C++ STL

```
#include<bits/stdc++.h>
using namespace std;
```
↳ C++ has lots of libraries math.h, string.h, everything covered by it.

- namespace raj
```
{
  int val = 50;
}
int main()
{
  double val = 10.0;
  cout<<val<<endl; → o/p: 10.0
  cout<<raj::val<<endl; → o/p: 50
}
```

- struct node
```
{
  string str;
  int num;
  double doub;
  char x;
  node(str_, num_, doub_, x_)
  {
    str = str_;
    num = num_;
    doub = doub_;
    x = x_;
  }
};
int main()
{
  node raj = node("SAM", 79, 10.0, "");
}
```

→ **for Array -**

```
int arr[5] = {1}; → {1,0,0,0,0}
                    0 initialized
int arr[5] = {0}; → {0,0,0,0,0}
                    best way to initialize array w/ 0.
```

but if you want to fill entire array with 1 value -

Container of STL
```
array<int, 3> arr; → if inside main(), {gv, gv, gv}
                   ↳ behave as int arr[3];
∴ arr.fill(10); ↳ {10, 10, 10} ←
```

now,

```
array<int, 4> arr;
arr.fill(1);
```

→ for this we can use – arr.at(index) gives element at index

eg>

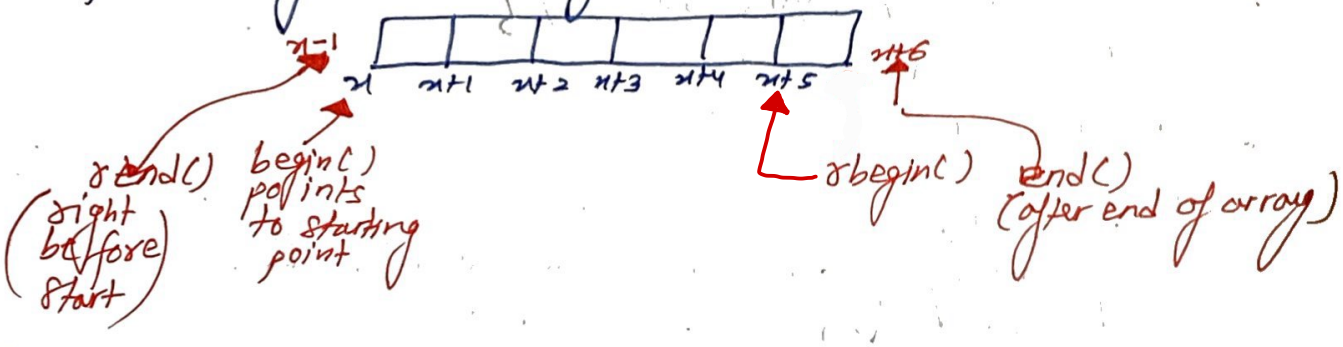
```
for (int i=0; i<4; i++)
{
    cout << arr.at(i) << endl;   → o/p: 1, 1, 1, 1
}
```

Iterators

↳ begin(), end(), rbegin(), rend()] these iterators can be used in all DS.

(rbegin = reverse begin)

for array, it is contiguous



to print

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

(it: a pointer, initially pointing to 1)

rev print

```
for (auto it = arr.rbegin(); it != arr.rend(); it++)
{
    cout << *it << " ";   → 6 5 4 3 1
}
```

(1, 3, 4, 5, 6 — arr.rbegin())

↳ not it--
as rbegin() is for reverse
∴ we automatically it++ krte rev jate rahega

(OR)

```
for (auto it = arr.end()-1; it >= arr.begin(); it--)
{
    cout << *it << " ";   → 6 5 4 3 1
}
```

```
for(auto it : arr) //goes forward
{                  ↳ element itself (assume the datatype)
    cout << it << " "; → 1 3 4 5 6
}
```

for string also-

```
string s = "xheg";
for(auto it : s)
{
    cout << it << " "; → o/p: x h e g
}
```

```
//size
cout << arr.size();
//front [to print front element]
cout << arr.front();   → or, arr.at(0).
//back [to print last element]
cout << arr.back();    → or, arr.at(arr.size()-1)
```

Vector :

Note: Max size of array we can define in C++

```
int arr[10^7]; //globally
int main()
{
    int arr[10^6]; //locally
}
```

for int, double, char array → 10^7 globally, 10^8 locally

for bool array → 10^8 globally, 10^7 locally

```
vector<int> arr; → a size of 0 or empty vector created i.e. { }
arr.push_back(1); → {1}
arr.size();
```

for this ↳ o/p = 0

```
arr.pop_back(); //pop out last element pushed in arr vector
```

→ for max size of vector [same as above: for int, double, char → 10^8 locally, 10^7 globally; for bool → 10^7 locally, 10^8 globally]

- `vector<int> v = {1, 2, 3, 4, 5};`
  (or)
  `vector<int> v{1, 2, 3, 4, 5};`

arr.clear(); // erase all elements at once ie ↓ 4
in vector

- vector<int> vec(4,0); // {0,0,0,0} (0,1,2,3)
[after this also, we can push back]

vector<int> vec(4,8); // {8,8,8,8}

- vector<int> vec1(4,0);
vector<int> vec2(4,8);
vector<int> vec3(vec2.begin(), vec2.end()); // for vec2, from begin() till end(), all elements of vec2 copied to vec3
↳ or, vector<int> vec3(vec2);

- vector<int> vec1;
vec1.push_back(1);
vec1.push_back(2);
vec1.push_back(3);
vec1.push_back(4);
⟶ {1,2,3,4} (0,1,2,3)

now, we want vec1 elements from index = 0 to 1 ie 1 & 2 elements

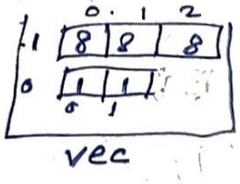
∴ vector<int> vec2(vec1.begin(), vec1.begin()+2);
(start iterator; end() iterator; points to 1; or vec1.end()-2)

- vec.empty();
↳ tells vector is empty → returns T
not empty → " F

- to define 2-D vector-

vector<vector<int>> vec;

vector<int> vec1(2,1); {1,1}
vector<int> vec2(3,8); {8,8,8}
vec.push_back(vec1);
vec.push_back(vec2);



→ here we have vec[1][2] ✓ but not vec[0][2] ✗

for(auto it : vec)
↳ here it is the vector itself
{
for(auto it1 : it)
↳ element of vector
{
cout<<it1<<" ";
}
cout<<endl;
}

isliye iterator use it Kiste hai, at it Know ends of every vector of vector

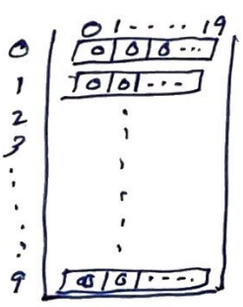
→ o/p: 1 1
8 8 8

or, other way -

```
for(int i=0; i<vec.size(); i++)
{
    for(int j=0; j<vec[i].size(); j++)
    {
        cout<<vec[i][j]<<" ";
    }
    cout<<endl;
}
```

- to define 10×20 array [2D-vector]
  ↳ dimension

```
vector<vector<int>> vec(10, vector<int>(20,0));
```

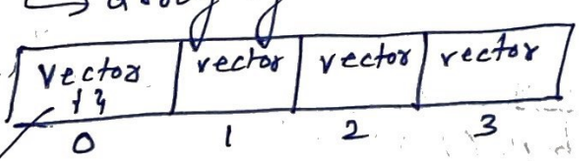


↳ we can do like -
vec.push_back(vector<int>(20,0));

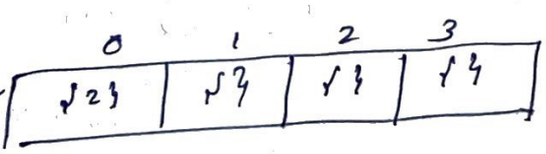
- vector<int> arr[4];
  ↳ array of vector

same cannot do for it

we can do -
arr[1].push_back(0);
(size of array fixed here)
[# of vectors = 4 fixed]



as of now empty vector



∴ arr[0].push_back(2);

but size of vector can be ↑ but not size of arr

- to define 3D vector -

say, 10×20×30

```
vector<vector<vector<int>>> vec(10, vector<vector<int>>(20,
                                vector<int>(30,0)));
```

- vec.front();
  vec.back();
  vec.clear(); it clears all elements in vector, making it size=0

↳ vec.erase() -

To erase 1 element

(I) vec has {1,2,3,4,5}

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

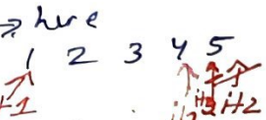
now vec has {2,3,4,5}

To erase range of elements:

(II) vec has {1,2,3,4,5}

```
vector<int>::iterator it1, it2;
it2--;
it2--;
```

→ here




```
vec.erase(it1, it2);
```

{4,5}

if we do -

```
for(auto it : vec)
{
    cout<<it<<" ";
}
```

o/p: 4 5

Set: (I) ordered set -

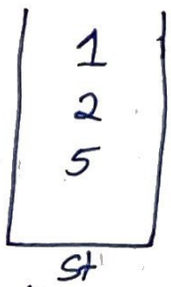
given n-elements, tell me the no. of unique elements?

arr[] = {2, 5, 2, 1, 5} → there are 3 unique elements {1, 2, 5}

to get unique elements -

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

↳ O(log N) (N = size of set)



here, st has {1, 2, 5} in sorted order

1st element of set, 2nd element of set, 3rd ''

we cannot access set as:
st[0] ✗
st[1] ✗

to access set, ←
* st.begin() → gives 1
* st.begin()+1 → '' 2

(OR) can use iterators.

- st {1, 2, 5} say

(1) st.erase(st.begin()); // delete first element from set st

[syntax: st.erase(iterator);]

↳ ∴ st is {2, 5}

(2) st.erase(st.begin(), st.begin()+2); // st.erase(start iterator, end iterator)

for st {1, 2, 5}

begin() = 1, begin()+2 = 2 → erased

∴ 1 2 5

→ st is {5}

[st.erase(....) ⇒ O(log n)]

(3) for st {1, 3, 5}

st.erase(5); // Syntax :: st.erase(key);

↳ st is {1, 3}

if 5 not there, then kch erase nai hoga

- to declare set – [whatever we learnt in vector is applicable over here]

① set<int> st = {1, 5, 7, 8}

set<int> st1(st); // copy set to st1

- to find iterator of an element/key

st {1, 5, 7, 8}
→ points to 7

auto it = st.find(7);
↳ O(log n)

Note: auto it = st.find(Key)

if Key doesnot exist in set st

then it = st.end(); ie 1, 5, 7, 8 ↑ it

- //size

st.size() → gives size of set st

- to store 'node' → first page mein likha hai

∴ set<node> st;

↙ st will have unique node

- to print set –

```
for(auto it = st.begin(); it != st.end(); it++)
    ↳ it is pointer to element
{
  cout<<*it<<" ";
}
```

(or)

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

- to delete entire set –

st.erase(st.begin(), st.end())

(or)

st.clear()

(II) Unordered set - (will have unique elements)

```
unordered_set<int> st;
st.insert(2);
st.insert(3);
st.insert(1);
```

→ don't give guarantee set will be sorted, set can be any order

↙ all the operations is same as ordered set.

for Unordered set → Avg TC is O(1) but WC is O(n) (n = set size)
Ordered set → Avg TC is O(logn)

↳ Note: if you don't want elements in ascending order use unordered set else ordered set

↳ if using this TLE occurs, then samjh jao kch test cases WC = O(n) layak hai ∴ use ordered set

(III) Multiset - (stores duplicates also)

```
multiset<int> ms;
ms.insert(1);
ms.insert(1);
ms.insert(2);
ms.insert(2);
ms.insert(3);
```

→ ms is {1, 1, 2, 2, 3}

↙ multiset helps to store elements in sorted fashion.

↓ all functions used here is same as in ordered set.

↳ Note: for erase:

```
ms.erase(2);
```

→ {1, 1, 3} (all instances of 2 is erased)

- `auto it = ms.find(2);` ↙ returns iterator pointing to first 2 in ms.
- `ms.clear();` ↳ delete entire set

(or)

```
ms.erase(ms.begin(), ms.end());
```

TC for every function in multiset = O(logn) (n = size of multiset)

- `for(auto it : st)` ← element
  `{ cout << it << " ";`
  `}`
  } to print elements in multiset

Note : In ① set → ordered, unordered, multiset

`st.erase( )`
↳ can be ① iterator of an element
② range : start iterator, end iterator
③ Key

② for vector, `v.erase( )`
↳ can be ① ✓
② ✓
③ ✗

- for ms → {1, 1, 2, 3, 3}
  `ms.erase(3);` → delete all instances of 3 ie {1,1,2}
  ↳ to delete/erase only 1 instance of 3, use
  `ms.erase(ms.find(3))`
  ↳ gives iterator of first instance of 3
  → to delete 2, 3, 3
  ∴ `ms.erase(ms.find(2), ms.find(2)+3))`
  1, 1, 2, 3, 3
  erased
- `ms.count(3);` → o/p = 2
  ↳ tells how many times 3 occurs

→ **MAP** (default ordered)
↳ use Key value concept

Ordered map

eg
raj → 27
hima → 31
sam → 1
tank → 89

```
map<string, int> m;
m["raj"] = 27;
m["hima"] = 31;
m["sam"] = 1;
m["tank"] = 89
```

m:
(hima, 31)
(raj, ~~27~~ 29)
(sam, 1)
(tank, 89)
↓ sorted acc to the Keys

here, in lexographical order (as string) Key is

(same as ordered set, maps store in sorted order)

→ if we write again: `m["raj"] = 29` (maps store only unique Keys
↳ overwrites 27 (see table)

- we can write –

```
m["raj"] = 29
```
(or)
```
m.insert({"raj", 45});
```

- 1) `m.erase("raj");` // m.erase(key)
  - just give the key name
  - only single instance in m
  - ↳ delete raj
- 2) `m.erase(m.begin());` // deletes first key
  - ↳ iterator position
- 3) `m.clear();` // delete entire map

(or)
```
m.erase(m.begin(), m.end())
```
- 4) `m.erase(m.begin(), m.begin()+2);`

here, ~~(hima, 31)~~ ~~(raj, 37)~~ deleted (sam, 1) (tank, 89)

- `auto it = m.find("raj");` → if raj does not exist, then it points to last key i.e. to "tank" in m
  - ↳ gives pointer to raj

i.e. it → (raj, 37) (first, second)

∴ `string s = it->first;`
`int x = it->second;`

- To check map is empty or not –
```
if(m.empty())
{
    cout << "Yes it is empty";
}
```
- `m.count("raj");`
  - ↳ always returns 1 as only 1 key as "raj" in map

TC of map = O(log n)

Note – to delete a row from map → use key & not any other values as it is unique

eg

key	age
raj	38
sam	1
tank	37

not unique

- printing map

↳ maps store <string, int> → a pair ie pair<int, int> p;
p.first = 1;
p.second = 20;

∴ for(auto it : m) ↳ pair
{
cout << it.first << " " << it.second << endl;
}

(or)

for(auto it = m.begin(); it != m.end(); it++) ↳ iterator ie address of pair
{
cout << it -> first << " " << it -> second << endl;
}

(II) unordered map ⇒ Unordered map (ordered)

↳ same as map, it uses same function
only thing O(1) in all cases
& O(n) in WC (rare)
↳ size of map

- unordered_map<int, int> m;
↳ does not store in any order (random order)

→ Pair class-

- pair<int, int> p = {1, 2};
p.first is 1 → p.second is 2
- pair<pair<int, int>, int> p = {{1, 2}, 2}
to access this p.first.second
- pair<pair<int, int>, pair<int, int>> p = {{1, 2}, {2, 4}};
p.first.first ⇒ 1 gives
- We can consider pair as an element -

∴ vector<pair<int, int>> v;
set<pair<int, int>> st;
map<pair<int, int>, int> m
↳ key

Note: for unordered_map,
unordered_map<pair<int, int>, int> m;
✗ not possible

** to store pair → use map
↳ unordered map can only store single keys.
↳ same for unordered-set

III multimap → stores everything in sorted order

- multimap<string, int> m;
  - ↳ m["raj"] = 2;
    m["raj"] = 4; → stores duplicates.

Key	val
raj	2
raj	4

[functions used here same as set]

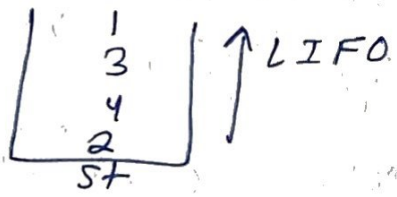
Stack & Queue

I Stack

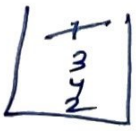
- stack<int> st; //LIFO DS

all operations TC = O(1)

- st.push(2);
- st.push(4);
- st.push(3);
- st.push(1);



- int x = st.top()
  - ↳ ∴ x = 1
- st.pop();



now
int x = st.top()
= 3

- to delete entire stack → TC = O(n) → #elements in stack

Note: bool flag = st.empty(); → returns T or F (empty / not empty)

```
while(!st.empty())
{
  st.pop();
}
```

- st.size(); //gives number of elements as stack

Note:
```
stack<int> st;
cout<<st.top();
```
↳ throw error as stack is empty

∴ use-
```
if(!st.empty()) //always have a check b using st.top()
{
  cout<<st.top();
}
```

II) Queue → All operations O(1)

- queue<int> q;
  q.push(1);
  q.push(5);
  q.push(3);
  q.push(6);

Imp

Note:
No iterator in stack, Queue

- int x = q.front();
  = 1
- q.pop(); → int x = q.front();
  = 5

- to clear queue - TC = O(n) ↳ elements in queue

```
while (!q.empty())
{
    q.pop();
}
```

Priority_Queue - (Heaps)

- set → stored unique elements & sorted order → TC = O(log n) for every functions

unordered set → stored unique elements → TC = O(1) (& WC = O(n) rare)

that's why we use priority queue
↳ store all in sorted order & TC = O(log n)

priority queue has → push()
→ pop()
→ top()
→ empty()
→ size()

uses concept of heap sort

If is Max heap

- priority_queue<int> pq;
  pq.push(1);
  pq.push(5);
  pq.push(2);
  pq.push(6);

→ pq { 6 5 2 1 } ie sorted (in descending order)

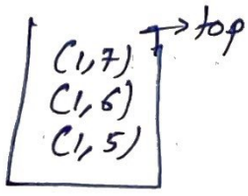
Max element is at top

- `int x = pq.top();`
  `= 6`
- `pq.pop();`
  `int x = pq.top();`
  `= 5`

- if storing pairs in priority queue-

```
priority_queue<pair<int,int>> pq;
pq.push(1,5);
pq.push(1,6);
pq.push(1,7);
```

→ priority queue store acc to pairs

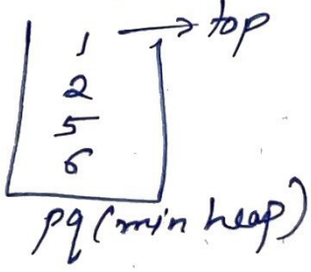


here, first is same for all ∴ sort acc to second

for min heap (if you want pair, replace int w/ pair)

↳ `priority_queue<int, vector<int>, greater<int>> pq;` // syntax for min heap

```
pq.push(1);
pq.push(5);
pq.push(2);
pq.push(6);
```



- `int x = pq.top();`
  `= 1`

List → a doubly LL (behaves same as vectors but here we can push from front & back & pop from front & back)

- `list<int> ls;`
  - 1) push_front()
  - 2) push_back()
  - 3) pop_front()
  - 4) pop_back()
  - 5) begin(), end(), rbegin(), rend() // for iterators
  - 6) size()
  - 7) clear()
  - 8) at()
  - 9) remove() → TC = O(1)
    - ↳ to remove any element

eg)
```
ls.push_front(1);
ls.push_front(2);
ls.push_front(3);
```
→ `ls.remove(2);` → removes all instances of 2

(PTO)

- ls.remove(key) → remove all instances of value = key in list

eg> ls {1, 2, 2, 3, 2}
ls.remove(2);
→ now ls {1, 3}

Question > N elements take from KeyBoard. Print the element that occurs max no. of times.

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

if map used

TC = O(n) * O(logn)
= O(nlogn)

if unordered_map used

TC = O(n) * O(1)
= O(n)

↳ in WC = O(n²) // rare → agar TLE aya used map

→ Note
if m[0] ↳ does not exist
↓
returns NULL
↓
if compared to int, converted to 0

here, m[0] if not exist returns NULL i.e. 0

(1)

Bit Set (only store 1 & 0)

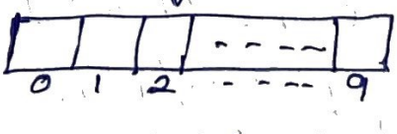
generally, int → 16 bits
char → 8 bits

∴ int a[100]; → 16b * 100
char a[100]; → 8b * 100

// bitset → 1 bit

bitset<10> bt; → creates array like structure

takes less space



It is generally used in segment trees problems as it uses less space.

- bitset<10> bt;
  cin >> bt;
  ↳ give: 1010100011 (possible)

(28)

- it has
  - all()
  - any()
  - count()
  - flip()
  - none()
  - test()
  - size()
  - reset()
  - set()

① say, bt is {1,1,1,1,1}
cout << bt.all(); → o/p: 1
↳ returns true → if all bits are set
false → if any of bits are not set

② bt {1,0,0,0,0}
cout << bt.any();
↳ if any of one bit is set → returns true
else false
↳ if all bits are not

③ bt {1,0,1,1,0}
bt.count(); → gives no. of bits that are set
o/p = 3

④ bt {1 0 1 0 0}
0 1 2 3 4

⑨ bt.flip(2); → o/p: bt becomes {1 0 0 0 0}
↳ flip the bit at 2nd index
(1 → 0, 0 → 1)

⑤ bt.flip(); → converts everything as 1 → 0, 0 → 1
i.e. 1's complement

⑤ bt {1,0,0,0}
bt.none(); → o/p = 0
↳ returns true if no bit are set
returns false if any of bits is set

⑥ set()
- to set entire bt with 1 → bt.set();
  eg bt {1,0,1,0}
  bt.set(); → {1,1,1,1}
- give position/index
  bt {1,0,1,0,0}
  0 1 2 3 4
  bt.set(3); → {1,0,1,1,0};
- bt.set(2,0); → change whatever present at index = 2 with 0.

does not matter whether bt has 0 or 1, it will replace them with 1 but this

- reset()
  - → bt.reset(); → turns all indexes to 0
  - → bt.reset(2); → turns the 2nd index to 0.
  - ~~bt.reset(2,0)~~ ✗ (no such format)
- bt.size(); → #elements in bt
  bt{1,0,1,0} → bt.size() is 5 [but bt.count() tells no. of 1's in bt]
- test()
  → checks if a bit is set or not at index i
  & returns true → if set
  false → if not set

  bt.test(ind);

→ Algorithms -

Sorting -

→ for array, vector

- int n;
  cin>>n;
  int arr[n];
  sort(arr, arr+n); // sort in ↑ order → TC = O(n log n)
- vector<int> vec{1, 5, 6, 7, 2};
  sort(vec.begin(), vec.end()); //[)
  (begin() → 1st iterator, end() → last iterator)
- say vec → {1, 6, 2, 7, 4}
  sort it from indexes 1 to 3 → final o/p: 1, 2, 6, 7, 4
  sort(vec.begin()+1, vec.begin()+4);

Reverse → TC = O(n)

→ array, vector

- reverse(arr, arr+n);
- reverse(start iterator, end iterator);
  ∴ reverse(vec.begin(), vec.end());
- to reverse element in index from 1 to 3 i.e {1, 5, 6, 7, 2} o/p → {1, 7, 6, 5, 2}
  reverse(vec.begin(), vec.begin()+4);

→ To find the max elements in any index range say i to j

normal method

```
int maxi=INT_MIN;
for(int K=i; K<=j ; K++)
    {
      if(a[K] > maxi)
        maxi = a[K];
    }
```

TC=O(n)

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

for vectors: (vec.begin(), vec.end());

↳ returns iterator

for min element,

```
int mini = *min_element(arr, arr+n);
```

↳ TC=O(n)

for vectors: (vec.begin(), vec.end())

→ Given a range, find sum in that range (i to j)

normal method

```
int sum=0;
for(int K=i; K<=j; K++)
    sum += arr[K];
```

```
int sum = accumulate(arr, arr+n, 0);
```

TC=O(n)

for vector- (vec.begin(), vec.end(), 0)

↳ initial sum value can be 10, 20, ...

→ Given arr[] → {1, 6, 7, 1, 2, 1, 3} & x = 1.
Tell how many times 'x' (or 1) occurs in in array.

normal method

```
int cnt=0;
for(int i=0; i<n; i++)
    {
      if(arr[i] == x)
        ++cnt;
    }
```

TC=O(n)

```
count(first iterator, last iterator, x);
∴ int cnt = count(arr, arr+n, x);
```

for vector: (vec.begin(), vec.end(), x);

for 1

→ given arr[] → {1, 2, 5, 1, 2, 4, 4}
(0 1 2 3 4 5 6)

find the first occurrence of 2:
↳ at index = 1

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

↳ auto it = find(arr, arr+n, x); // return pointer pointing to first instance of x if x is present else returns pointer pointing to end() (if not there)

we don't get the index
↓
but since array is contiguous

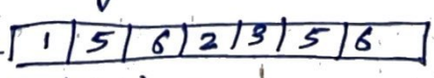
to get index no. -
int ind = (it - arr); → & *ind → get element

for vector:

```
auto it = find(vec.begin(), vec.end(), x);
int ind = it - vec.begin();
```
↳ *ind gives element

- say, vec → {1, 5, 6, 2, 3, 5, 6}
  x = 4

```
auto it = find(vec.begin(), vec.end(), 4);
```



iterator will be at end()

```
∴ if (it == vec.end())
    cout << "element not present";
  else
    cout << "element first occurs at :" << it - vec.begin();
```

Binary Search → TC = O(log n)

↳ works on sorted arrays / vectors [but find() works on every array sorted / not sorted ↳ TC = O(n)]

arr[] → {1, 5, 7, 9, 10}
x = 9

Syntax: binary_search(first iterator, last iterator, x);
↳ element
↳ returns true → if x present
false → if not present

bool res = binary_search(arr, arr+n, 8);
↳ return false;

for vector: (vec.begin(), vec.end(), 8);

→ lower_bound function (for sorted) → implement in BS → TC = O(log n)

↳ returns an iterator pointing to first element which is not less than x.

arr[] → {1, 5, 7, 7, 8, 10, 10, 11, 11, 12}

for x = 10;

for x = 6;

for x = 13; → points to end()

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

for vector: (vec.begin(), vec.end(), x);

vec.begin()

→ upper_bound (for sorted) [implemented in BS & TC = O(log n)]

↳ 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()

x = 15 → same

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

↳ for vector: (vec.begin(), vec.end(), x)

vec.begin()

Q) find the first index (→ first occurrence) where element x lies in arr & arr is sorted.

① if (binary_search(arr, arr+n, x) == true) → O(logn)
{ cout << lower_bound(arr, arr+n, x) - arr; → O(logn)
else cout << "does not exists"; → O(logn) only

Better ②
int ind = lower_bound(arr, arr+n, x) - arr;
if (ind != n && arr[ind] == x)
cout << ind;
else cout << "not found"; ↳ x not present

Q) find me the last occurrence of x in arr & arr is sorted.
eg arr[] → {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
ans

Soln
int ind = upper_bound(arr, arr+n, x) - arr;
if (ind > 0 && arr[ind-1] == x)
cout << ind-1;
else cout << "not found"; ↳ x not present

eg if x = 0
1, 5, 7, 7, 8, ...
UB
∴ (ind-1)
here

if x = 14
1, 5, 7, 7, ..., 11, 12
UB
∴ ind-1
& arr[ind-1] ≠ 14
∴ not found

Q) Tell me the no. of times x appears in arr?
eg arr[] → {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
if x = 10 → ans = 3
x = 7 → ans = 2

Soln
0	1	2	3	4	5	6	7	8	9	10
1	5	7	7	8	10	10	10	11	11	12

for x = 10 → lb = 5, ub = 8
∴ ub - lb = 3

for x = 6
1 5 7 7 8 10 10 10 11 11 12 end()
up, lb, ub
→ ∴ ub - lb = 0

for x = 13
∴ ub - lb = 0

∴ int ans = upper_bound(arr, arr+n, x) - lower_bound(arr, arr+n, x);
(∴ no need to write external checks)
as if element not exist ub - lb = 0 as both points to same

Next permutation → TC = O(n)

String s = "abc"
all permutations are as follows:

abc
acb
bac
bca
cab
cba

↓ all are in lexographical sorted order.

→ we can have array also
i.e arr[] → {2, 1, 3};

∴ 1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

↓ sorted

Say, s = "bca"

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

↳ after this: bca ke baad cab hai
∴ s becomes: s = "cab"

↳ & returns true as it gets to next permutati

if s = "cba"
∴ bool res = next_permutation(s.begin(), s.end());

here, after cba koi nai hai
∴ no next permutation & s remains same but

↓ returns
res = false

Q) given any random string s = "bca"
print all the permutations.

Soln

```
string s="bca";
sort(s.begin(),s.end()); // marks the string "abc"
do{
  cout<<s<<endl;
}while(next_permutation(s.begin(),s.end()));
```

o/p: abc
acb
bac
bca
cab
cba

↓ lexographically sorted

↳ #permutations = O(n!)

TC = O(nlogn) + O(n!)
↙ to sort ↘ do while()

Similarly we have prev_permutation( )
prev_permutation(s.begin(), s.end())
↳ eg?

abc
acb
bac
bca
cab
cba

if s = bac
o/p = acb
∴ in previous page, for prev_perm don't sort

Comparator

- sort(arr, arr+n) // sorts in ascending order

for descending order
↳ we have to write comparator funct'n
↳ returns true or false

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

if (el1 <= el2) → still sorted

consider 2 elements ele1 ele2
if el1 is before el2 return true
else false
(so the 2 element ko sort kisne kar)

```
int main()
{
    sort(arr, arr+n, greater<int>());
}
```

- pair<int,int> arr[] = {{1,4},{5,2},{5,9}};
  // sort in such a way that the element who have first element in pair is smaller
  [if first is equal, sort according to second
  & Keep the larger second
  ↳ o/p: 1,4
  5,9 } **
  5,2 }

as elements are pairs

```
bool comp(pair<int,int> ele1, pair<int,int> ele2)
{
    if (ele1.first < ele2.first)
        return true;
    if (ele1.first == ele2.first) // if equal
    {
        if (ele1.second > ele2.second)
            return true;
    }
    return false;
}

int main()
{
    ...
    sort(arr, arr+3, comp);
    ...
}
```

but if we do-
sort(arr, arr+3);
↳ sorts everything in ascending order
1st (1,4)
(5,2)
(5,9)

We can sort: array, vector, string (Linear DS)