- Outside main(), size of the int/double/char array is 10^7 and the boolean array is 10^8.
- Inside main(), the size of the int/double/char array is 10^6 and the boolean array is 10^7.

- array<int, 3> arr; // Outside int main() -> {0, 0, 0} but inside int main(),garbage value.
- array <int,5> arr={0} //->{0,0,0,0,0}
- array <int,5> arr={1}//->{1,0,0,0,0}
- array <int,5> arr={2,1,4}//->{2,1,4,0,0}
- int arr[10000] = {0}; // all set to zero
- int arr[10000] = {5}; // first set to 5 and rest all set to zero
- array<int, 5> arr; arr.fill(10); -> /// {10, 10, 10, 10, 10}

begin() points to first element , end() points to next of last element, rbegin() points to last element, rend() points to previous of first element

- for(auto it= arr.begin(); it!=arr.end();it++) {cout << *it << " ";}
- for(auto it:arr) {cout << it << " ";}
- string s = "xhegcwe"; for(auto c:s) {cout << c << " ";}

- vector<int> vec2(4, 10); // -> {10,10,10,10}
- vec.emplace_back(1); // emplace_back takes lesser time than push back
- vector<int> vec3(vec2); // copy the entire vec2 into vec3
- vector<int> vec1(vec.begin()+1, vec.begin() + 3); // -> [1, 3) -> 1st index to 2nd index

Sets

Inserts unique elements in ascending order.
Sets etc. can’t be directly accessed like vectors.Only iterator can be traversed

- set<int> st; st.insert(x);
- st.emplace(6); //faster than insert()
- st.erase(startIterator, endIterator)
- st.erase(5) // deletes element with value 5 if present
- auto it = st.find(7); //if 7 present returns iterator pointing to 7 else points to end of set

All operations in set,map,multimap, and multiset are of O(logn) time complexity.

unordered_set

Inserts unique elements in any unpredictable order.
Average time complexity of unordered_set and unordered_map functions(e.g. find) are O(1) (practically always applicable) but the worst case is linear in nature, O(n)
→ Don't make a unordered_set of vector.

multiset

Inserts elements in ascending order and can store duplicate elements.

- ms.erase(2); //all elements with value 2 will be erased
- auto it = ms.find(2); // returns an iterator pointing to the first element of 2
- ms.count(2); // finds how many times 2 occurs
- ms.erase(ms.find(2)); // only first element with value 2 will be deleted

map

- stores keys and values in increasing order of key and keys are unique.
- functions like `mpp["raj"] = 27; ,mpp.emplace("raj", 45); mpp.erase(iterator);,mpp.erase(key) ,auto it = mpp.find("raj");mpp.empty(),mpp.count("raj");` are similiar to other containers.
- Iterate over a map using itr i.e. **itr->first='key'** & **itr->second='value'.**
- Implemented through balanced tree or red-black tree.

Read map internal implementation later

unordered_map

Inserts unique elements in any unpredictable order.Implemented through hash table.

multimap

Inserts elements in ascending order and can store duplicate elements.

Operations (insertions and search) in multimap and map take O(logn) time and in unordered_map ,it takes → O(1) average time.(practically always true) and theoretically O(n) time.So unordered_map always preferable.
Unordered hashmap takes N in the worst case due to collisions, we can say it as n or n log n in an interview and then explain !

stack

pop // top // size // empty // push and emplace

queue

push,pop,front,size,empty

priority_queue

Inserts in descending order
push,pop,top,size,empty

For storing in ascending order: priority_queue<int, vector<int>, greater<int>> pq;
Min heap sorted by first element:
priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;

deque

push_front() // push_back() // pop_front() // pop_back() // begin, end, rbegin, rend // size // clear // empty // at // remove

list

push_front() // push_back() // pop_front() // pop_back() // begin, end, rbegin, rend // size // clear // empty // at // remove

bitset

Boolean array where each element size=1 bit and size of array is static

- bitset<8> bset1; // default constructor initializes with all bits 0 i.e. 00000000
- bitset<8> bset2(20); // bset2 is initialized with bits of 20 i.e. 00010100
- bitset<8> bset3(string("1100")); // bset3 is initialized with bits of specified binary string i.e. 00001100
- bset1[1]=1; // 00000010
- test(),all(),none(),count(),flip(),set(),reset(),size(),count()
- Maximum size 10^8

Binary search in-built functions

→ Work only in ascend. sorted array/vector
→ **binary_search(arr+a,arr+b,x)** returns true if x is present in vector 'vec' in range [a,b).
→ **lower_bound(arr,arr+n,x)-arr** returns position of the first element in the range [0,n) which has a value >=x,returns n if 'x' not found.
→ **upper_bound(arr,arr+n,x)-arr** returns position of the first element in the range [0,n) which has a value>x,returns n if 'x' not found.
→ In vectors use lower_bound(vec.begin(),vec.end(),x)-vec.begin() instead.
→ If you want to get iterator instead of position,don't use **-vec.begin()** in last

Algorithmic in-built functions

- max_element(firstIterator, lastIterator); returns iterator of maximum value in range.
- accumulate(startIterator, endIterator, initialSum); returns sum in the range + initialSum
- count(firstIterator, lastIterator, x) ; returns how many times the element 1 occurs in the range

- auto it = find(arr, arr+n, 2); // return an iterator pointing to the first instance of it, or else it returns pointing to the end() if it is not there
- **next_permutation**(firstIterator, lastIterator) returns next permutation of a string/array:O(n).To print all permutations of a string etc.,use a do while loop with next_permutation function:O(n!)
- **Comparators**: returns true if no need to swap and false if need to swap.Better to use < or > operator . Caution : don't use >= or <=.Write comp() outside main().
- greater<int> OR greater<pair<int,int>> are inbuilt comparator to sort in ascending order.

C++ STL

Iterate over map using itr i.e. **itr->first='key**' & **itr->second='value'.**

→ **Sort vector of pair by second value**

```
bool sortbysec(const pair<int,int> &a,
               const pair<int,int> &b)
{
    return (a.second < b.second);
}

sort(vect.begin(), vect.end(), sortbysec);
```

Priority queue of structures

Struct st{
Int a,b,c;
};

```
Struct comp
{
Bool operator ()(st &s1,st&s2)
Return (s1.b>s2.b); //return true if need to swap
};
Priority_queue <st,vector <st>,comp> pq;
```

Strings Functions

Copying substring from string S1 to S2;
string S2=S1.subtr(ind,len); //ind=starting index of S1,len=no. Of characters.

Comparing substring of S1 to S2
S1.compare(ind,len,S2) //ind=starting index of S1,len=no. Of characters.
returns 0 if substring of S1 equal to S2;
return negative no. if length of substring of S1 is shorter than S2 or first character that doesn't match is smaller.
A value > 0 : if *this is longer than S2 or the first character that doesn't match is greater.

cin.getline(key, 25); //taking input in predefined string

```
char a[]={'a','b','c','d','e'};
string s1=string(a);
cout<<s1;
```

strcmp(s1,s2) //returns diff b/w ASCII of first unmatched character of s1 and s2.
s1 & s2 can be char array also.

strcat(key, buffer); //adding all values of buffer to key

strcpy(key, buffer); //copying all values of buffer to key

strlen(s1) //Length of String

str.erase() //erases all characters of string
str.erase(i) //erases all characters with idx>=1
str.erase(i,len) //erases 'len' no. of characters starting with i.
str.erase(str.begin() + 4); // Deletes character at position 4

str.erase(str.begin() + 0, str.end() - 6); // Deletes all characters between 0th index and str.end() - 6 inclusive

stoi and to_string

string str=to_string(num); //converts number (of int/long long etc) to string
int num=stoi(str); // converts string to num;