**C++ Libs:**

#include<bits/stdc++.h> includes every standard library

**// sort asc**

    sort(v.begin(), v.end());

**// sort desc**  
    sort(v.begin(), v.end(), greater<int>());

**// sort based on column**

static bool sortcol( const vector<int>& v1, const vector<int>& v2 ) {

return v1[1] > v2[1];

}

    sort(v.begin(), v.end(), sortcol);

**// count occurrences of element eg. 3**  
    count(vect.begin(), vect.end(), 3)

**// searches in[start,last)**

**// returns last if elem not found**  
 find (vec.begin(), vec.end(), 3);

**// find sum of all elements**

accumulate(a.begin(), a.end(), 0);

**// find max**

\*max\_element(a.begin(), a.end());

\*min\_element(a.begin(), a.end());

**// character c will be printed n times**

cout << string(n, c) << endl;

**// swap two values**

swap(a,b)

**// for iterating over any structure (eg. Unordered map)**

for(auto i:um){

cout<<i.first<<" ";

}

**Vector:**

#include <vector>

**// erase elem from vector based on index**

vec.erase(vec.begin()+i);

**// erase all elem from vector**

vec.clear();

**//Initialize vector**

vector<int> vec(n,0);

**//Initialize 2d vector**

vector<int> v(r,0);

vector<vector<int>> visited(c,v);

**//Append vectors**

v1.insert(v1.end(), v2.begin(), v2.end());

**Hashmap:**

**//Declare hashmap**

unordered\_map<int, bool> um;

**//Find in hashmap**

um.find(23) == um.end()

**//Erasing element \*\*\*IMP**

um.erase(34) //instead of um[34]—where ele can still be found using.find, use erase to remove completely.

**Array:**

**//Initialize array**

int arr[n] = {0};

#include <algorithm>

**// searches in[start,last)**

**// returns last if elem not found**  
 find (arr, arr+n, 3);

**String:**

**//Append to string**

string s+= ‘-’;

**//Substrings**

s=”Hello World”

s=s.substr(1,3) //ell (pos,len)

s=s.substr(6) //World (new beginning)

**Pair:**

**//Get elements from pair**

cout<<p.first<<” “<<p.second;

**Set:**

**//Initialize set using array**

set<int> s(arr, arr+n);

**//Iterate over set till kth element**

set<int>::iterator itr = s.begin();

advance(itr, k-1);

**Priority Queue:**

**//Used when min needs to be found in O(1) time, insertion takes O(logn) lesser than comparisons of the O(n) approach;**

**//Maintains order**

**//Initialize priority queue (max heap by default)**

priority\_queue<int> q;

**//Initialize priority queue (min heap, pi can be replaced by type, even pair)**

priority\_queue<pi, vector<pi>, greater<pi> > q;

**//Operations**

q.pop();

q.top();

q.push(1);

**LinkedList:**

**//Psuedo head**

ListNode psuedo\_head(0);

ListNode\* curr=&psuedo\_head;

//Keep assigning nodes to curr->next;

Return pseudo\_head.next;

**BitSet:**

**//Declare hashmap**

unordered\_map<int, bool> um;