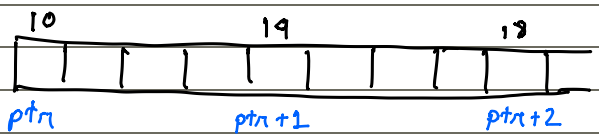
#⭐, #C++STL

Cpp STL :-

Pointers :-

#**&&**

```
int* ptr;
int num = 10;
ptr = &num;
*ptr = 100;
```
(now num is 100)



Because int* increments in 4.

✓✓ int arr[100] ⇒ arr is the pointer for the first element of the array.

✓✓
```
for (int a : arr)
{ cout << &a << endl; }
```

o/p:-
```
0x1010
0x1010
0x1010
 ⋮
```
a is in a fixed position and values of arr elements getting copied to a.

For evaluation expression Question Leetcode;

#LeetCode/Medium

#⭐

```
unordered_map<string, function<int(int,int)>>
mp = {
    {"+", [](int a, int b){return a+b;}},
    {"-", [](int a, int b){return a-b;}},
    {"*", [](int a, int b){return a*b;}},
    {"/", [](int a, int b){return a/b;}}
};
```

void function (vector<> & v) → Original vector is passed. original v is modified.

void function (vector<> v) → takes O(N) time to make a copy. original v unmodified.

vector<int> v[3] → v[0] [vector<int>
v[1] [vector<int>
v[2] [vector<int>

v = [[1,2,3], [4], [5,6]]

v[i].push_back

Iterator :-

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

for (int val : arr) → copy of val

for (int & val : arr) → by reference → any change in val persists afterwards.

auto x = 1; auto y = 1.0;

map<int, string> mp;
mp[1] = "abc";
mp[5] = "aab";
mp[2] = "gh";

* map stores in sorted order

map	
1	'abc'
2	'gh'
5	'aab'

```
map<int, string> :: iterator it;
for (it = mp.begin(); it != mp.end(); ++it)
    { cout << (*it).first << " " << (*it).second << endl; }
```

* map takes O(log n) to insert, delete, find Key val pairs

✓✓ map → implemented on Red Black Tree O(log n)

unordered_map → implemented on Hash Table O(1)

map<pair<int,int>, string>

valid

not valid

unordered_map<pair<int,int>, string>

hashing pair<int,int> is not implemented in STL.

map ≅ Set unordered_map ≅ unordered_Set

✓ multimap , multiset

↳ allows non unique entries

works on Red Black Trees.

→ Multiset is same thing as Priority Queue.

eg.

```
map<pair<int, string>, vector<string>> mp;
for (auto &ptr : mp)
{ ptr.first = pair   ptr.second = vector<> }
```

stack <int> st; queue <string> q; deque <int> dq;

st.push()	q.push()	dq.push_back()	dq.push_front()
st.pop()	q.pop()	dq.pop_back()	dq.pop_front()
st.top()	q.front()	dq.back()	dq.front()

- empty()
  returns bool value.

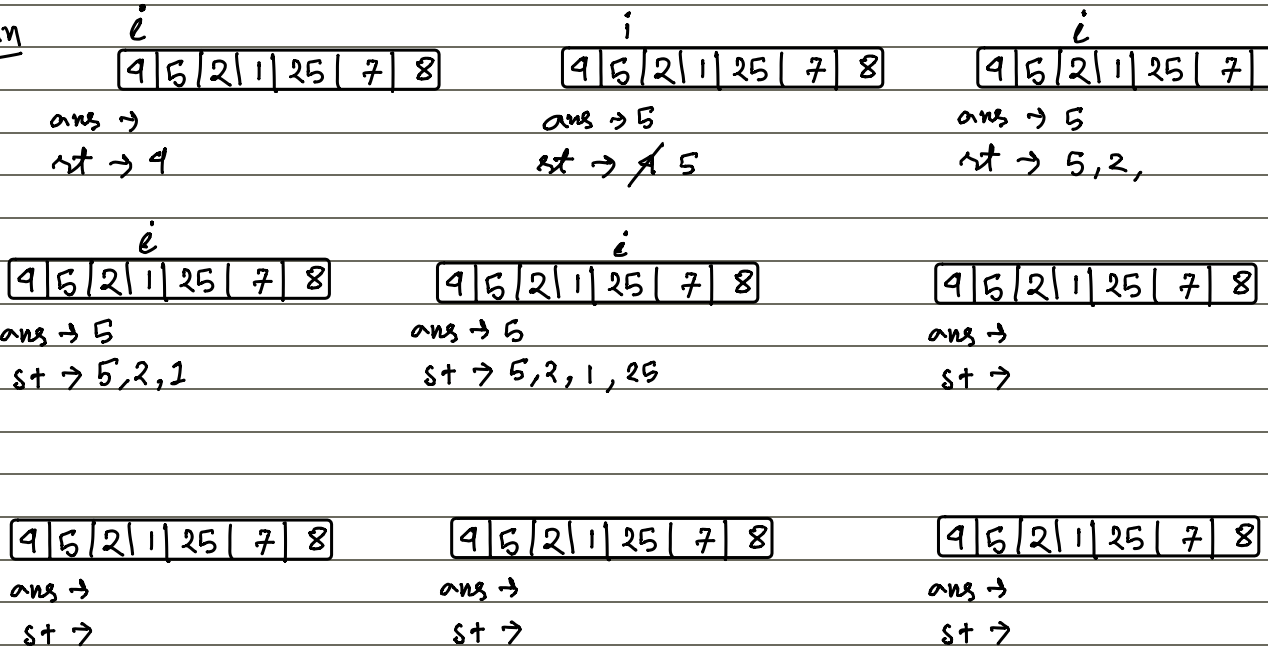
#algorithm/next greater element

→ Next Greater element Using Stack :-

input ⇒ [4, 5, 2, 1, 25, 7, 8]
o/p ⇒ [5, 25, 25, 25, -1, 8, -1]

Data Structure : vector<int> ans.
stack<int> st;

Dry Run



Built In Sorting C++

sort(arr.begin(), arr.end(), cmp)
↳ it's a function.

```
bool cmp(int a, int b){
    return a > b;
}
```

This cmp Reverse sort an array.

assume comparison between 2 elements a & b

```
bool cmp(int a, int b){
    return a > b;
}
```

if cmp returns true a stays before b else a goes after b.

Upper Bound Lower Bound :-

int arr[n] if arr = 1, 2, 3, 7, 9, 10, 12 ... → Sorted.

```
int (*ptr) = upper_bound(arr, arr+n, 5);
cout << (*ptr) ↳ 7
```

int (*ptr) = upper_bound(arr.begin()+2, arr.end()-3, 5);

Range works too ...

Complexity = O(log(n))

For set<int> S, map<int, int> mp. → only key of mp

auto it = upper_bound(S.begin(), S.end(), number)

this is O(N)

But;

auto it = S.upper_bound(number) ... This is O(log N)

Switch Statement :-

```
for (auto & edge : edges) {
    type = edge[0]
    node1 = edge[1]
    node2 = edge[2]

    switch (type) {

        case 1: {

        }

        case 2: {

        }

        case 3: {

        }

    }
```

1 red, 2 green, 3 blue type