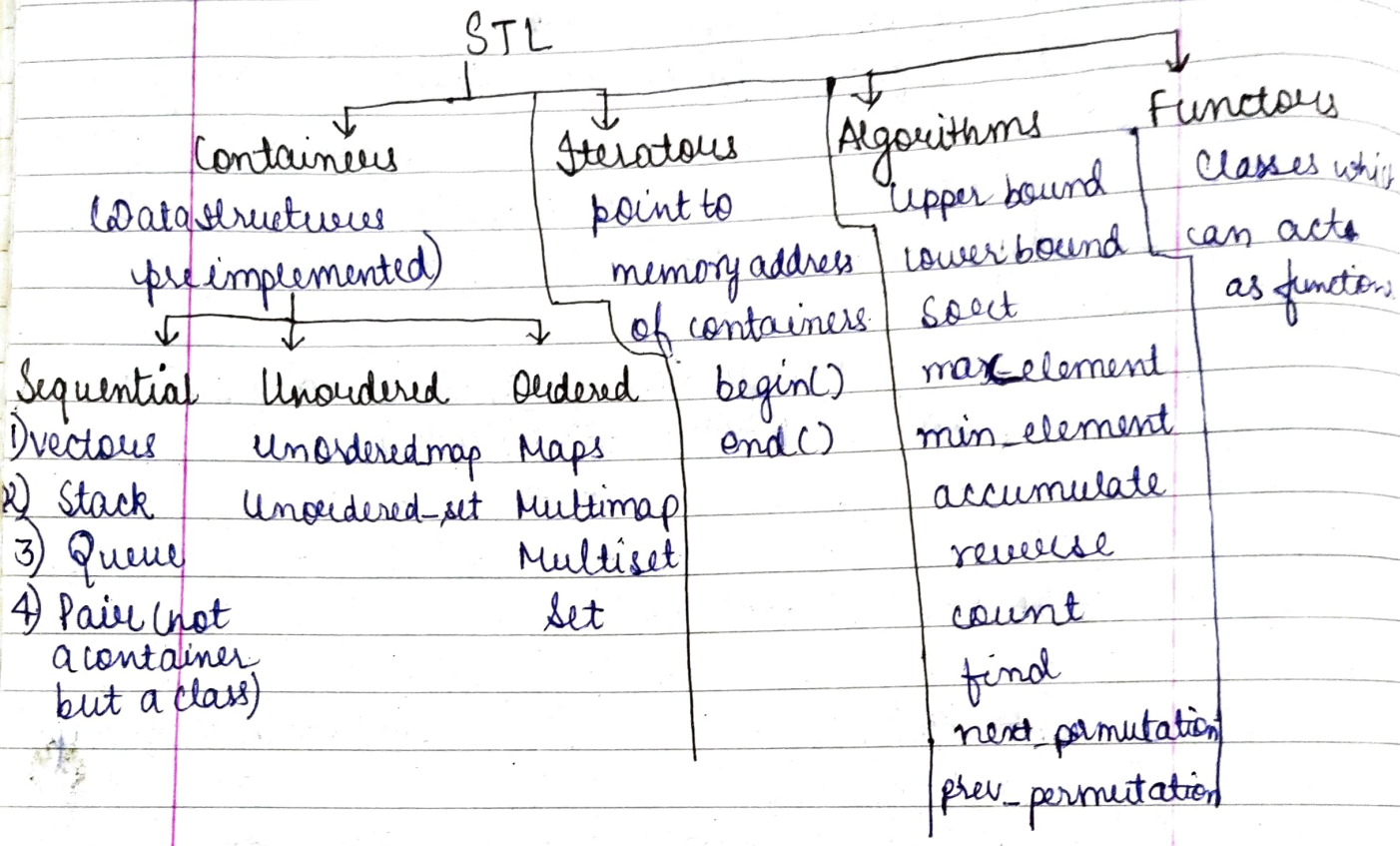
Notes Made By

- Riti Kumari

C++ STL (By LUV)



Pair and Vectors

Pair → class which stores 2 values.

```
pair <int , string > p ;
```

```
pair <1 , 2 > p1 ;
```
(2, 1) → datatype, containers

How to initialise pair ?

```
p = make_pair (2, "abc");
```
or
```
p = { 2 , "abc" };
```

cout << p.first << " " << p.second << endl;

↓ to print first val of pair

↓ to print second val of pair

How to copy pair.

pair<int, string> p1 = p;

pairs are used to maintain relationship betn. 2 containers or 2 datatypes.

```
pair<int, int> p_array[3]; // pair array.
p_array[0] = { 1, 2 };
p_array[1] = { 3, 4 };
p_array[2] = { 2, 3 };

for(int i=0; i<3; i++){
    cout << p_array[i].first << p_array[i].second;
}
```

O/P

1 2
3 4
2 3.

we generally use vector of pairs.

Vectors

Cont^n memory blocks with dynamic size.

```
vector<int> v(10);   (int a[10];)
int n;               (10 size vector)
cin >> n;

for(int i=0; i<n; i++){
    int x;
    cin >> x;
    v.push_back(x); // O(1) push elements at last
}
}

void printVector(vector<int> v){
for(int i=0; i< v.size(); i++){
    cout << v[i] << " ";
}
cout << endl;
}
```

v.size() → O(1)

limits of vector declared locally → 10^5
globally → 10^7

```
vector <int> v(5, 3);
v.push_back(7);
printVec(v);
```

O/P | 3 | 3 | 3 | 3 | 3 | 7 |

```
v.pop_back()  // removes from last O(1)
printVec(v);
```

Copying of vector

```
Vector <int> v2 = v;   // O(n)
```

In functions vector is passed as a copy.

```
void printVec (vector <int> v)
```

so, we pass it using reference to reflect the changes.

```
void printVec (vector <int> &v)
```

Nesting in Vectors

1) Vector of pairs

```
Vector <pair <int, int>> v;
```

v[0] - pair

For initialising directly

```
vector <pair <int, int>> v = {{1, 2}, {2, 3}};
```

to access element

v[i].first & v[i].second

OR

How to insert element

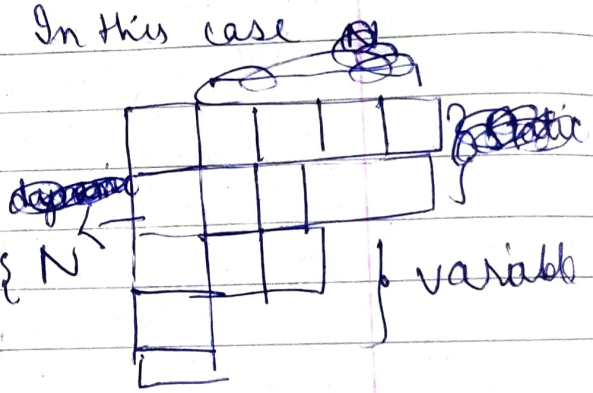
```
for (int i=0; i<n; i++) {
    int x, y;
    cin >> x >> y;
    v.push_back ({x, y});
}
                    or
        (make_pair(x,y));
```

difference betn array of vectors & vectors of vector

array of vectors

```
int main() {
    int N;
    cin >> N;
    vector <int> v[N];

for (int i=0; i<N; i++){
        int n;
        cin >> n;
    for (int j=0; j<n; j++){
            int x;
            cin >> x;
        v[i].push_back(x);
    }
}
```

→ In this case



no of rows → fixed
no of col → variable

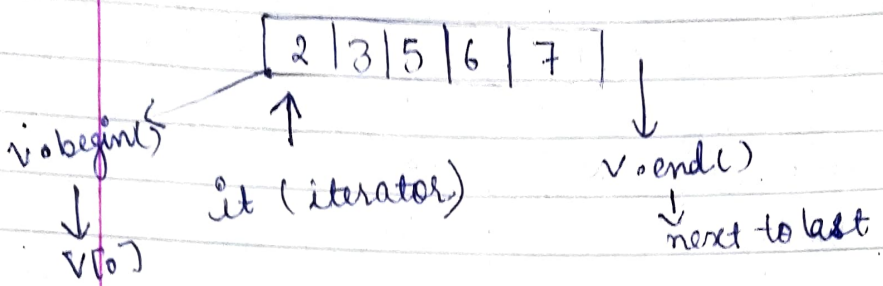
vector of vectors
(for no of rows or columns to be dynamic)

```
int main(){
 int N;
 cin>>N;
 vector<vector<int> V;

for (int i=0; i<N; i++){
     int n;
     cin>>n;
    vector<int> temp;
 for (int j=0; j<n ; j++){
        int x;
        cin>>x;
      temp.push_back(x);
    }
  v.push_back(temp);
 }

for (int i=0 ; i<v.size() ; i++){
        printVec (v[i]);
    }
```

Iterators

Maps doesn't allow random access to any element.

```
int main () {
vector<int> v = {2, 3, 5, 6, 7};
for( int i=0; i<v.size(); i++) {
    cout << v[i] << " ";
}
```



Iterator declaration.

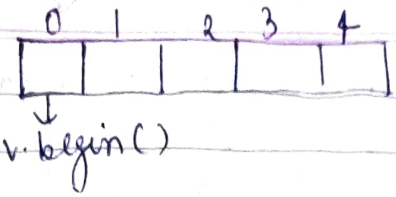
```
vector<int>::iterator it = v.begin();
cout << *it << endl;
```

↓

(acts as pointer & gives value)

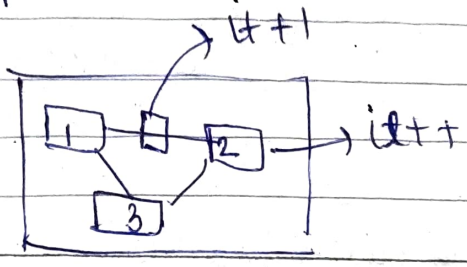
for loop on iterating using iterator.

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



it++ → next iterator
it+1 → next location.

Incase of map



it+1 → invalid operation

	vectors	maps/sets
it++	✓	✓
it+1	✓	X

Iterators in case of pairs

```
vector<pair<int, int>> v_p = {{1,2}, {3,4}};
vector<pair<int, int>>::iterator it;
for(it= v_p.begin(); it!= v_p.end(); it++){
        cout<<(*it).first<<" "<<(*it).second
}
```

(*it).first ⇔ (it→first) (it pointing to a pair)

Range based loops & Auto keyword

Range based loops

```
for (int value : vector_name) {
    cout << value << " ";
}
```

Eg:

```
for (int value : v) {
    cout << value << " ";
}
```

value mei jo value aashe ie. copy, actual values ni aavshi.

```
for(int &value: v){
    cout << value << " ";
}
```

Auto keyword.

Automatically determines the data type.

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

```
for (auto &value : v) {
    cout << value << " ";
}
```

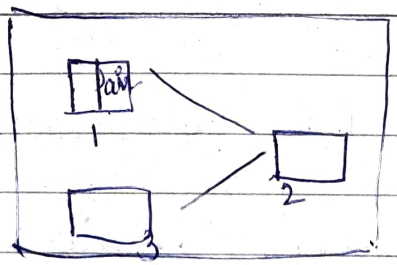
Maps

maps | unordered_map | Multimap

map → DS which store key, value pair. It creates a mapping for key to value.

key	value
int	string
1	abc
5	cde
3	acd

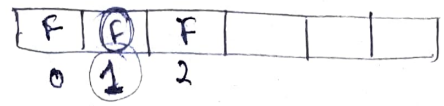
maps → values are stored according to key. (Red Black tree)
unordered_tree → values are stored as it is given.



it++ ✓
it+1 ✗

maps

```
int main(){
  map<int, string> m;
  m[1] = "abc"          // O(log(n))
  m[5] = "cde"
  m[3] = "acd"
```



nums = [1, 2, 1, 3]
0 1 2 3

```
m.insert({4,"afg"});

for(auto it = m.begin(); it != m.end(); it++){
    cout << it->first << " " << it->second;
}
```

OR.

O(nlogn) // `for(auto &pr : m){ cout<<pr.first<<pr.second; }`

O/P

1	abc
3	acd
4	afg
5	cdc.

keys are unique in case of map. The value at given key get replaced.

```
m[5] = "cdc"
m[5] = "cde"
```

O/P 5 cde.

```
auto it=m.find(3);   // it returns an iterator // O(logn)
```

If the value 3 doesn't exists it returns .end()

```
if(it ==m.end()){
    cout << "No value";
}
else {
    cout <<(*it).first << " " <<(*it).second;
}
```

```
m.erase(3); // delete the given key O(logn)
```

we can given value of it also.

```
auto it = m.find(s)
m.erase(it);
print(m);
```

If the iterator doesn't exsist it gives Segmentation fault.

```
m.clear(); // clears the map.
```

Insertion time depends on key.

```
m["abcd"] = "abcd"   O(logn) X
```

↓

It compares with diff strings to get stored in sorted order

$\therefore$ $O(\log n) + s.size();$

Given N strings, print unique strings in lexiographical order with their frequency.

$N <= 10^5$

$|s| <= 100$.

```
int main(){
    map<string, int> m;
    int n;
    cin >> n;
```

```
for(int i=0; i<n; i++){
    string s;
    cin >> s;
    m[s]++;
}
for(auto pr:m){
    cout << m.first << " " << m.second;
```

Unordered_map

differences

hash table ← 1) inbuilt implementation
2) Time complexity
3) valid keys datatypes

} order isn't maintained

```
unordered_map<int, string> m;
    m[7] = "abc";   // O(1) (inserting as well as accessing)
auto it = m.find(7)  // O(1)
```

Only to calculate frequency but order doesn't matter.

unordered_map doesn't allow complex datatypes due to inbuilt implementation.

like unordered_map<pair<int,int>, string> m

pair doesn't have hashtables. In maps.

the values are inserted using comparisions.

int
longlong
float
double
} hash functions defined internally

Given N strings and Q queries. In each query you are given a string, print frequency of that string.

```
int main(){
  int N;
  cin >> N;

  unordered_map<string, int> m;
for(int i=0; i<N; i++){
    string s;
     cin >> s;
   m[s]++;
}

int q;
cin q;
while(q--){
   string s;
   cin >> s;
cout << m[s] << endl;
```

Multimap (Red black tree)

No unique keys

```
multimap < int , string > m;
```

but we don't use multimap we use

```
map < int , vector <string>> m;
```

Set, Unordered_set, Multiset

Sets → In map we store pair (key & value) but in set we store only key but they are unique. (Stores in sorted order)

```
Set < string > S;
S.insert ("abc");          // O(logn)
S.insert ("zsdf");
S.insert ("bcdf");
```

No random access is their, we use find.

```
auto it = S.find("abc");

for (string v : S){          // log(n)
    cout << v << endl;
```

S.erase("abc"); // can take iterator or element in i/p.

Q. Given N strings, print unique strings in lexiographical order.

$N <= 10^5$.
$|S| <= 100000$.

```
int main() {
    set<int> sl;
    int N;
    cin>>N;
  for(int i=0; i<N; i++){
      string s;
      cin>>s;
      sl.insert(s);
    }

  for(auto v:s){
      cout<<v<<endl;
    }
```

Unordered_set O(1)

```
unordered_set<string> s;
```

Q. Given N string & Q queries. find if the string is present or not?

```
int main() {
unordered_set <string> s;
  int n;
  cin >> n;
for(int i = 0; i < n; i++) {
        string str;
        cin >> str;
        s.insert(str);
  }
 int q;
 cin >> q;
  while(q--) {
    string str;
    cin >> str;
   if(s.find(str) == s.end()) {
        cout << "no";
    }
    else {
        cout << "yes";
        }.
```

Multi set → (Allows multiple values)
↓ // O(logn)
used in place
of priority-queues.

If duplicates are found it returns the iterator of first string present.

```
auto it = s.find("abc");
if (it != s.end()){
    s.erase(it);
}
```

i/p

```
abc
abc
dbg
```

o/p

```
abc
dbg.
```

but if we use.

```
s.erase("abc"); → it find iterator until it doesn't find all the value.
print(s);
```

i/p

```
abc
abc
dbg
```

o/p

```
dbg.
```

Nesting in STL

In case of

```
map <pair<int, int>, int> m;
```

first p.first is checked then if both are same. then p.second is checked.

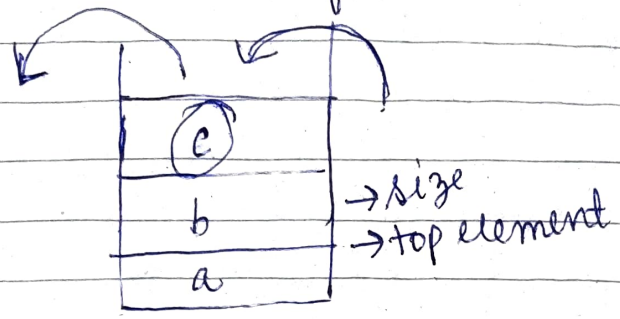
2
a b 4
1 2 3 4
c d 2
1 2

```
map < pair < string , string >, vector < int >> m;
int n;
cin>>n;
for(int i= 0; i<n ;i++){
  string fn, ln ;
  cin>> fn >>ln ;
  int ct ;
  cin>> ct;
                    (t
for(int j= 0 ;j<t , j++ ){

    int x;
      cin>>x;

      m[{fn, ln}].push_back (x);
  }
}

for (auto &pr:m) {
    auto &full_name = pr.first;
    auto & list = pr.second;
  cout << full_name.first << " " << fullname.second
    cout << list.size() << endl;
     for( auto & e : list) {
          cout << e << " ";
     }
```

vector (1 2 3 4)	vector (1,2)
pair {a,b}	pair {c, d}

Stacks & Queues

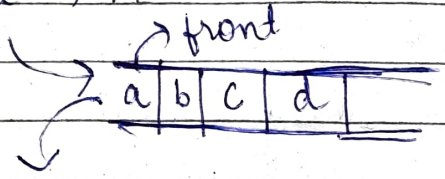
Stack → LIFO DS
(last in first out)



i/p → a b c
o/p → c b a

① push ii) pop iii) Top

Queue → FIFO



i/p → a b c d
o/p → a b c d

① push ii) pop (front element) iii) front

```
int main() {
    Stack <int> s;
    s.push(2);
    s.push(3);
    s.push(4);
    s.push(5);

    while (!s.empty()) {
        cout << s.top() << endl;
        s.pop();
    }
```

o/p - 5
4
3
2

```
int main(){
queue <string> q
  q.push("abc");
  q.push("bcd");
  q.push("cde");
  q.push("def");
  q.push("ghi");

while(!q.empty()){
    cout << q.front() << endl;
    q.pop();
  }
}
```

o/p
abc
bcd
cde
def
ghi

Balancing parenthesis

Check if a given string is valid or not.

(((()))) → yes

have opening as well as closing

(((

((()

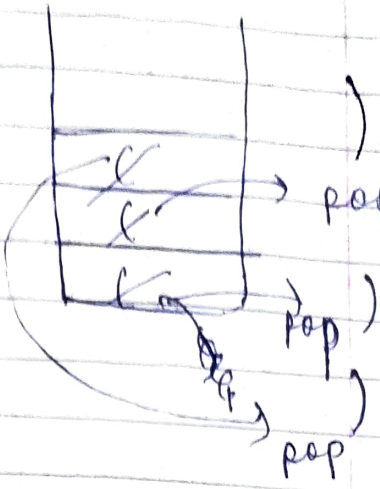
(((()))(()

} Not valid

String : (())()

We will put opening braket in stack

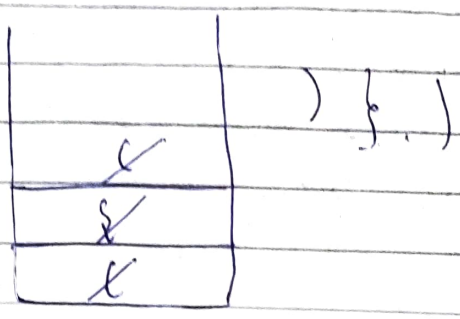
(( )) ( )



if stack in empty after all the traversal it is balanced string

ii) when we have different type of parenthesis

({ ( ) })



Using stack O(N)

```
unordered_map<char, int> symbols = {{'[', -1}, {'(', -2}
                                    {{'{', -3},
                                    {{']', 1}, {')', 2},
                                     {'}', 3}};
String isBalanced (string s) {
    Stack<char> St;
for (char braket : s) {
    if ( symbols[braket] < 0) {
        St.push (braket);}
    else {
        if ( st.empty()) return "No";
        char top = St.top();
        St.pop();
```

```
if ( symbols(top)+symbols[bracket]!=0) {
        return "No" ;
      }
  }
}
if (st.empty()) return "Yes";
  return "No" ;
}

int main (){
   int t;
   cin >> t;
    while (t--){
       String s;
       cin >> s;
       cout << isBalanced(s)<< endl;
    }
}
```

Next Greater Element

4, 5, 2, 2 5, 7, 8.

4	5	2	2 5	7	8

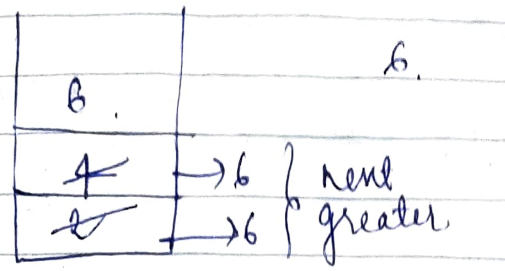
NGE

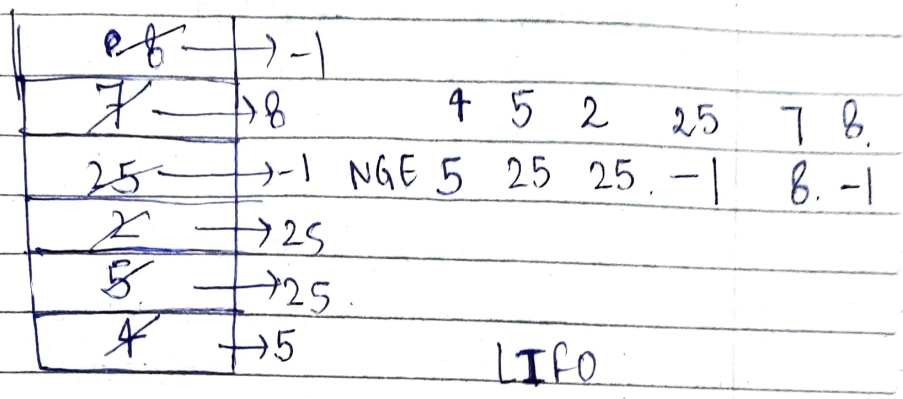
5	25	25	-1	8	-1

Using stack we can do in O(N)

4, 5, 2, 25, 7, 8.

general example





We will put indexes in stack.

```
vector <int> NGE (vector <int>v) {
    vector <int> nge (v.size());
    Stack <int> st;
for (int i= 0; i< v.size(); ++i) {
  while (! st.empty() && v[i] > v[st.top()]
        nge [st.top()] = i;
        st.pop();
    }
    St.push(i);
}
```

```
while ( ! st.empty() ) {
  nge[st.top()] = -1;
    st.pop();
}
return nge;

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

  vector<int> nge = NGE(v);
for(int i=0; i<n; i++){
cout << v[i] << " " << (nge[i]==-1 ? -1 : v[nge[i]])
                                    << endl;
}
```

Inbuilt sort (introsort)

↓

Quick + heap + insertion

```
intmain(){
 int n;
 cin >> n;
 int a[n];
for(int i=0; i<n; i++){
      cin >> a[i];
}
 sort ( a, a+n);
```

```
for (int i = 0; i < n; i++){
    cout << a[i] << " ";
}
```

In case of vector

```
vector <int> a(n);

sort (a.begin(), a.end());
```

Complexity : O(n log n)

for in built sorting algo

Comparator function

→ Swap krna hai to → false
Swap nikrana to → true

Taking example of a pair in which we need to sort the first in ascending order. If two values are equal then sort second in descending order.

```
bool comp ( pair <int, int> a, pair <int, int > b){
    if (a.first != b.first ) {
        return a.first < b.first;
    }
```

Jaise chahe waise likh do

```
int main() {
    int n;
    cin>>n;
    vector <pair <int, int>> a(n);
```

```
for(int i=0; i<n; i++)
    cin >> a[i].first >> a[i].second;

sort(a.begin(), a.end(), cmp);

for(int i=0; i<n; i++){
    cout<<a[i].first<<a[i].second<<" ";
    }
cout<<endl;
```

by default comparator

greater<pair<int,int>>()

Upper bound & lower bound
(only works on sorted array)

4 5 5 7 8 25

lower bound of 7: ya-to 7 present hai to wo
ni toh phir 8

upper bound of 7: 8 hi hoga hmesha

lower bound of 6 = 7
lower bound of 26 = next element ka pointer
upper bound of 5 = 7

lower_bound & upper_bound return pointer or iterator. // O(log n)

```
int main() {
    int n;
    cin>> n;
    int a[n];
  for(int i=0; i<n; i++) {
        cin>>a[i];
    }
    Sort (a, a+n);
  for (int i=0; i<n; i++) {
        cout << a[i] << "  ";
    }
    cout <<endl;

    int *ptr = lower_bound (a, a+n, 5);   // 5
    cout <<(*ptr) << endl;
    int *ptr1 = upper_bound (a, a+n, 5);  // 7
    cout <<(*ptr1) << endl;
```

i/p → 6 4 5 5 25 78

for vector

```
auto it = lower_bound (a.begin(), a.end(), 5);
```

for sets & maps — lower_bound & upper_bound works in O(N)

O(N) `auto it = lower_bound( s.begin(), s.end(), 5);`

```
auto it = s.lower_bound (5);
```

Maps. (in case of maps it works for keys only)

C++ STL (Inbuilt Algorithms)

① min_element() & max_element

min_element (start, end+1);

returns ptr

int min = *min_element (v.begin(), v.end())

int max = *max_element (v.begin(), v.end());

↳ to find the value since it returns the ptr

② accumulate() - sum of array.

accumulate (start, end+1, initial sum)

(returns value)

int sum = accumulate(v.begin(), v.end(), 0);

③ count() → count of elements.

↓

returns value

int ct = count (v.begin(), v.end(), 2);

④ find() → find the element & return the array.

int fa = find (v.begin(), v.end(), 2);

a. To find if a particular element exists in array or not.

```
auto it = find (v.begin(), v.end(), 10);
if (it != v.end())
    cout << (*it) << endl;
else
    cout << "Element not found";
```

5) reverse() - reverse the string, array.

for array
```
reverse (v.begin(), v.end());
for (auto val : v) {
    cout << val << " ";
}
cout << endl;
```

for string
```
string s = "abcdefgh";
reverse (s.begin(), s.end());
cout << s << endl;
```

(Lamba functions & STL Algorithms)

all_of none_of any_of (returns true or false)

all_of → check a condition on every element without using loop.

all_of (start , end+1 , lamba function)

Lambda funct → small function

cout << [ ] (int x) { return x+2; } (2)

func.

auto sum=
[ ] (int x, int y) {return x+y ; }.

cout << sum(2,3)

int main() {
vector <int> v={ 2,-3,-5 };
cout << all_of (v.begin(), v.end(), [ ]int x {return x>0;});

O/p → 0

cout << any_of (v.begin (), v.end (),
[ ] int x { return x>0; }) << endl

O/p → 1

cout << none_of (v.begin (), v.end (),
[ ] int x { return x>0; }) << endl;

O/p → 0