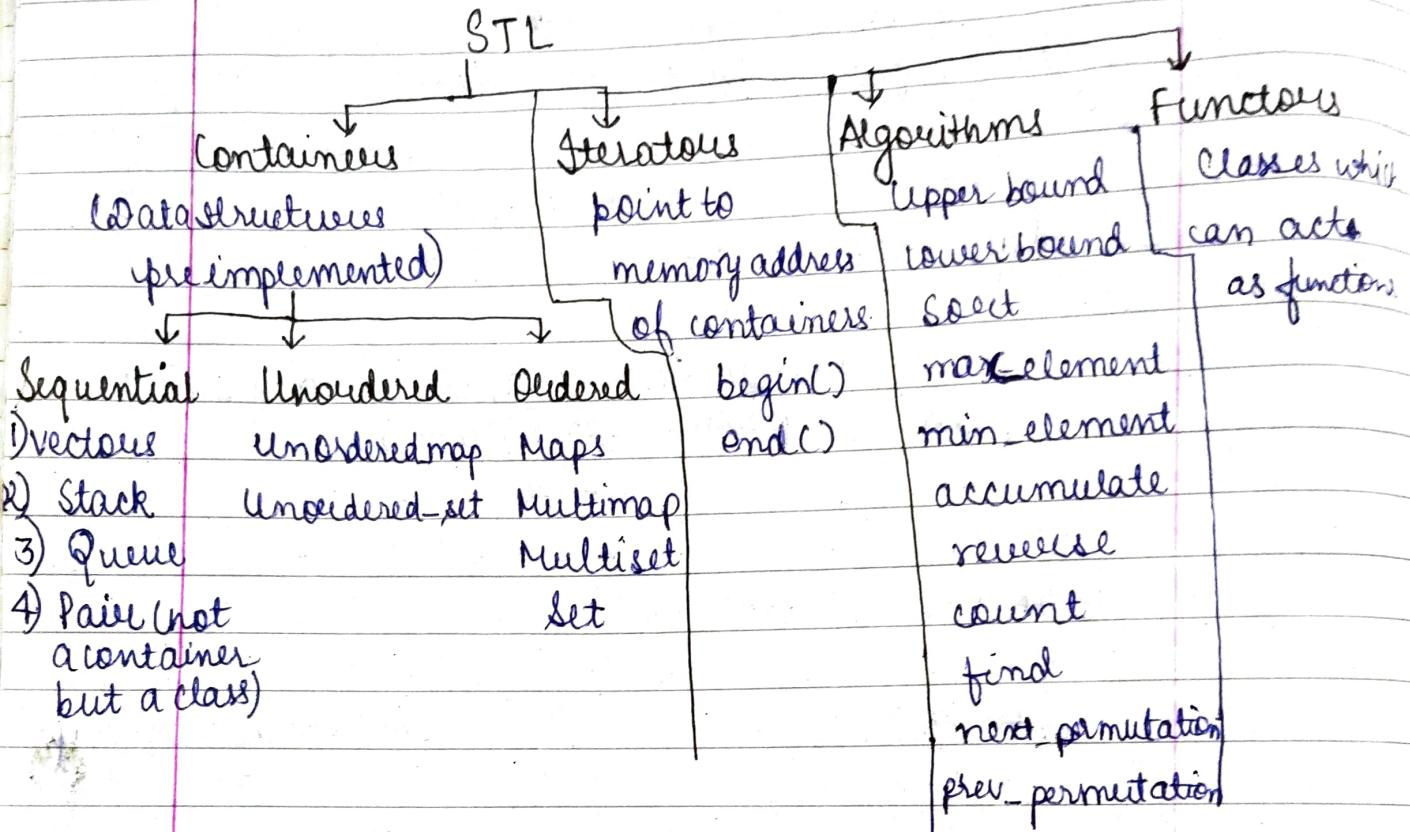


Notes Made By

- RITI Kumari

C++ STL (By LUV)



Pair and Vectors

Pair → class which stores 2 values.

pair<int, string> p;

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

How to initialize pair?

p = make_pair(2, "abc");
or

p = {2, "abc"} ;

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

↓ ↓

to print to print
first val second val
of pair of pair

How to copy pair.

pair<int, string> p1 = p;

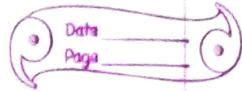
pairs are used to maintain relationship betw.
2 container or 2 datatypes.

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

Contⁿ memory blocks with dynamic size.

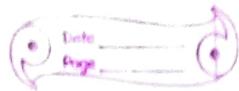
```
vector<int> v(10); (int a[10];)  
int n; // 10size vector  
v[0]  
cin >> n;
```

```
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);  
v.pop
```

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

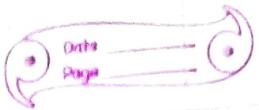
i) Vector of pairs

```
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

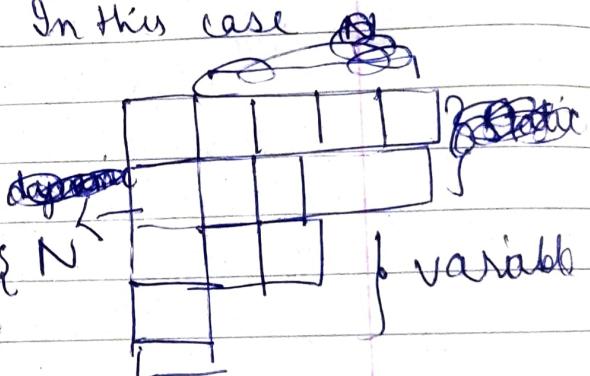
```
for (int i=0; i<n; i++) {  
    int x, y;  
    cin >> x >> y;  
    v.push_back ({x,y});  
    // or  
    (make_pair(x,y));
```

difference bet^n' array of vectors & vectors of vector

array of vectors

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

→ In this case



```
for (int i=0; i<N; i++) { N
```

int n;

cin >> n;

```
    for (int j=0; j<n; j++) { }
```

int x;

cin >> x;

v[i].push_back (x);

}

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;
            int x;
            cin >> x;
            temp.push_back(x);
        }
        v.push_back(temp);
    }

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

```

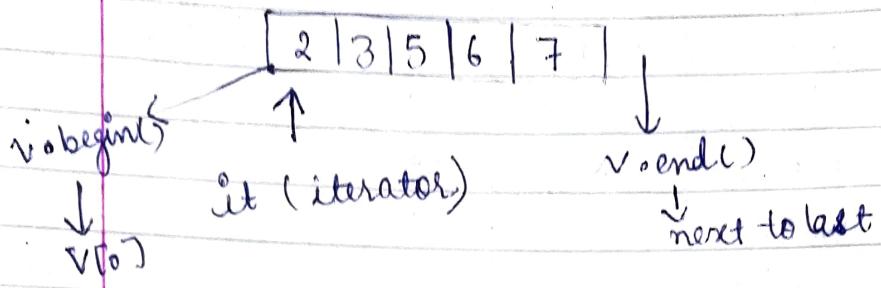
Iterators

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;

```

\downarrow

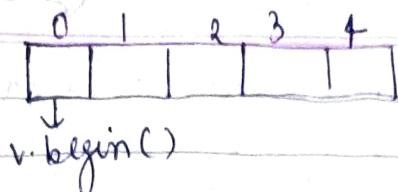
(acts as pointer &
gives value)

for loop all iterating using iterator.

```

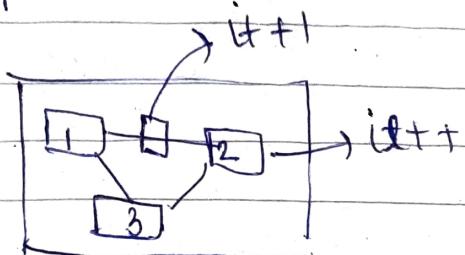
vector<int>::iterator it = v.begin();
for (it = v.begin(); it != v.end(); it++) {
    cout << (*it) << endl;
}

```



$it++ \rightarrow$ next iterator
 $it+1 \rightarrow$ next location.

Incase
of map



$it+1 \rightarrow$ invalid operation

| | vectors | maps/sets |
|--------|---------|-----------|
| $it++$ | ✓ | ✓ |
| $it+1$ | ✓ | X |

2 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  

}
```

$(*it).first \Leftrightarrow (it \rightarrow \text{first})$ (it pointing to a pair)

Range based loops & Auto key word

Range based loops

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

Eg:

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

value unhe jo value aashe i.e. copy, actual values ni aashi.

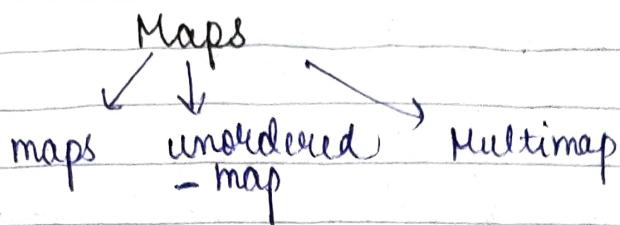
```
for (int & value : v) {
    cout << value << " ";
}
```

Auto keyword.

Automatically determines the data type.

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

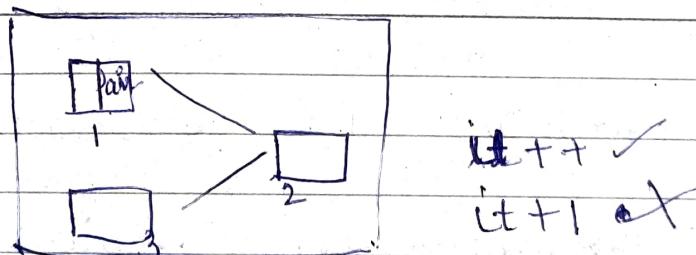
```
for (auto & value : v) {
    cout << value << " ";
}
```



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

| key | value |
|-----|--------|
| int | string |
| 1 | abc |
| 5 | cde |
| 3 | acd |

(red black tree) maps → values are stored according to key.
unordered tree → values are stored as it is given.



maps

```

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

| | | | | | | |
|---|---|-----|---|--|--|--|
| m | F | (F) | F | | | |
| 0 | 1 | 2 | | | | |

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



m.insert ({1, "afg"})

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

OR.

O(nlogn) || for (auto & pr : m) {

O/P

| | | |
|---|-----|--------------------------------|
| 1 | abc | cout << pr.first << pr.second; |
| 3 | acd | |
| 4 | afg | |
| 5 | cde | |

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

m[5] = "dc"

m[5] = "cde"

O/P S 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 give value of it also.

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

If the iterator doesn't exist it gives segmentation fault.

m.clear(); It clears the map.

Insertion time depends on key.

$m["abcd"] = "abcd"$ $O(\log n) \times$

↓
It compares with diff strings to get stored in sorted order

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

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

$$N \leq 10^5$$

$$|S| \leq 100$$

```
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 datatype.
- } 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 datatype due to inbuilt implementation.

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

pair doesn't have hash tables. In maps.

the values are inserted using comparisons.

```
int  
long long } hash functions  
float  
double }
```

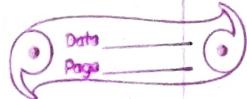
defined internally

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"); $\text{II } O(\log n)$

s.insert("zsd");

s.insert("bcd");

No random access is there, we use find.

auto it = s.find("abc");

for (string v : s) { $\text{II } \log(n)$
cout << v << endl;}

S.erase ("abc"); // can take iterator see
movement in i, p.

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

$$N \leq 10^5$$

$$|S| \leq 100000$$

```
int main() {
    set<int> s1;
    int N;
    cin >> N;
    for (int i = 0; i < N; i++) {
        string s;
        cin >> s;
        s1.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 \rightarrow (Allows multiple values)

\downarrow $O(n \log n)$
 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 finds iterator until
print(s); it doesn't find all
the value.

I/P

abc
abc
dbg

O/P

dbg.

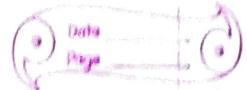
Nesting in STL

In case of

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

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

$\begin{array}{c} a \ b \ 4 \\ | \ 2 \ 3 \ 4 \\ c \ d \ 2 \\ | \ 2 \end{array}$



```

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

```

~~int x;~~
~~cin >> x;~~

```

}
m[{fn, ln}] = pushback(x);

```

```

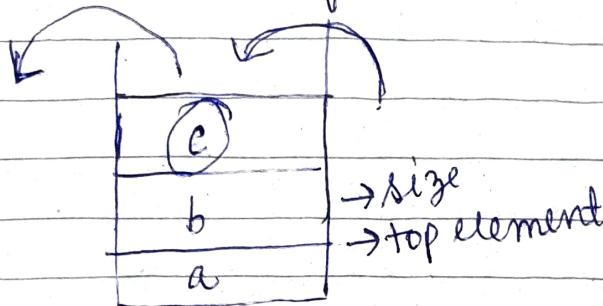
for(auto & pr:m) {
    auto & fullname = pr.first;
    auto & list = pr.second;
    cout << fullname.first << " " << fullname.second;
    cout << list.size() << endl;
    for(auto & e : list) {
        cout << e << endl;
    }
}

```

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

Stacks & Queues

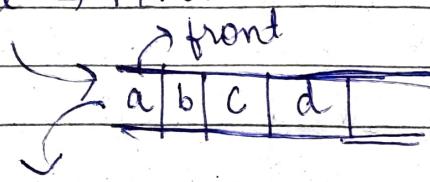
Stack → LIFO DS
(last in first out)



i/p → abc,
o/p → c.b.a

- i) push
- ii) pop
- iii) Top

Queue → FIFO



i/p → abcd
o/p → abcd

- i) push
- ii) pop
(front element)
- iii) front

int main() {

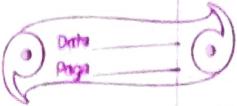
```
Stack<int> s;
s.push(2);
s.push(3);
s.push(4);
s.push(5);
```

O/P - $\frac{5}{4}$
3
2

while (!s.empty()) {

```
cout << s.top() << endl;
s.pop();
```

}



int main() {

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

O/P abc
 bcd
 cde
 def
 ghi

while (!q.empty()) {

```
cout << q.front() << endl;
q.pop();
```

}

Balancing parenthesis

Check if a given string is valid or not -

() () () () → yes

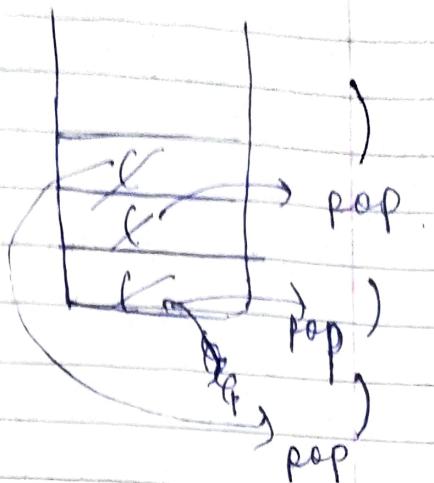
have opening as
well as closing

| | | |
|-----------|---|--------------|
| (((| } | |
| (((| | Not valid |
|)) (() | | |

String : (()) ()

We will put opening bracket in stack

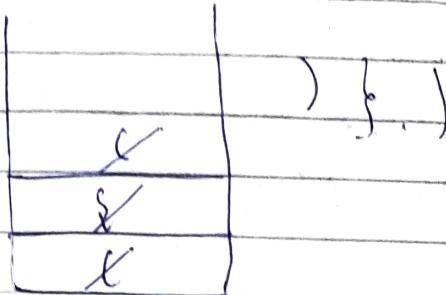
(()) ()



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

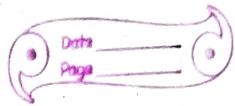
ii) when we have different type of parenthesis

({ () })



Using stack O(N)

```
unordered_map<char, int> symbols = {{'(', -1}, {'{', -2},  
String.isbalanced (String s) {  
    stack<char> st;  
    for (char bracket : s) {  
        if (symbols[bracket] < 0) {  
            st.push(bracket);  
        }  
        else {  
            if (st.empty()) return "No";  
            char top = st.top();  
            st.pop();  
        }  
    }  
    if (st.empty()) return "Yes";  
    else return "No";  
}
```



```
if ( symbols[top] + symbols[bracket] == 0 ) {  
    between "No";  
}
```

```
}  
if ( st.empty() ) return "Yes";  
between "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.

NGE

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 5 | 2 | 2 | 5 | 7 | 8 |
|---|---|---|---|---|---|---|

NGE

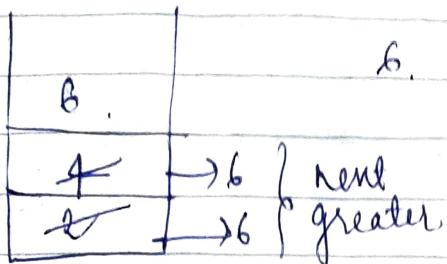
| | | | | | |
|---|----|----|----|---|----|
| 5 | 25 | 25 | -1 | 8 | -1 |
|---|----|----|----|---|----|



Using stack we can do in $O(N)$

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

general example



| | | | | | | | |
|----|------------------|-----|---|----|----|----|-------|
| 8 | \rightarrow -1 | | | | | | |
| 7 | \rightarrow 8 | 4 | 5 | 2 | 25 | 7 | 8. |
| 25 | \rightarrow -1 | NGE | 5 | 25 | 25 | -1 | 8. -1 |
| 2 | \rightarrow 25 | | | | | | |
| 5 | \rightarrow 25 | | | | | | |
| 4 | \rightarrow 5 | | | | | | |

LIFO

we will push 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 (int sort)

↓
Quick + heap + insertion

```

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] << " ";  
}
```

In case of vector

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

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

Complexity : $O(n \log n)$

for inbuilt
sorting algo

Comparator function

Swap karna hai to \rightarrow false

swap ni karna to \rightarrow true

Taking example of a pair in which we need to sort the first in ascending order. If ~~is~~ 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;  
    }
```

int main() {

int n;

cin >> n;

vector<pair<int,int>> a(n);

pair change
wise
like
do

```
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 << endl;
}
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 kamesha

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

i/p \rightarrow 6 4 5 5 2 5 7 8

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

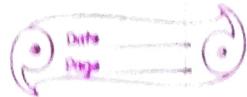
for vector:

```
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(), s); X

auto it = s.lower_bound(s);



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)
(return 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 f1 = find(v.begin(), v.end(), 2);

- Q. 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;
```

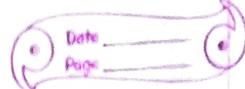
```
for string
    string s = "bcdefgh";
    reverse(s.begin(), s.end());
    cout << s << endl;
```

(Lambda 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 , lambda function)



lambda funct → small function

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

auto sum =

```
cout [ ] (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