1	3	2
2	1	2
3	2	3



[Finished in 7.35s]

42:05 / 1:07:36 • Set >

CC

Binary Search with C++ STL | 4 Problems followed up | Lower Bound and Upper Bound explained
43K views · 2 years ago

FLIP take U forward 152K subscribers

In case you are thinking to buy courses, please check below: Link to get 20% additional Discount at Coding Ninjas:...

Binary search | st | Binary search st | upper bound | lower bound | upper bound st | lower bound st | sorted array | stl | Binary search playlist

lower bound
upper bound

```

366 // begin() end() rbegin() rend() size()
367
368
369
370
371
372
373
374
375 // {1, 4, 5}
376 st.erase(5); // erases 5 // takes logarithmic time
377
378
379 int cnt = st.count(1);
380
381
382 auto it = st.find(3);
383 st.erase(it); // it takes constant time
384
385 // {1, 2, 3, 4, 5}
386 auto it1 = st.find(2);
387 auto it2 = st.find(4);
388 st.erase(it1, it2); // after erase {1, 4, 5} [first, last)
389
390 // lower_bound() and upper_bound() function works in the same way
391 // as in vector it does.
392
393 // This is the syntax
394 auto it = st.lower_bound(2);
395
396 auto it = st.upper_bound(3);
397 }
398
399 void explainMultiSet() {

```



[Finished in 7.35]

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);

20

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(), 4) - a.begin();`

47 upper_bound function:

48
49 `a[] = {1, 4, 5, 6, 9, 9}`

50
51 `int ind = upper_bound(a, a+n, 4) - a;`

52
53 `int ind = upper_bound(a, a+n, 7) - a;`

54
55 `int ind = upper_bound(a, a+n, 10) - a;`

$(\log n)$

56
57

58

59

60

61

62

63 `int ind = upper_bound(a.begin(), a.end(), a) - a.begin();`

64

```

397 }
398
399
400
401
402 void explainMultiSet() {
403     // Everything is same as set
404     // only stores duplicate elements also
405
406
407     multiset<int>ms;
408     ms.insert(1); // {1}
409     ms.insert(1); // {1, 1}
410     ms.insert(1); // {1, 1, 1}
411
412     ms.erase(1); // all 1's erased
413     int cnt = ms.count(1);
414     // only a single one erased
415     ms.erase(ms.find(1));
416
417     ms.erase(ms.find(1), ms.find(1)+2);
418     // rest all function same as set
419 }
420
421
422
423
424
425
426
427
428
429
430

```

sorted

erase (start, end)

erase(1)

find

erase(absolute)

1 1 1 2 3 3 4

1	3	2
2	1	2
3	2	3



```
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 }
```



~~Set~~
(unique)

{ 6 1 2 5 3 }

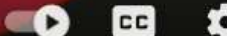
1	3	2
2	1	2
3	2	3



output.txt x

[Finished in 7.35]

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 work, rest all functions are same
438     // as above, it does not store in any
439     // particular order it has a better complexity
440     // than set in most cases, except some when collision happens
441 }
```



~~set~~
unique

1	3	2
2	1	2
3	2	3

min

{ 6 1 2 5 3 }

worst case



0(1)




```
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
```

```
474 mpp.emplace({3, 1});  
475
```

```
476 mpp.insert({2, 4});  
477
```

```
478 mpp[{2, 3}] = 10;  
479
```

```
480 {  
481     {1, 2}  
482     {2, 4}  
483     {3, 1}  
484 }
```

```
485 for(auto it : mpp) {  
486     cout << it.first << " " << it.second << endl;  
487 }  
488  
489
```

```
490 cout << mpp[1];  
491
```

```
492 cout << mpp[5];  
493
```

{key, value}
↑
Is int
double
pair



1	3	2
2	1	2
3	2	3

[Finished in 7.3s]

:10 / 1:07:36 · Map >

```

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

```

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

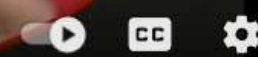
Map → unique
 150



2	1	2
3	2	3

[Finished in 7.3s]

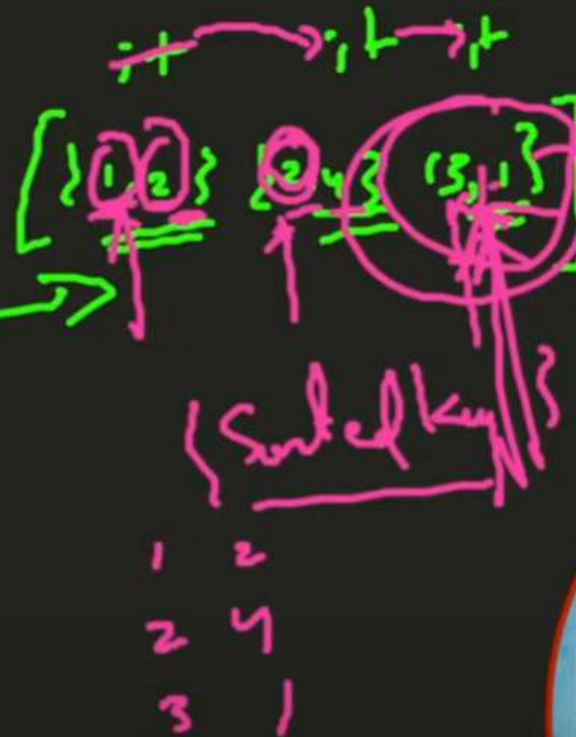
8 / 1:07:36 · Map >



```

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

```



1	3	2
2	1	2
3	2	3

tput.txt

GREEN SOUL

[Finished in 7.3s]

2:40 / 1:07:36 · Map >



1	3	2
2	1	2
3	2	3

```
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
```

Handwritten diagram showing a map entry {3, 1} circled in pink, with an arrow pointing to the code line `mpp.emplace({3, 1});`.




```

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 void explainUnorderedMap() {
510     // same as set and unordered_Set difference.
511 }
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     sort(a+2, a+4);
525
526     sort(a, a+n, greater<int>);
527

```

duplicate keys
sorted

{1, 2}
{1, 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 void explainUnorderedMap() {
510     // same as set and unordered_Set difference.
511 }
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     sort(a+2, a+4);
525
526     sort(a, a+n, greater<int>);
527

```

[Finished in 7.3s]

55 / 1:07:36 • Multimap >

duplicate keys

~~scribbled out text~~

{1, 2}
{1, 3}



1	3	2
2	1	2
3	2	3

```

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 void explainMultimap() {
503     // everything same as map, only it can store multiple keys
504     // only mpp[key] cannot be used here
505 }
506
507 void explainUnorderedMap() {
508     // same as set and unordered_set difference.
509 }
510
511 bool comp(pair<int,int>p1, pair<int,int>p2) {
512     if(p1.second < p2.second) {
513         return true;
514     } else if(p1.second == p2.second){
515         if(p1.first > p2.second) return true;
516     }
517     return false;
518 }
519
520 void explainExtra() {
521     sort(a+2, a+4);
522
523     sort(a, a+n, greater<int>);
524 }

```

~~duplicate keys~~

~~scribbled out text~~

→ can you

{1, 2}
{1, 3}

map
↓
can w

O(1)



1	3	2
2	1	2
3	2	3

```

520
521
522 bool comp(pair<int,int> p1, pair<int,int> p2) {
523     if(p1.second < p2.second) return true;
524     if(p1.second > p2.second) return false;
525     // they are same
526
527     if(p1.first > p2.first) return true;
528     return false;
529 }
530
531 void explainExtra() {
532
533     sort(a, a + n);
534     sort(v.begin(), v.end());
535
536     sort(a+2, a+4);
537
538     sort(a, a+n, greater<int>);
539
540     pair<int,int> a[] = {{1,2}, {2, 1}, {4, 1}};
541
542     // sort it according to second element
543     // if second element is same, then sort
544     // it according to first element but in descending
545
546     sort(a, a+n ,comp);
547
548     // {4,1}, {2, 1}, {1, 2}};
549
550
551     int num = 7;
552     int cnt = __builtin_popcount();
553

```




```
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;
```

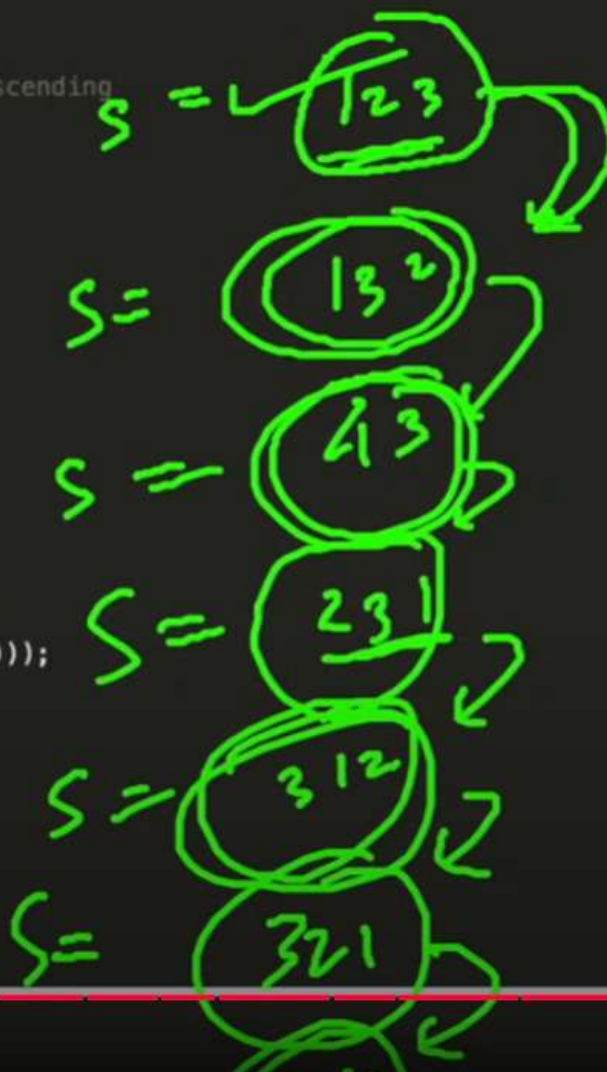
```
} while(next_permutation(s.begin(), s.end()));
```

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

```
int main() {
```

```
    return 0;
```

```
}
```



1	3	2
2	1	2
3	2	3



[Finished in 7.3s]

5:38 / 1:07:36 · Sorting in descending order >

```

540 pair<int,int> a[] = {{1,2}, {2, 1}, {4, 1}};
541
542 // sort it according to second element
543 // if second element is same, then sort
544 // it according to first element but in descending
545
546 sort(a, a+n ,comp);
547
548 // {4,1}, {2, 1}, {1, 2}};
549
550
551
552 int num = 7;
553 int cnt = __builtin_popcount();
554
555 long long num = 165786578687;
556 int cnt = __builtin_popcountll();
557
558
559 string s = "123";
560 sort(s.begin(), s.end());
561
562 do {
563     cout << s << endl;
564 } while(next_permutation(s.begin(), s.end()));
565
566 int maxi = *max_element(a,a+n);
567 }
568
569
570 int main() {
571     return 0;
572 }
573

```

[Finished in 7.3s]

49 / 1:07:36 · Sorting in descending order >

1	3	2
2	1	2
3	2	3



```

540 pair<int,int> a[] = {{1,2}, {2, 1}, {4, 1}};
541
542 // sort it according to second element
543 // if second element is same, then sort
544 // it according to first element but in descending
545
546 sort(a, a+n ,comp);
547
548 // {4,1}, {2, 1}, {1, 2}};
549
550
551
552 int num = 7;
553 int cnt = __builtin_popcount();
554
555 long long num = 165786578687;
556 int cnt = __builtin_popcountll();
557
558
559 string s = "123";
560 sort(s.begin(), s.end());
561
562 do {
563     cout << s << endl;
564 } while(next_permutation(s.begin(), s.end()));
565
566 int maxi = *max_element(a,a+n);
567 }
568
569
570 int main() {
571
572     return 0;
573 }

```

[Finished in 7.3s]

49 / 1:07:36 · Sorting in descending order >

1	3	2
2	1	2
3	2	3

