***C++ notes***

**Episode 3.**

**OVERFLOW**

Int, char, long int, long long int, float, double

Int -10^9 to 10^9.

Long int 10^12.

Long long int 10^18.

Int a=10000;

Int b=10000;

Int c=a\*b; output-10^10 expected but real output 1410065408 this is overflow. Range of c is limited to the range of int.

PRECISION ERRORS.

Setprecision(n) function can be used to fix the precision of a double data type.

We can store large numbers in double but accuracy and precision falls. The stored number will be very close to the actual number but not the number.

**Episode 4**

**CONDITIONAL STATEMENTS**

SCOPE…

Some variables are global and local..

LOOP STATEMENTS AND JUMP STATEMENTS..

**Episode 5**

**STRINGS – double quotes, character – single quotes. Strings concatenate with sign ‘+’.**

Strings in c++ work like an array of characters with indexing from 0.

. size () function can be used to calculate the size of string.

To compare strings, we can use == or. equals () function.

Getline function-it takes one whole line as input including empty spaces.

Getline(cin,a);

LARGE NUMBERS CAN BE TAKEN INPUT USING STRING.

**Episode 6**

**Arrays its size limits both 1d and 2d arrays**

Segmentation fault- this arises when u try to access an element of array whose index is greater than or not in the range of array index

Like trying to access n+3 in an array of size n.

We can also store a string using character array.(AVOID)

In order to write numbers in exponential form, we can use e method

Like 1e5,1e7 etc.

If we declare a variable as **const**, its value doesn’t change.

**Episode 7**

**FUNCTION, C++ REFERENCES…**

Main() is a function, a default function…

Defining a function-

Void has no return type..

Void printHello(){

}

When we return a value for a function, the below code inside the function will not be executed. If the function has a void data type and doesn’t return any value and we are using the function inside main, we wont observe any changes in the output as the function creates a copy of original variable. Hence it is necessary to pass the variables as references in that case.(&)

Int n=a – pass by value in a function

Int &n =a – pass by reference

Swap() – inbuilt function inside c++;

Max(), min()

ARRAYS ARE ALWAYS PASSED BY REFERENCES.

**Episode 8**

**POINTERS** – it stores the memory address of a variable denoted by \*

Ex -\*p (think of star as value at !! value at p.)

Memory allocations.

8 bits = 1byte

&x represents the address of x.

Array and pointers

Int a[10]; - Allocation of array values is continuous

Int y;

Int x =4;

Int \*p =&x;

Cout<<&x<<endl; --- 0x7ffee…..(hexadecimal number)

Cout<<p<<endl; ---- 0x7ffee… (same value stored)

Cout<<\*p<<endl; - 4 (value stored in address of p or value pointed by p)

If we change the value pointed by pointer, then the value of x will automatically change to that value

If we declare an array a[10] then a is a pointer pointing to the address a[0].

**Episode 9**

**ERRORS**

TLE – time limit exceeded.

MLE – memory limit exceeded.

RE – runtime error.

Compilation time and execution times are calculated separately.

If u r running a loop that goes till N and N is of order 10^9 or greater than that then execution time will shoot 1 seconds for sure.

RECURSION – calling a function within a function.

**Episode 10**

**TIME COMPLEXITY ANALYSIS AND CONSTRAINTS.**

Code complexity – 0(3) 3 denotes that no. of iterations/statements are there.

0(5)+0(N)

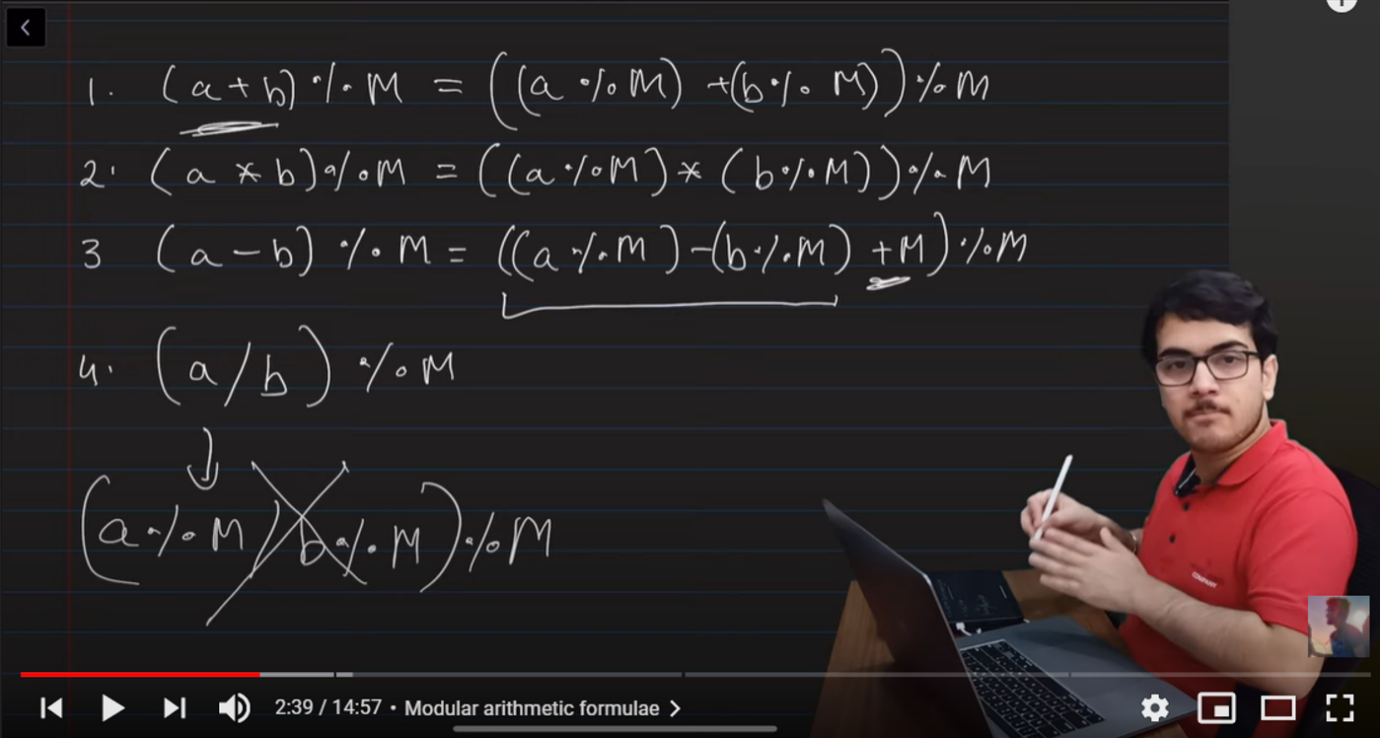
For online coding platforms in 1 second, the number of iterations limit is 10^7 – 8.

O(n) + O(n) + O(n) = O(n) there even if we run 4 or 5 for loops till n the execution time won’t vary a lot.

Whereas if we do a nested for loop 2 times the complexity becomes n^2 and TLE error will generate.

**Episode 11**

**Print answer modulo 10^9+7 and Modular Arithmetic**



Modulo is used when we have a huge calculation which we cannot store even in long long. For example 21 !- it is a very big number. Therefore we output its modulo with M where M can be any integer say 47. Now 21 ! % 47 will always be less than 47 and hence we can store the result.

**Episode 12**

**Pre-Computation techniques and hashing and code optimization**

Consider a code to print factorial of N for T test cases.

And both T & N have constraints till 10^5

Therefore time complexity will be 10^10 for entire execution and will overshoot 1 second as in 1 second we can only perform 10^7 iterations.

Solution –

We will declare an array and store all the factorials till N, a bit greater than N say 10^5+10.

Const int N = 1e5+10;

Long long fact[N];

We will store all factorials outside N, so that it doesn’t increase the time complexity.

Therefore,

Int main(){

Fact[0]=fact[1]=1;

For(int i=2;i<N;i++){

Fact[i]=fact[i-1]\*i;

}

}

After this we can run the loop of T and output the factorial of N entered by the user.

**HASHING…**

We make a hash array

Then we will store the count of numbers of array given in program to print count of a number in the whole array.

Program

Const int N = 1e7+10;

Int hsh[N];

#global arrays are initialized as 0

Then we will

For(int i=0;i<n;i++){

Cin>>a[i];

Hsh[a[i]]++;

}

**Episode 13**

**PREFIX SUM IN 1D AND 2D ARRAYS**

Consider a program to print the array elements from L to R where L and R are input by the user

We pre compute another array which stores the sum till ith element

When we want sum from L to R we can subtract the two sums.

Using this technique the time complexity gets reduced.

This method is named as PREFIX SUM.

Program-

Int pf[N];

For(int i=1;i<=N;i++){

Cin>>a[i];

Pf[i]=pf[i-1]+i;

}

Now we will try prefix sum in 2D arrays;

Formula

**Pf[i][j]=a[i][j] + pf[i-1][j] + pf[i][j-1] -pf[i-1][j-1];**

**Episode 14**

**Practicing different methods based questions on prefix computation**

The question says to print the GCD of remaining terms from an array after excluding terms from L to R input by the user.

INBUILT FUNCTION TO CALCULATE GCD ------- \_\_gcd(a,b)

The first approach is to compute the gcd of the two sections that is before L and after R and find the gcd of the two numbers we got.

But using this approach, we will get a TLE error for large test cases and value of N.

The SECOND APPROACH..

We will make two arrays a forward and a backward array. Then we will store the gcds from 1 to i and from i to n in reverse order.

#Precomputation

**Episode 15**

**PRE COMPUTATION TECHNIQUES ( medium level )**

**Given-**

We are given a 1D array filled with zeroes and a number of queries.

Like

A b k

1 5 2

This means we r supposed to fill the indexes from 1 to 5 with 2 or add 2 in these indexes of our 1D array.

The Given constraints are very large so we need an optimized program for this.

We can do this by prefix sum and hashing in order to avoid TLE error.

If we want to add 5 from index 2 to 4, we can add +5 to 2nd index and -5 to 4+1th index and calculate its prefix sum. It would give us the required array.

**Episode 16**

**PRE COMPUTATION TECHNIQUES ( Hard level )**

The question says we have a string a multiple queries that have two integers from L to R. We are required to check if the letters from L to R can be rearranged to form a palindrome.

LOGIC – The logic is we will count the count of each kind of letter from L to R, if the number of counts that are odd is more than 1, we cannot form a palindrome. If it is one we can put it in the center.

We will use a hash array denoted by hsh[] for this question. Since string has indexes from 0, we will have a 0 based array mapped with a and 25 mapped to z. Therefore we will get the count of all the letters in our hash array using

Hsh[s[i]-‘a’]++; inside the for loop.

Then we will check the array elements with odd count.

OPTIMISATION LOGIC- We will create 26 arrays of n size, one array for each letter which stores the count at every position of string.

Then we will calculate its PF(Prefix Sum), and to find the count of that letter from L to R, we can simply subtract the counts of that positions in our prefix sum array.

So basically, we created a 2D hash array of N x 26.

**Episode 17**

**RECURSION**

A function which calls itself is known as a Recursive function.

Ex.

Void func(){

Func();

}

Int main(){

Func(); }

If we run this code, we will get a SEGMENTATION FAULT due to STACK OVERFLOW. (Infinite calling of function)

Stack overflow is caused because each time the stack stores the address of function being called. The memory of stack gets filled due to infinite calling and hence it overflows.

How to write a recursive code !!

Take an example- program to print factorial of N

Fact(N)=Fact(N-1)\*N

CODE

Int fact(int n){

If(n==0) return 1;

Return Fact(n-1)\*n;

}

Int main(){

Int n;

Cin>>n;

Cout<<fact(n);

}

**Episode 18**

**DIGIT SUM AND ARRAY SUM USING RECURSION**

**// Sum of Array**

We define a function sum which takes two parameters n and a, and returns the sum of array elements of a till nth index.

**Recursive code-**

Sum(n,a)=a[n]+sum(n-1,a);

**The function-**

Int sum(int n,int a[]){

If(n<0) return 0; - (BASE CONDTION)

Return sum(n-1,a)+a[n];

}

**// Sum of Digits**

Int digitsum(int n){

If(n==0) return 0;

Return digitsum(n/10)+n%10;

}

**Episode 19**

**MERGE SORT AND SELECTION SORT (Sorting)**

**Concepts**

**Selection Sort**

Int main(){

For(int i=0;i<n;i++){

Int (minimum index)m = I;

For(int j=i+1;j<n;j++){

If(a[j]<a[m])

M=j;

}

Swap(a[i],a[m]);

}

}

**Merge Sort**

We will use this algo using recursion

Const int N=1e5+10;

Int a[N];

Void merge(int l, int r, int mid){

Blah blah blah….

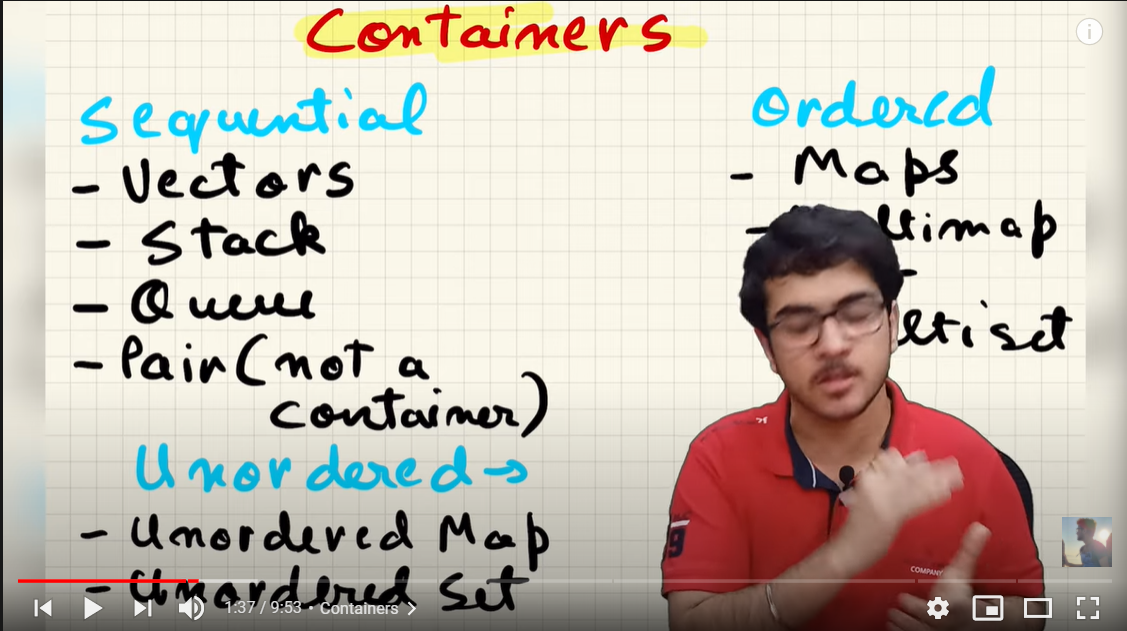
}

**Episode 20**

**C++ STL**

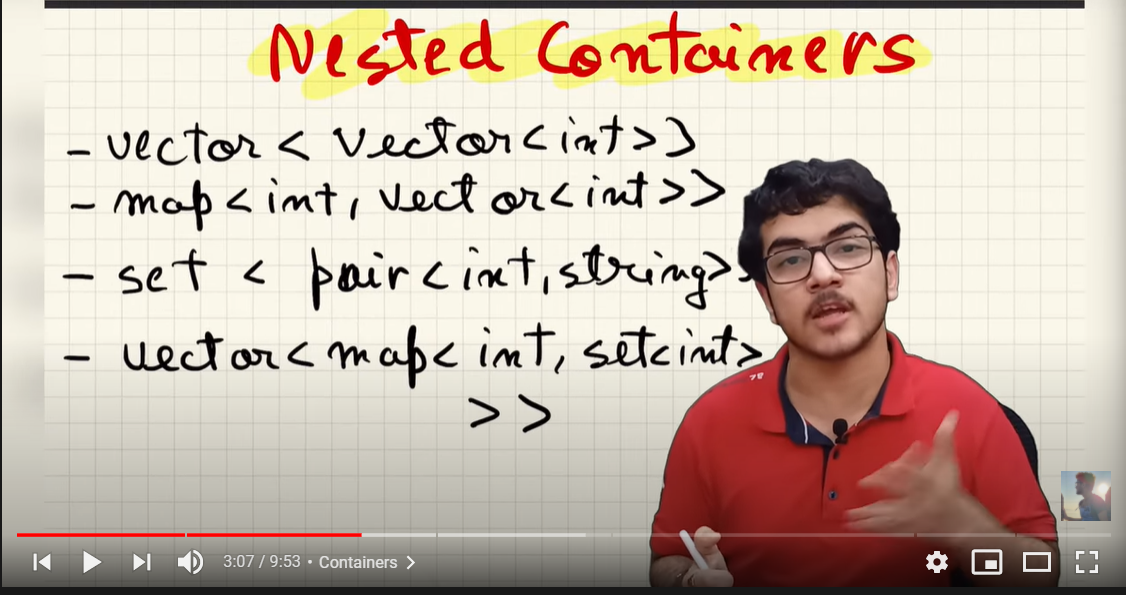
**Standard Template Library**

**Containers –** These are the data structures pre implemented in data structures of C++;

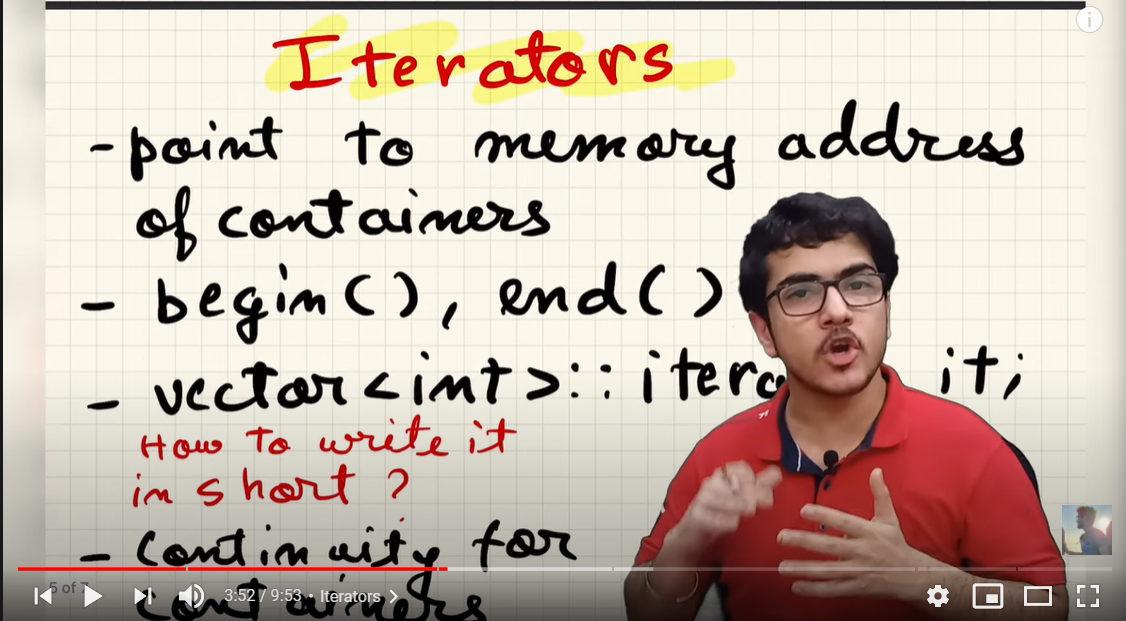


Ordered containers – Map, Multimap, Multiset.

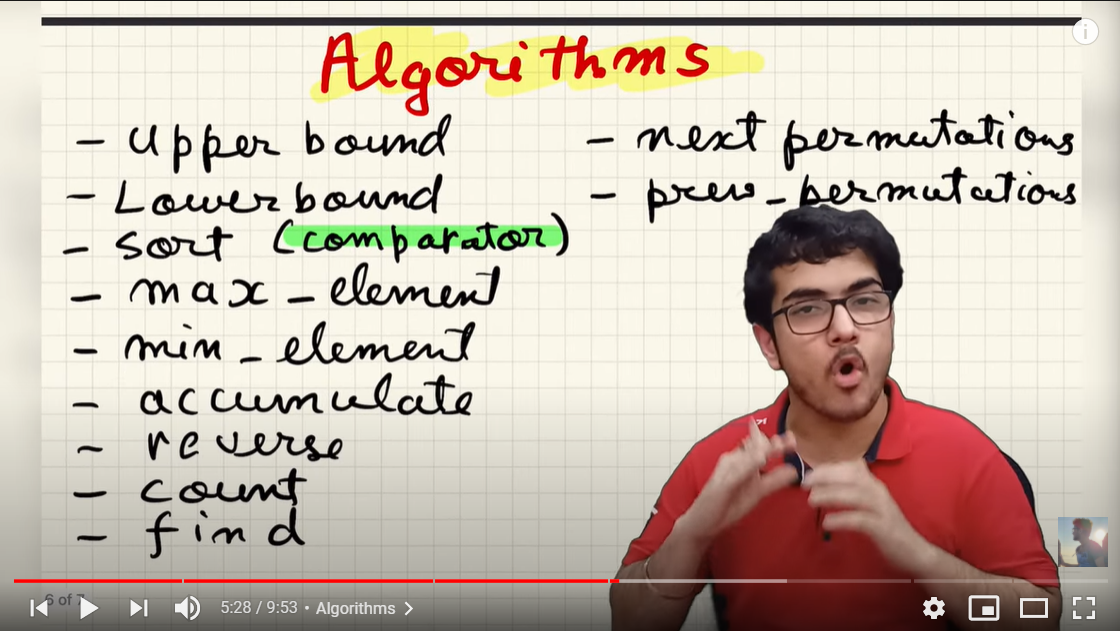
**NESTED CONTAINER- CONTAINERS WITHIN CONTAINERS**

****

**Iterators**

****

**Algorithms**

****

**Functors** – Classes which can act like functions.

**Episode 21**

**PAIRS AND VECTORS**

Pair is a class in STL which stores 2 values.

Syntax-

Pair<1,2> 1,2 – data types.

Ex- **pair**<int,String>p;

P will become the pair of two data types.

P=make\_pair(2,”abc”);

**make\_pair()** – it is an inbuilt function inside c++ stl.

Cout<<p.first – 2.

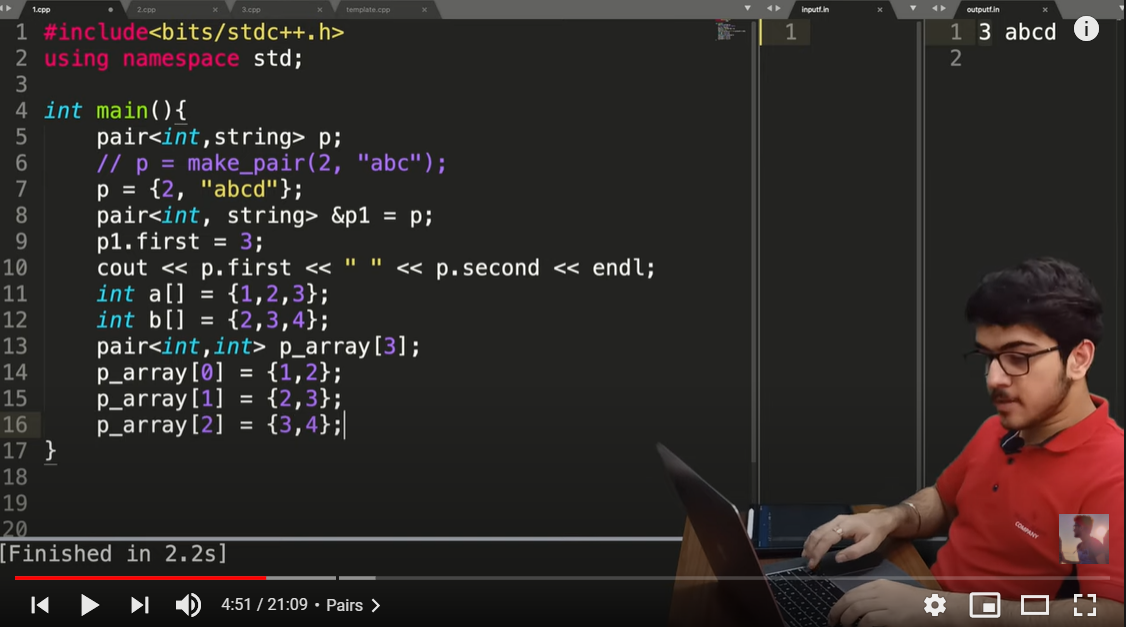
Cout<<p.second - abc.

M2

***P={2,”abc”};***

***To copy pairs,***

***Pair<int,string> p = p1;***



Pair is used to maintain a relation between two things that are generally mapped.

***Vectors.***

Vectors are continuous memory blocks like arrays of dynamic size.

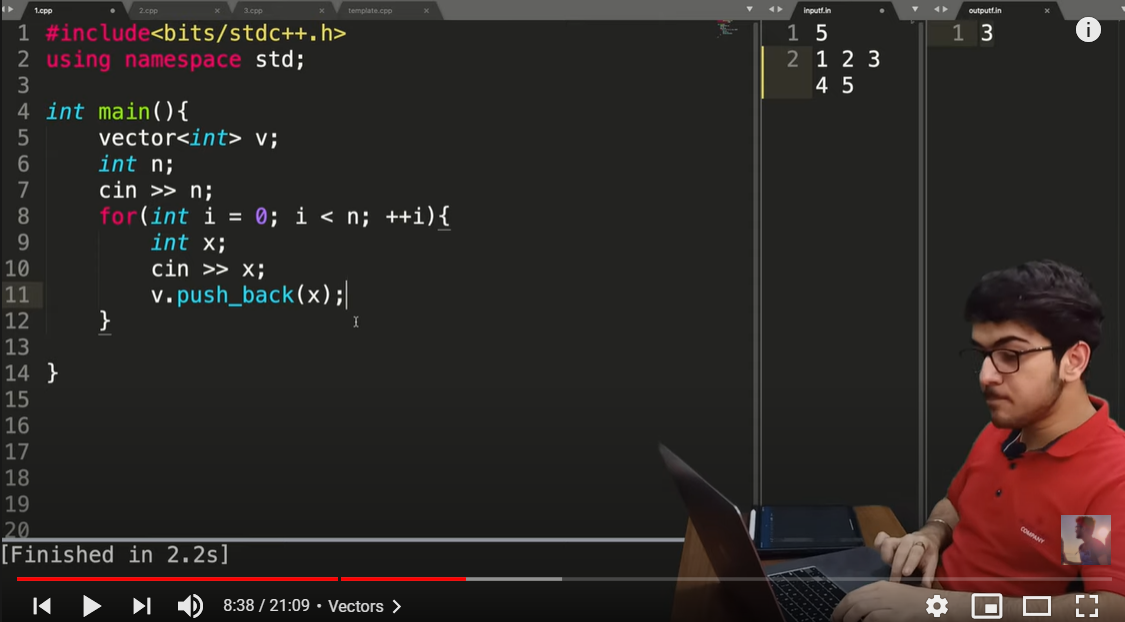
Syntax –

Vector<(data type)> name of variable.

Ex.

Vector<int> v;

***How to add values inside vector ??***



Pushback() function is used to store the value provided at the end of vector say v.



Vector<int> v(10)- a vector filled with 0 of initial size 10;

Vector<int>v(10,3)- a vector filled with 3 of initial size 10;

***v.pop\_back() –*** it removes the last element from the vector.

We can easily copy a vector in to another vector v2 unlike array.

**Vector<int> v2=v; (o(n)) complexity…..**

**# If we pass a variable as a parameter in a function, it creates a copy of that variable. Therefore any change in that copy is not reflected in the actual variable.**

**# But if we pass the variable as a reference parameter in function using &, then any change in the variable inside the function is reflected in the actual variable.**

Vector<string> v;

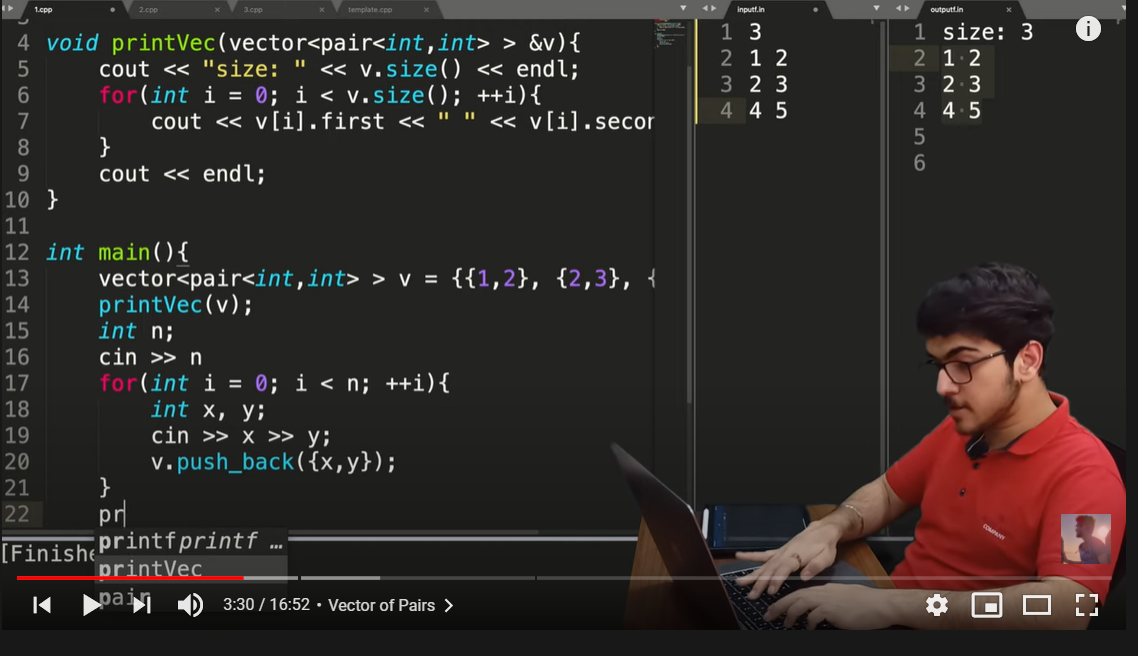
v.pushback(x);

**Episode 22**

**NESTING IN VECTORS**

Vector<pair<int,int>>v;

V={{1,2},{3,4}}; each element of vector is a pair.



**ARRAY OF VECTORS**

Vector<int>v[10]; an array of 10 vectors.

**VECTOR OF VECTORS**

Vector<vector<int>>v;

In case of vector of pairs, we can access individual elements using .first() and .second() for each pair.

**Episode 23**

**ITERATORS-They are pointers like structures**

They help in performing different operations on containers of STL.

Iterators point towards the elements of container

First iterator - .begin(); **points towards the first element of a vector**

Last iterator-.end(); **points towards the next position of vector**

Declaring an iterator

Vector<int> :: iterator it;

If

Vector<int> :: iterator it = v.begin();

Cout<<(\*it)<<endl; returns the first element of vector v;

If (\*(it+1))- returns the second element of vector.

Looping with iterator

For( it =v.begin();it!=v.end();it++){

Cout<<(\*it)<<endl;

}

It++ - **moves to next iterator**

It+1 – **moves to next location**

**No diffn for vectors as they are continuous but can cause runtime and compilation error in maps and sets which are discontinuous. Therefore use it++;**

**Episode 24**

**AUTO AND RANGE BASED LOOPS**

**Range base loops – works for all containers**

Ex- for(int value : v){

Cout<<value<<” “;

}

Cout<<endl;

Here, the value variable stores the values of vector v as a copy and not its reference. Therefore any change in the variable wont cause difference in the output.

**AUTO KEYWORD**

Auto is used to declare a variable or iterator. It is useful as it dynamically determines the data type of the variable .

Ex- auto a =1;

So a will be assumed as an integer and if a=0.1 then a will be assumed as a double or float data type.

**Episode 25**

**C++ STL Maps**

**Maps** – normal maps and unordered maps apart from this there are multi maps.

Map is a data structure which stores key value pairs.

Example – key is integer and value is string. Then standard map helps in mapping them which are easily accessible.

**RED BLACK TREE** – a self-balancing tree which implements normal maps

Each element of a map is a pair of key and value.

They are not of continuous nature . random locations of elements with links.

**DECLARING A MAP**

**Map<int,string> m;**

m[1] = “abc”;

m[2]=”cdc”;

m[5]=”acd”;

Printing maps is just like that of vectors.

Map<int,string>:: iterator it;

For(it= m.begin(); it!=m.end() ; it++){

Cout<<(\*it).first<<” “<<(\*it).second();

}

***m.size() gives the size of the map.***

Keys of value are unique

We cannot insert duplicate keys.

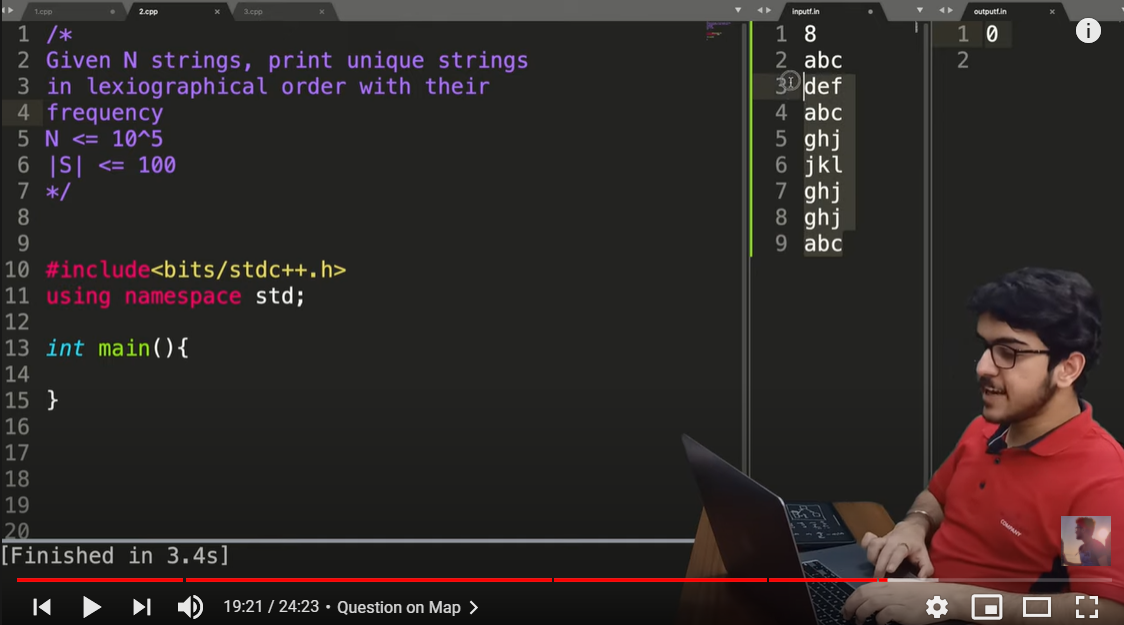
M.find() – this function finds a value corresponding to a key

Similarly M.erase(3); will erase the 3rd numbered element from M map.

These function are common functions for all containers

.clear() will clear all the map elements

QUESTION ON MAP

Q.

SOLN –

We declare a map

Map<string,int> m;

Int n;

Cin>>n;

For(int i=0;i<n;i++){

String s;

Cin>>s;

m[s]=m[s]+1;

}

For(auto pr : m){

Cout<<pr.first<<” “<<pr.second<<endl;

}

}

**# The lexicographical part is solved by maps itself as it gives output in sorted format.**

**Episode 26**

**C++ STL MAPS PART 2**

Unordered maps

Unordered\_map<int,string> m;

**The keys will be printed in random order rest is same.**

**They use inbuilt hash tables instead of trees.**

It should be used when order of our data need not be sorted. Therefore it takes less time than ordered maps.

Question.. same as maps

**Episode 27**

**SETS,UNORDERED SETS AND MULTISETS**

Sets are similar to maps but they only store keys and not values.

Syntax:

Set<string>s;

It will also store the values in sorted order.

To input values, we use **.insert()** function

s.insert(“abec);

s.insert(“bbcd”);

s.insert(“ccd”);

To access a value,

Auto it = s.find(“abc”);

Unordered\_set is used when we don’t need to sort the set. It takes less storage also. **S.find()** finds a value from set and if its equal to **s.end()** then the value is absent from the set s.

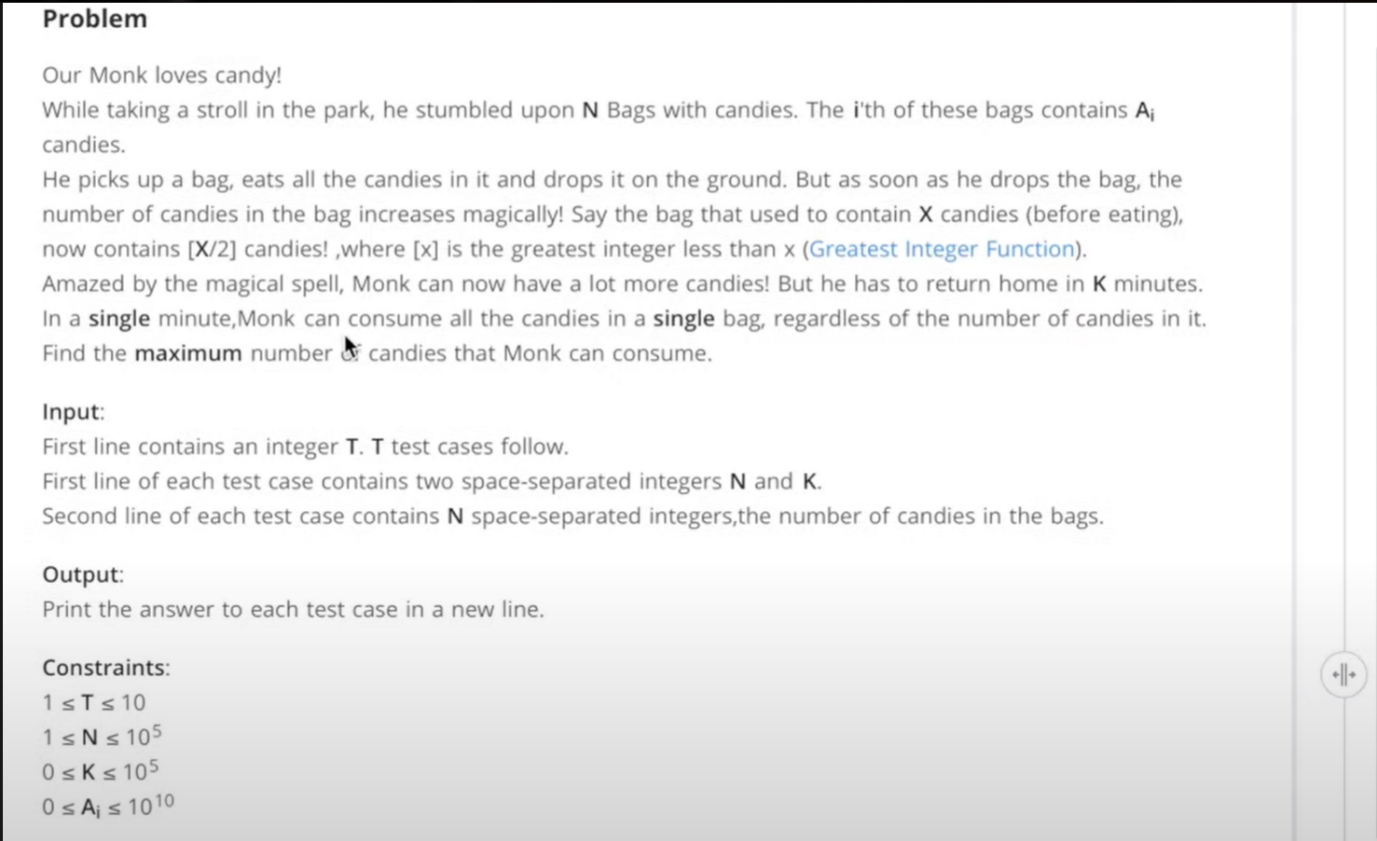
**MULTISETS**

**In multisets, we can have duplicate values and it we erase ex lets say an abc value then it will remove all the abc values from the multiset.**

**Episode 28**

**STL QUESTIONS**

Candie question



**If we use sets, then the 2 2s will be combined and the output will be wrong, hence we make use of multisets.**

While(t--){

Int n, k;

Cin>>n>>k;

Multiset< long long > bags;

For(int i=0;i<n;i++){

Int candy;

Cin>>candy;

Bags.insert(candy);

}

Long long total =0;

For(int i=0;i<k;i++){

Auto last\_itr = (--bags.end());

Int max = \*last\_itr;

Total = total + max;

Bags.erase(last\_itr);

Bags.insert(max/2);

}

Cout<<max<<endl;

}

**Episode 29**

**NESTING IN STL MAPS AND SETS QUESTIONS**

Map<int,int> m; normal map

Map<pair<int,int>,int> m; nesting in maps

When we compare two values in pair, then firstly the first member is compared and if its equal the second member is compared.

Similarly,

Map<set<int>,int> m;

For example if we want to store some names along with their marks we can create a map like this

Map<pair<string,string>,int> m;

**Episode 29**

**STACKS AND QUEUES**

Stacks – LIFO data structure **last in first out**

Think of a bucket in which we are filling water.

In stacks, we can access the size and topmost element of current stack.

Functions—

Push

Pop

Top

SYNTAX-

Stack<int> s;

s.push(2);

s.push(3);

s.push(4);

s.push(5);

while(!s.empty())-returns a Boolean value

{

Cout<<s.top()<<” “;

s.pop();

}

Output- 5 4 3 2

Queues are FIFO data structures. **First in first out**

Functions-

Push

Pop

front

SYNTAX\_

Queue<string> q;

q.push(“abc”);

q.push(“bcd”);

while(!q.empty()){

cout<<q.front()<<” “;

q.pop();

}

Output – abc bcd

VECTOR FUNCTIONS

Is\_sorted

\*max\_element