Don't watch my A2Z DSA Course

https://www.youtube.com/watch?v=0bHoB32fuj0&list=PLgUwDviBIf0oF6QL8m22w1hIDC1vJ_BHz&index=1

So if you're looking out for a structured DS algo course, I must say you're on the correct channel.
So in this video, I'll be telling you how to learn DS algo from A to Z in depth.
Yes.
So first of all, where to learn not on the YouTube channel because it is not structured.
You'll have to head over to the website, take you forward dot o r g might be seeing it on the screen.
The UI might be different some months down the line, but doesn't matter which course is it.
There's something as A to Z DS a course, open it up and as 455 modules covering 18 or 19 topics.
Every topic has sub modules like sub parts as well.
So there are a bunch of problems that you will be solving in this particular course.
Is it paid?
No, it is absolutely free and it covers a lot more problems than what the paid courses will do.
You can buy any of the paid courses and you can have a comparison done as well.
So the next question, how do you get the best output of this particular course?
First of all, the course has like every topic has some theory and every topic has some problems rather a lot of problems.
For the theory part, what I will say is, watch out the videos just to watch out the videos and do not die deep.
Just know the topic and then start solving problems because in DS algo like theory is not that important.
Solving problems is important because that will help you to develop the problem solving ability.
Got it.
So watch out the videos.
They do not die much and then start solving problems.
Stentwise.
Pick up a problem according to the step.
Try out that problem now when I am saying try out that problem means if you have an extreme
naive solution, extreme brute solution, hit it.
Convert your thought process into the code, write it and submit it, doesn't work, doesn't matter.
Read the blocks, there are blocks attached to every problem and solutions are categorized if a problem has multiple solution, you will find categorized so you can read out all of them.
If you are not able to understand, watch out the videos and for every problem that you are not able to understand or there is something new, create nodes so that you can revise afterwards.
And then again, retry whatever solution you have learned, retry it without watching the video without watching the blog, retry it.
This should be your method of learning.
Got it?
And this is how you can get the best output but in order to get a accelerated output, what I will recommend you is to get a mentor from pre-placed, a pre-placed is a website which offers amazing mentors, the mentors who are working at fan companies or in-depth top product

based companies.
So if you get a mentor, what will happen is they will have a personalized road back along with DSA system design, low level design, whatever you require, they will give you a personalized
road map and they will have a deadline check.
One week for arrays, after one week, hey did you complete arrays, let's have a small test.
After that, they will be giving you a lot of resources on system design, low level design, DSL go and other things, project ideas.
Since this people work in industry, they know how everything works.
So they will be giving you project ideas as well.
After that, once you're prepared, you will be able to have mock interviews with them before the real interview.
Resurvey reviews, according to the jobs that you are applying, you can ask them to review your resumes, you can have reference from them as well and from their friends as well.
And right at the end, you can have a personalized discount code from them.
So if you are liking a mentor, ask them for a discount coupon code, they will be giving you.
If they do not, you can use striver 1000 for an extra 1000 credits.
So I will be leaving out the link in the description.
You can book a free trial session with any of the mentor, it's absolutely free, go and check it out.
So let's head over to the next part covered to write scam.
So there is a lot of misinformation in the market because it's not paid, it's actually taking down a lot of businesses.
As I said, we have 90 lack visits on this course in the last 1.5 years.
So first of all, the first thing that everyone is seeing is this course isn't C++.
Absolute garbage, absolute nonsense.
So I have one video, the initial video that says C++ in one shot, that's it.
That's it.
I haven't made the video Java in one shot.
But apart from this video, you pick up any topic, I'm throwing an open challenge, pick up any topic, arrays, trees, DP, graphs, any of the topics.
You'll find that I write pseudo code, I don't write code in C++.
And in my blogs, we are attaching blogs, notes to every problem, we find C++ Java, Python JavaScript codes.
So we are not teaching C++ Java.
The entire course is in pseudo code, pseudo code, you're focusing on problem solving, right?
The second thing is it's not beginner friendly.
This is something which everyone, like the influences are spreading to because it's a free course.
So what I'll ask you to do is, I'll not say anything.
Reach out to any of your seniors who are, who got recently placed, ask them, hey, there's this guy's driver.
Should we follow him?
Should we follow the strivers A2ZDSC course?
You can find reviews on mobile, you can find reviews on Twitter, link in anywhere you wish to post a tweet, people will tell you the reviews of this course.
Do not ask an influencer or anyone who is running a business around this one, got it?
It is absolutely beginner friendly.
That is why it has full 55 modules, but I start from easy, then go to medium and then I cover a lot of advanced problems with a lot of paid courses in the market of a lot of free courses in the market, what they do is they cover up basic, easy, in-depth and when

it comes to medium and hard, they tend to skip.
You can pick up any of the free DSAC courses, they will end up at 171, 182, 200 at max and they'll be skipping the advanced problems or they'll be doing three or four of them.
But over here in our course, we do a bunch of advanced problems.
So it has stuff, but in order to exceed, you need to do duffer things, got it?
We need to solve all four 55, the answer to that is, no, technically, this course has around 250 to 300 concepts of problems, for every concept, I have multiple problems.
So if you're understanding a concept, this might have a four follow up problems, so you can solve two and you can skip the rest two, that is your wish.
It's not necessary to solve 455, I've given them for practice for your betterment, got it?
The next thing, along with the course, what else should you do?
One thing is an absolute must, point is, head over to lead code, so basically lead code, there's a website as lead code, head over to code forces, give every point is, so there is something in the market that like a lot of beginners think that I should learn everything and then I should start appearing for contest, do not do that.
You know basic programming, start appearing on the contest, even if you're able to solve one problem that is absolutely fine, give every contest for the next two years, give every contest along with learning DS algo, two years down the line, you'll come back and say me, you were correct, giving contests actually helped me to learn because they face a lot of new concepts, a lot of new unseen problems, so you'll learn a lot, lot more, got it?
So I'll be leaving out the link of this particular A2Z DSC course in the description, I'll be leaving out the link of pre-placed, you can book a free trial with any of the mentors you wish to and get on board it with them, if you haven't followed me on social media, I'll be leaving out all the social media handles in the description as well, with this I'll be wrapping up this video, make sure to follow the website, not the YouTube channel, with this I'll be wrapping up this video, let's meet in some other video, till then, bye take care.

How to setup VS code for DSA and CP | Input / Output split format

https://www.youtube.com/watch?v=h3uDCJ5mvgw&list=PLgUwDviBIf0oF6QL8m22w1hIDC1vJ_BHz&index=2

So in case you're looking for a setup which works something like this where the code is on the left, the input is on the right and the output is on the right and they're splitted across screen and you input something and you write the code and then you click one button and then the output is visible.
If you want to set up something like this, this video is going to be about setting up a similar kind of setup.
But if you're that in case you're for the first time here, please make sure that you check out drivers A to ZDS a course.
Now this is the world's most in-depth DS algo course.
Why do I see that?
Because the course has 455 modules, yes, we have solved over 455 problems and all the videos
are free.
You get premium quality articles as well.
Over there we have C++, Java, JavaScript, Python codes covered as well.
So you get everything at one place and you don't need anything else.
You can straight away prepare for it and this is targeted for companies like Google Lamas in Facebook and all the other product based companies in the market.
And it's one other thing that I want to clarify.
There's a lot of rumor that this particular course is in C++.
No, that is not the truth.
So the first couple of videos are in C++ where I'm teaching for loop, while loop because they're pretty much similar in C++ and Java.
So just plot in C++ if you know Java, you can easily grasp it, right?
But when I move to recursion, when I move to arrays, when I move to binary search or any other thing, each of those lectures are recorded.
With logic and then pseudo code, I do not write code in any language.
And after that in the description, you'll find notes where you'll find C++, Java, JavaScript, Python codes in those notes.
So this course is not language dependent.
The first couple of videos might be, but the entire course is not only the basics like the for loop, while you're printing stuff isn't C++.
After that, everything is pseudo code, yes, pseudo code, I focus on the logic.
So please remember this and the link is in the description, make sure you check it out.
So let's get back to the VS code and try to set it up.
So first I'll be setting it up for C++, after that I'll be setting it up for Java and for any of the language.
If you see these two setups, you will be able to set it up.
The first thing that you'll do is you just open the VS code editor and right after that you open a folder.
So what I'll do is I'll go over here and I'll open C++.
So I'm open C++, after that I go to the extensions and I install C++ extension.
So search and you find an extension from Microsoft, I'm already installed it.
So please make sure that you install it once you've installed it, go back to this for a little and try to create a file.
So maybe I'll just create demo.cpp.
So that's your file.
After that create another file with input.txt, after that create one more file with output.txt.
Done.
Once you've created these three files, go to view and go to editor layout and say split

left.
Once you split in left you'll find demo is on the left as input and output.
So just cross it out.
So you have input over here and output.
So you have input and output on the right, you want to split them across.
So go to view again, editor layout and say split down.
Once you split them into down.
So you can just drag the output and that should be done.
So the input is on the top and the output is over here.
Let's take a quick break before you move ahead.
So nowadays everyone is focusing on DS, Algo, but people are ignoring all the other aspects.
Something like projects is going to be of extreme importance when you're going in for placements.
So what I'll recommend you is to get a strong portfolio with projects and with DS, Algo as well.
What you can do is you can check out cryo do's project hub over there.
They have structured a lot of projects according to domains.
Let's say we go to web dev and you can see a lot of project ideas and they are given in a structured manner.
So you can follow them and build some amazing projects and you can build a portfolio like Parvita Sharma over here.
You can see is portfolio is build some amazing projects.
So you can have a portfolio, something like this when you're going in for your placement.
So let's talk about cryo do as well.
So grader do is world's first experiential learning and upscaling platform over here.
You don't typically watch videos.
You basically learn by doing.
So they offer you programs where you'll be working like a software engineer does at Amazon Netflix, Google, etc.
You'll be building some amazing projects.
You can see on the screen and you'll be building under mentors who are working at Google Amazon Microsoft Walmart people.
So talking about placement stats, cryo has already transformed 1000 plus careers.
Cryo graduates from various backgrounds have already been placed in 750 plus companies.
So they offer you fellowship program in software development.
What you can do is you can book a free trial session and test it out for yourself and at the same time make sure that you bookmark the project's hub.
I'll be leaving the link in the description and you'll also find the free trial session link in the description.
Make sure you check them out.
Now quickly try to run a code.
Sort of done as I've copy pasted a demo code.
Now the next step is to run this.
But before that you have to go to terminal, configure tasks, there's something as create, task, click it, others.
Once you click on others, you see there's a VS code folder and there's a task or JSON.
Remove this configuration and go to the description and be pasting the configuration.
Just paste it and remember one thing the compile and run should be together.
Leave it and close it and write after that I'll be pressing command plus shift plus B.
But before that let's give an input 13 command shift and B and you will see that there's an output quite simple isn't it?
So the configuration that you see over here that's for C plus plus for MacBook in the notes

like in the description I'll be leaving some notes I'll find for windows as well in case you have any other operating system what you need to do is this copy paste the configuration that I've given in the notes take it to chat GPT and whatever is your system.
Ask chat GPT can you give me the EQ valent for my system and he'll give you this copy paste and it will start working.
So this is this I use set it up.
Remember the star's dot JSON is under your C plus plus folder in case you're switching folders you'll have to again repeat the same thing got it?
So that was easy for C plus plus isn't it?
Now it's time for Java it's very much similar I go to open I go to desktop add open one of my folders you can open from anywhere and next thing install an extension Java extension pack this one I've already installed it so I'll not be reinstalling it right after this what's the next job?
Just add a Java file probably I can give demo dot Java.
After that add an input dot TXT which takes your input add an output dot TXT which takes your output.
Go to view editor layout split left once you've done this this is how it looks like and go to view editor layout and it says split down.
So once you do this you just drag it off and move this input.
So you have input and your output ready let's quickly paste a demo Java code.
So I've written a demo Java code now what is the next step to run it?
So I'll go to terminal and there's something that's configured task create task dot JSON others and over here you paste this configuration I'll be leaving this configuration in the links so this is what you'll paste and this is this basically command which kind of takes the input.
So the file names have to be input and output and you can save it and close it.
So once you click save you can see that it is under a dot VS code folder and this is under the folder that I opened initially so if you're changing folders it will not run you have to reconfigure the task got it.
So what I'll do is I'll give an input over here try to run it command shift plus B and you can see that it is running this is how easy it was to set it up.
So yeah this was it quite simple isn't it in case you're someone who uses python or javascript the blog links will be in the description and in case you run into any of the errors please do use chat GPD that will be helping you out if you're still watching please do consider giving us a like and if you're new to our channel do consider subscribing to us and make sure you check out strivers a to ZDS because that can be a one stop solution if you're preparing for placements we haven't followed me on Instagram Twitter I'll link in all the profiling serving the description when this I will be wrapping up this video

Time and Space Complexity - Strivers A2Z DSA Course

https://www.youtube.com/watch?v=FPu9Uld7W-E&list=PLgUwDviBIf0oF6QL8m22w1hIDC1vJ_BHz&index=4

If you want to come back to the channel, I hope you guys are doing extremely well.
Then this lecture will be learning about time complexity and this is the continuation of strivers A to Z, DSA course slash cheat.
So if you remember in the previous video, we have covered all of these green tick mark things and this video will be covering the time complexity, learn basics and then analyze in next steps.
What does that mean?
Teaching the basics of time complexity, what is time complexity, how to compute time complexity
and all the terms related to it and then eventually when you go through all the later steps at every problem that we solve, we will be discussing time complexity of every problem in depth.
So eventually when you complete all the steps or all the problems, you will have a very very good hold on the time complexity part.
Now in this video, I cannot teach you time complexity of very complex codes because you do not know them.
So once you learn the complex codes and recursion, backtracking, we will be discussing time complexity
in depth over here, but as of now, just let's start with the basics.
So the first question, what is time complexity?
Now why is time complexity required first of all?
Now whenever you are going to an interview, you might end up writing a piece of code.
Now that code, how does an interview judge that code?
This is where they will analyze the time complexity of the code.
They are not going to take your code, run into a machine and see how much time does the machine take and on the basis of that, tell you that your code is right or wrong.
That is not going to happen.
This is where something like time complexity comes in.
Whatever code you are going to write in an interview, in every interview, they are going to analyze you, what is the time complexity and what is the space complexity.
And then on the basis of that, you will be judged, right?
So wherever you write a code that might end up taking, let's say, two seconds on a machine, a might end up taking three seconds on a machine, might end up taking five seconds on a machine.
So the time taken, will you call it the time complexity?
No.
You will never call time complexity as time taken.
Now you might ask, but why?
If a code is taking certain time, that should be its time complexity.
Let's give you a very, very easy example to explain why not.
Imagine, I take this code, I take this code, this can be any code, any code.
I take this code and I run it on two machines.
One is the old Windows machine and the other one is the new MacBook machine.
Now I am sure the old Windows hypothetically might end up taking two seconds to run that old, well, the new MacBook, which is a lot more in terms of the, like, lot more latest in terms of the configuration might end up taking one second.
So does it mean that your code is taking one second or does it mean that your code is taking two seconds?
It might happen.
I bring up some other machine the next day.

It might end up taking lesser than one second.
So depending on the system, the code ends up taking a different time.
So that is why time complexity cannot be set as time taken because it is dependent on system.
It is dependent on configuration.
Right?
So this is why that is, this is the first point.
Time taken is never equal to time taken, never ever.
So you might ask, if time complexity is not time taken, then what is it?
That is where I see the definition.
The rate, remember, the rate at which the time taken increases, the rate at which the time taken increases with respect to the input with respect to the input size rather.
So what does it mean?
Let's explain this with the help of a graph.
So again, let's take the same example, where I am saying I have a old Windows machine.
And I have a new MacBook, okay?
Now imagine you are giving the old Windows a size 10, just as hypothetical, a size 10, input size and I am plotting the graph.
And this is one second, this is two seconds, this is three seconds, this is four seconds, this is five seconds.
This is 10, input size 20, 30, 40, at 10 input size, it might end up taking two seconds, at 20, it might end up taking four, at 30, it might end up taking six, which will be here.
So you see the time increasing at some, at some theta angle, like it's increasing at a certain slope.
So this increase, this increase, this theta angle is what you call as the rate of increase.
This is what you call the rate of increase, okay?
Now let's see the new MacBook.
So if I take the new MacBook, the 10 might end up taking one second, the 20 might end up taking two seconds, the 40 might end up taking three seconds.
So it's here, here, here.
So this is the rate and this theta is the rate for the new MacBook.
So I can say the rate at which it increases time is what you call as time complexity.
Depending on input, if they are giving you more input, it might end up taking a certain more time.
So the rate at which the time increases is what is generally referred as time complexity of any given code.
The next question that might come to your brain is, okay, stable.
If we are saying the time complexity is not computed in terms of seconds or minutes, then how do we, how do we write it?
This is where something like a big, oh, a big notation comes in.
Now in all the interviews that you're going to give or in all the examinations, if you're writing a piece of code, they will ask you to analyze the time gone.
That is when you do not like, you will not say two seconds.
No, you will say it in terms of big notation, every piece of code takes time in terms of big notation.
What is big notation?
I'll just give you a priming example.
So it's like a capital O open parenthesis and a closed pattern.
This is what big notation is and inside it, you write whatever the time is taken.
You write the time taken or the hypothetical time, which I'll show you how to compute.
Let's take a very, very small example to compute the time complexity of a certain code.
So imagine I write the fall loop.

I've already taught you loops in the previous video, I equal to one, I lesser than five, I plus plus and then I write C out and my name, Raj, okay, so can you tell me what is the bigo of this?
This is nothing but the number of steps that this code will take.
First step is very short assigning.
The second step is comparison, the third step is printing, right?
And then you come across, and this is your fourth step.
And then again, you check, this is your fifth step.
And then again, you print, this is your sixth step.
So this is how every step is computed.
So whatever is the total number of steps, that is what you write inside it.
And that is what you call as the prime bigo notation.
But you cannot just keep on counting step one, step two, step three, step four, step five, step six, step seven, that is insane.
You just cannot keep on manually counting the steps.
This is where three rules come in and I'll write the rules.
Three rules are very simple.
Always compute time complexity in terms of worst case scenario, time complexity to be computed
in terms of worst case scenario, why worst case I'll explain you.
The next one is avoid constants, avoid constants, the third and the third is avoid lower values.
So if I have to compute the time complexity or the bigo of this piece of code, let's look at it on a very high level, can I say this fall loop is running for five days?
I can.
At every time what is happening?
It's like increment, check, print, increment, check, print.
So can I say for every time it's operating twice, for every time it is operating three times increment, check, print, five into three, can I say this?
I can.
So can I say a total of 15 times, that is, are you write it?
You write it as p go of 15 is what you write, this is time complexity, it runs for 15 times, but this is a number, this is a number, you generally do not represent your code in terms of numbers.
Okay.
So let's, let's change this.
Now can I say this, if I change this to, if I, if I probably will change this to lesser than n, lesser than equal to n, so now can you say how many times will this loop run?
I know one thing, this is going to run for, we go off n times, the loop, the code will run for n times n iteration and at every iteration, one, two, three things will happen, but you obviously, this is what will be the time complexity, we go off, we go off, we go off, we go off, we will be your time gone, so whenever you are saying we go, you go to the code and you see how many times it is running and then you write we go off, we end as your time complexity, does that make sense?
Does it?
Okay.
So let's go back to the three points that I have written, always compute time complexity in terms of worst case scenario, now there are three things, one is the best case, one is the average case and the other one is the worst case.
Now let's take a snippet of the code and explain you what is best case, what is average case and what is worst case, okay, let's, so this is the snippet of the code.
Imagine, the snippet of the code is very simple, it states of Marx is less than 25, print grade D, it states of Marx is less than 45, print grade C, if Marx is less than 65, then you go ahead and print grade D, or else you go ahead and print grade A, so let's talk

about the best case scenario, what is best case, when the program takes the least amount of time.
So over here, can I say, if someone gives you an input of Marx equal to, let's say 10, what will happen, this will execute it and grade D will be printed and none of these lines will be executed, none of these lines, so can I take, can I say, number of operations will be just one, a check and a print, which is like two operations, which is like two operations, can I say, imagine, if I give you Marx equal to 70, so at Marx 10, it was be go of two operations, correct, be go of two operations, now if I give you Marx equal to 70, what will happen, check, balls, check, balls, check, balls, it comes in, three and then four, can I say, four operations by here, like one, two, three and then goes to else, somewhere around four operations, so it takes four operations, so whenever you are analyzing the time complexity, will you say that my code, transcend B go of, or will you say that your code runs in B go of four, obviously you will say B go of four, because that might be the worst case, the computer might encounter, when it is given an input, so always, yes, always compute the time complexity in terms of worst case scenario, and if they
will ask you the best case, you know the best case, whatever is the best case, like in this case, is the first statement getting executed, which is be go of two operations, so you have got an idea about the best case and the worst case, you know, what is average case, there's nothing but the best plus worse and divided by two, it's the median of it, so pretty self-explanatory, why if I give you a question, why do we compute it in terms of worst case, the answer is again self-explanatory, if you are building a system, will you build it
for one person, or will you build it for one million person, if given an option, if given an option, you will say one million, because you want to build a system which can scale up, you always think of the worst that can happen, that is why when I talk about time complexity,
your code extreme worst case is what is considered while computing time complexity, now let's talk
about constants, I'm saying avoid constants, why am I seeing that, let's take a shuttle example,
let's say something like 4n cube, something like 3n square plus 8, let's say this, so over here, 4n cube plus 3n square plus 8, these are the number of operations that your code is taking, hypothetically, I don't know what these are for, I'm just taking a hypothetical, we go expression, now imagine if I tell you that the input size n, the input size n is somewhere
around 10 to the power 5, I'm saying 10 to the power 5 is the input size, so when I see 10 to the power 5, and you put that in, what will it be, 10 to the power 15, 10 to the power 5 cube, 15 plus 3n square, that's 310 to the power 10 plus 8, tell me a very easy thing, 10 to the power 15 plus 10 to the power 10 into plus 8, will this 8 have any significance in the operations, this is in itself such a huge number, if you add 8, will it, will it matter to them, I take it like billion like whatever is 10 to the power 15, I'm not sure what it is, if I billion trillion whatever, like more than that I guess, with that, a plus 8 does it have a significance, it doesn't, if you're earning 1 million and I give you 1 buck, that will not have any significance, that is why, we do not consider constants, we do not consider constants, now in terms of code, imagine I say you something like this, INT x equal to 2, so this is the modified code, this is the modified code, x equal to 2 and then the follow, so this is one operation, it's something like n into 3 plus 1, the time complexity updated will be n into 3 plus 1 operation and this is the one operation, so you never ever take this unit operation, so you will not consider this, you will say no, it will still stay as n into 3, I will avoid constants because whenever the input size is very large, those constants have very less significance, got it, now let's take the next thing, avoid lower values, let's take the same example, I'm saying 10 to the power of 15, 10 to the power of 15,

if you add 10 to the power of 10 to the power of 10 to the power of 10, will it have any significance, it won't, so adding 10 to the power of 10 to the power of 10 to the power of 15, 10 to the power of 15 will have absolutely no significance, it is similar to saying I will add, let's say 1, 2000, maybe even less than that, it's similar to actually saying 1, 2, yeah, I'm adding 1 to this much or 1 more 0, yeah, will it have any significance, no, so adding this one to this one, will it have any significance, no, the number of operations will be like slightly more, because 10 to the power of 15 in itself is such a huge number, but this is why again, it's only very important, avoid the lower values which does not change its
significance by much, avoid lower values, correct, so these are the three points that you will always keep in mind while computing the Biko notation, because Biko notation as what, you will be expressing your code and interviews always, all the problems that I'll be solving, I'll be telling you about Biko notation only, apart from Biko notation, there are certain other terms like theta notation, omega notation, they'll find a lot of other terms in text books, are they required in interviews, no, no one is going to ask you these things, no one, just for knowledge, let me tell you, the Biko notation is always the highest complexity, or the worst case complexity, remember this, worst case complexity or the highest or generally known
as the upper bound complexity, it's known as the upper bound, the omega is the lower bound, as I
said the kind of the best case, is the lower bound or the lowest bound or whatever you can, and this is nothing but the average complexity,
just a quick disclaimer, there are a lot of theoretical stuff revolving around Biko, theta, omega, a lot of limit derivations, but I'm not teaching you theory stuff, I'm not teaching you for your semester examinations, I am teaching you for interviews, I'm preparing you for the
DAC coding rounds and they will not be asking it, over there they'll be asking you programic logic, they'll have to solve problems, they won't be asking you, tell the formula for Biko notation
in limit terms, no one is going to ask, might be asking semesters, for that you'll find different teachers on YouTube, if you're learning from me, you're learning for interviews for coding rounds,
for problem solving abilities, not for your semester exam, so please do not comment, that you did not teach the mathematical derivation, because that is something which is not required while problem solving, right, so you know how to compute Biko now, let's do one thing,
let's quickly solve some questions, so the first snippet of the code for which you have to find the time complexity, imagine inside your main, you have something like i and i equal to zero, i lesser than n, i plus plus and for i and t j equal to zero, j lesser than n, j plus plus
and imagine it has a block of code like something, a single line of code and this is your entire snippet of the code, okay, this is your simple block of code, you can consider this to be taking very constant time, constant time, so this can be avoided, tell me the time complexity, you know time complexity will be computed in terms of Biko notation, tell me the time complexity
of this particular code, pause the video and maybe think about it and then see if you are correct or not,
so how do you analyze the time complexity of snippets of the code, it's very simple, analyze how
the code works, so if you see the outer loop is starting from i equal to zero and going till n, that's definitely running for n types and the inner loop is going from j equal to zero till n, this is also running for n types, now if you're a beginner you might find it a bit difficult, so what do you want to do is, you will say at i equal to zero what is happening, let's see, whenever i is zero what is happening, whenever i is zero it goes inside and j starts its value

from zero and goes till n and this completes n iterations, can i say when i is zero j goes from one zero one two three dot dot dot till n, can i say, and then again when the i's value increases to
one, can i say, again goes from zero one two three because it's inside it, it's re-itreating again can i say if i is two again it again goes from zero till n, can i say till i is n minus one,
why n minus one because it is less than n, can i say again goes from zero till n for other n minus one to be very accurate can i say this, so what is happening, every time it is running for n,
n, n, n, every time it is running for n and n, so can i say it's running for n times first
then for another n times second time then for another n times second and then dot dot dot dot till
n times how many times is it running, can i say it's running for exactly n times, so can i say this is nothing but n into n, yes thereby n square is the number of iterations that is
yes there are other things as well like increase comparison but we usually do not consider which rule avoid lower values if you remember because three into something won't matter that much
because you can avoid it what i can say is it's running for n square yeah if you want to consider
all these constant operations you can do it that is your wish i usually do not do it so
be go of n square is what is the time complexity of this particular, let's say the next example and
try to compute the time complexity of this one, so it's like the first loop is definitely running for n and second loop is running from zero to i, so let's analyze slowly and probably we'll get the time complexity, so can i say if i equal to zero j is nothing but from zero j is only running for zero can i say this because when i is zero j is running for zero j less than equal to zero it will run for just one time well when i becomes to what happens j now runs for zero sorry
when i becomes one rather when i becomes one what happens j now runs for zero and one can i say
why because i is one it's like zero lesser than one correct nice i can i say if i is equal to what happens j now runs for zero one two so when i at the end becomes n minus one what does
happen j will be zero one two dot dot n minus one can i say this is how many operations it is taking again i'm ignoring all these constant operations i'm taking the overall iterations
now the first time it's one iteration next time it's two next time it's three four until n so
can i say the number of iterations the first time it is one the next time it is two the next time it is three the next time it is four so on till n at last and i see and do you remember the formula for this particular thing some of the first n natural numbers simple maths n into n plus one all divided by two which is nothing but n square by two plus can i say n by two
thereby this is the exact time complex but we can avoid if you remember the rules
constant to be a like the smaller values to be avoided so it will like n square by two
so this is the time complexity again if you try to remove the constants like one by half you can say the time complexity is near about we go of n square but if someone does ask you the exact
time complexity this is the exact time complexity but according to the rules this is what it boils down to so these were the basics of time complexity okay now next i'll be doing all these patterns in
the next lecture well be writing all the nested for loops and there you can analyze time complexity
in a much better way but eventually all the time complexity terms like login and login n square n cube n square login all of these things you will see as i proceed with the playlist but do not want to rush you in i just want to you to get an overview of what is time complexity how to think on it how to compute how to avoid terms these things should be crystal clear to you so i'm

assuming the time complexity our stuff is crystal clear to you they'll now be moving to something
like space complexity again what a space complex it's the memory space that your program takes
it's the memory space that your program takes in a very very naive term when i talk about memory
space again it will vary from machine to machine so you cannot be dependent on the machine and
that is why again in order to complete space complexity we will be using the bego notation again
so bego notation is kind of very very important in all the interviews bego in terms of time bego
in terms of the space so we'll again be using bego notation and you know the reasons why not in
terms of kb mbgb etc so what is exactly the definition of space complexity it is nothing but auxiliary space plus input space what is auxiliary space the space that you take to solve the problem
the space that you take to solve the problem input space the space that you take to store the problem
space that you take to store the input rather not the problem so let's give a very very small example in order to explain what is auxiliary space and what is the input space so i'll write a very very small thing imagine i'm taking an input of a and b in java you know how to take it and c plus plus you know so i'm assuming i'm taking an input of a so i'm not writing any syntax
because then it'll be c plus plus a java specific as one due to learn pseudo code and logic
so imagine i give an input to variable a imagine i give an input to variable d and i'm saying c equal to a plus b now i'm saying c equal to a plus b so the auxiliary space that you're using is an extra c variable this is the auxiliary space why because you're using a c variable in order to
solve the problem you cannot use that is okay i'm saying if you use any extra variable or extra space to solve the problem that is what you refer as auxiliary space what about the input space
this is one and this is one so both of these can be referred as input space whereas this can be
referred as auxiliary space and combined i can say this is the space complexity or i can say we go
of three over here because you're using three different variables you can convert them into bytes
if i take an integer you can convert them into bytes i'm not concerned about this
in an interview you will have to say it in terms of b go of something not in terms of bytes kb mb so for an example if i define an array of size n it means i'm consuming b go of n size or n space complexity so in an interview if you're solving a problem which uses an array of size n you
say b go of n is the space complexity that i'm using as simple as that now a thing that i want to
highlight over here is if you're doing space complexity a lot of times it will happen is imagine you're given an input to a imagine you're given an input to b and if task is to some add both the
numbers a lot of people say i'll do this b equal to a plus b so summation of a plus b will be stored in b and i'll not be using the third variable it is okay to do it is okay but this is something you should avoid it takes lesser space but in an interview the interview will say this method is not done why is it not done just think in perspective of software engineering imagine

you are writing a code for a big mnc and they're giving you some data they've given you some
set of data and they want you to do something on that data and you're like you manipulate the
data over here you change the data you changed b this is why never ever do anything on the input
very very crucial rule in interviews never do anything to the never do anything to the input
unless the interview sees anything to the input because data should not be touched
you can do whatever you wish by taking the data but you cannot do on the data because if you
manipulate the data it has an issue because in software engineering that data might be used in some
of the place so never ever manipulate data this might save you a bit of space for the interview
will end up rejecting you because this is a very very bad practice so never ever do anything to
the input given always take extra variables extra array extra matrix it might be tempting that if
you use the same input variables it will end up taking lesser space but it's it's not a big concern
if you're using b go of n space like if 2n is used by taking an extra array it is okay this is
okay no one is going to reject if you're using b go of 2n instead of b go of n if the interview
specifically says do it on the input then you can use it otherwise do not and you can tell that
to the interview you can explain that I don't want to I don't want to tamper with the data that is
why I'm taking an extra variable in order to solve this problem keep this in mind forever this
was about the high level idea about space complexity and the practices you should follow regarding
the in-depth knowledge when we solve different problems on different data structures you will see
how I'll be explaining space complexity in depth because the more problems you solve the space
complexity gets in your head in a much much better way so before ending up this lecture in case
you went into competitive programming then remember one thing over there you do not run your
quotes on your MacBook or your windows whatever code you write you send it to the server
there is a server you send it to the server everyone sends to the same server and these servers
in competitive programming or let's say you're using lead code or let's say code studio or let's say
gfg any platform if you're using they have their servers and most of the servers take one second for
10 to the power 8 operations remember this most of the servers most of the server take one second
for roughly 10 to the power 8 operation might be 10 to the power 8 plus something might be 10 to the
power 8 minus something but roughly it is 10 to the power 8 operations is what you can execute in
one seconds on the server most of the servers are relatively the same and if they're saying two
seconds it doesn't mean 10 to the power 16 it doesn't it doesn't it means two into 10 to the
power 8 operations if they're saying five seconds it means five into 10 to the power 8 operations
do not do 10 to the power 8 to the power do not a lot of people do that okay so this is something
just keep in your mind because a lot of times you will see time limit one second they'll write it

they'll write it in your let's say coding round that the time limit given to you is one second where you should analyze that the time complexity of your snippet of the code or your entire code
should be roughly we go off 10 to the power 8 operations if in a coding round they are stating time limit one second it means roughly when you compute the time complexity of your code
avoiding constants avoiding lower values it should be we go off 10 to the power 8 operations so with this I will be completing the step 1.1 guys I hope you've understood everything
and the next lecture will be about do all these patterns yes I'll be doing all of these patterns each of 22 of them in the next lecture just in case you have understood time complexity and space
complexity in depth we have a ritual on taking you forward on every lecture if you understand everything write at the end of the lecture you drop a comment understood in the comment section
so that I get that believe that you guys are understanding I get that motivation that I come back
and record these videos so please please do consider dropping the comment understood
if you're new to our channel please please do consider subscribing to us if you haven't checked out strives a to ZDS a course link will be in the description add you get everything on the description my link in Instagram Twitter everything you can follow me at any any of these places
and if you're learning online you can always use the hashtag that is over here and with this I'll be wrapping up this video experience about the video

Solve any Pattern Question - Trick Explained | 22 Patterns in 1 Shot | Strivers A2Z DSA Course

https://www.youtube.com/watch?v=tNm_NNSB3_w&list=PLgUwDviBIf0oF6QL8m22w1hIDC1vJ_BHz&index=5

Hey, we are back to the channel. I hope you guys are doing extremely well. So this is another lecture of the strivers A to Z DSA course. And the previous lectures we have completed step 1.1. So we'll be starting off with step 1.2. And this is nothing but do all these patterns.
So you can see that this page contains all the patterns that you have to do before starting off data structures and algorithms. Now when I talk about patterns, why are they important in order to start off with DSA? Whenever I talk about DSA, whatever topic we dynamic programming,
graphs, trees, arrays, binary research, one thing is common in them. That is the loops.
If you do not understand loops in depth, if you cannot improvise with loops, then there will be a
problem in order to do, like there will be a problem when you do DSA. That is why patterns are
so important. Because if you want to master something like loops, you want to understand how
loops work. If you know how to play around it, then you can get very, very good at data structures
and algorithms. And for loops, you have to do patterns. This is the reason we will be solving a bunch of patterns. Are they asked in interviews? The answer to that is, no, I have never seen
an interviewer ask a pattern-based question. Until unless it is a TCS interview or something like a service-based company. So patterns are not asked in interviews of the top tier companies,
but patterns do play a significant role when you are starting off with data structures and algorithms.
So let's look at the first pattern. The first pattern is something like this. So if I draw
that in our iPad, it is something like this. Now, before moving on to pattern, I'm going to teach you
four steps. I'm going to teach you four steps that will help you to print any pattern in the world.
And the first point is, so generally, all the patterns that you will be printing will have
nested loops. Again, very important to remember, will have nested loops. And most of the pattern,
most of the pattern will have two loops. That is the outer loop and that is the inner loop.
Okay. Now, the outer loop is specifically for the lines. The outer loop is specifically for
the lines and the inner loop is specifically for the columns. By the way,
I can say these are the rows and these are the columns. So the four rules are based on it.
So the first rule states for the outer loop. Yes. For the outer loop, count the number of lines.
Count the rows, count the number of lines, whatever you can write. Count the number of lines.
And this will determine what your outer loop is going to be. At the point number two,
what did I say? Pattern means nested loops, nested means outer and inner. So for the inner loop,
for the inner loop, what you have to do is you have to focus on the columns, focus on the columns
and connect them and connect them somehow to the rows. That's the rule number two. You have to
focus on the columns and connect them somehow to the rows. I'll understand other afterwards.
Now, whatever you're printing, print them inside the inner loop. Whatever you're printing, print the,

let's say, over here, it's a start that the places can be one, two, can be a, b, c, d. Whatever you're
printing, print them inside the inner fall loop. I'll show you an example. Everything will get terrified. The fourth one is observe symmetry in case of some patterns. Observe symmetry. So this
is an optional one. This step is an optional step. Why? Because this step will be only applicable
to certain patterns, not to all the patterns. So this is an optional step. But the first three steps are mandatory. The fourth step is an optional one. We will solve patterns which will be requiring
observing symmetry. So let's come to the first pattern. I'll be explaining this in depth.
And then the next patterns we will be moving faster. So if you look at this pattern,
let's have the first rule. For the outer loop, count the number of planes. So if you see,
it is actually doing some tasks for four times. It is exactly doing the task for four times.
So the outer loop will be running for four times. Or you can say it will be like I equal to zero. I lesser than four and I plus plus. Can I say this is what the outer loop will be? So this is going to be our outer loop. Pretty simple and straightforward. This is your outer loop. So I can say,
I have somehow done the step one, which is printing for the outer loop. Okay, so we have we are moving for the outer loops. Now the next step comes for the inner loop, focus on the columns.
For the inner loop, focus on the columns and connect them somehow to the rows. Let's see, we have four rows. And what is happening at every line? At every line, what is happening? We are
printing four stars. At every line, what is happening? We are printing four stars. At every line, what is happening? We are printing four stars. So can I say, but try to write it? Can I say,
for the zeroth row, I'm printing four stars. For the first row, I'm printing four stars.
For the second row, I'm printing four stars. From the third row, I'm printing four stars.
Like if I take zero based indexing, can I say I'm printing four, four, four, four? Can I connect it somehow to the rows? Yes, the total number of rows were four. And at every line, we are printing
those many stars. So can I say the inner loop will be very straightforward and it will be running for
J. Do not take it. Okay, because I is the outer loop. J equal to zero. J less than four. J plus plus. Can I see this? This will be an inner loop. So can I say, whenever I is zero, J will be zero, one, two, three. Whenever I is one, J will be zero, one, two, three. Can I say whenever I is two, J will be zero, one, two, three. Whenever I is three, J will be zero, one, two, three. It will be
such step number two is also completed because we have connected at every line or whatever
at every column, whatever is happening to the number of rows somehow have connected them.
Now since I've connected them, what's my next step? Print, what are we printing inside the inner
for loop? You have determined what has to be done. Now, what are you printing stars? Go ahead and
just print those stars. That's it. Now there is a certain thing that you have to keep in mind.
You print them inside the inner for loop, but make sure since everyone is an individual line at every like once you have done everything for the first line, you have to actually go to the new
line. Once you have done everything for the next line, you have to go to the new line.
So what you will do is, you will just give away a print new line over here. If you're using Java, it will be system.out.println. If you're using C++, it will be Cout and what this will do is,
now let's do a typical drawing. Let's see what happens. For the first time, I is zero. Whenever I

is zero, what happens? j is zero. So j is zero. Goes and prints the star, right? And then comes back,
j is one, prints the star, comes back, j is two, prints the star, comes back, j is three, prints the star. Once j is three, and it comes back, j becomes four, and this condition is false, and this fall loop is done. Now, what happens? It goes to see out of endel. So that means it goes to the new
line. When it goes to the new line, i is this time one, and whenever i is one, it again starts j from,
it again starts j from zero, and again starts printing, j becomes one, again starts printing, j becomes two, again starts printing, j becomes three, again starts printing, and then the endel
happens and it goes to the next line. Then I becomes two. Again, the same thing will happen, and I become three. Again, the same thing will happen, and eventually I will become four, and this particular condition will turn out to be false. And I can see my entire pattern is printed on the screen. Quite easy. So this was step number three. Do we have, do we have to observe
symmetry? No, in this case, no symmetrical observation has to be done. I'll show you patterns
where you have to do it. So we'll be super quickly coding this up just a case, fall loops and C++ and fall loops and Java are same. So it doesn't matter which language I'm coding, and you can code it in Java as well. I'll be leaving the notes link in the description, which will also have the Java code. If you own the Java code or the Python code, you can just open it up, and you can keep it on the side just to check out how everything works. Okay. So we have int main in C++ in Java. It will be public static void main. So imagine, imagine, I just write a function, which is going to do that task for me, which is void print pattern one, I'll probably print
pattern one. And I see the pattern was something like zero, four, I++, and then I go like INTJ equal to zero, J less than four, J++, and then I go like C out, and this goes like star, and this is where C out, and it happens. And in order to execute this pattern, I have to call it from here. Correct. Now what I'll do is I'll go ahead and run this task. Let's see,
on running what happens. So when I run this, you will see this is getting printed, but usually you do not do it like four or five. What they do is they give you an input. So probably imagine I give you an input five. So what happens is you take n as the input, and this input is passed over here. You remember function. So you give n over here, and if you write n over here, now what will happen is based on how many rows you want, this pattern will get printed. So now
again, I'll go and run this task. So you need five lines, it gets printed. You need several lines, it will get printed. So depending on however lines you want, you can just give it as an input, and this function will make sure this particular pattern is printed. Now we need to learn about the
online compilers. So when you go for interviews or whenever you're preparing for placements, you will not be coding in domain. This is something you'll not be coding up. Okay. In compilers or in coding rounds, what you just need to do is you just need to write this function. All of these things will be hidden. So let's go to an online compiler. So this is code studio by coding ninjas, which I'll be using for this particular video. Now this is similar to all the other compilers, and you have to use online compilers when you're preparing for coding interviews. You just can't
code in your local compilers. So get used to this as of now. So what I'll do is I'll just take this particular piece of code and I'll try to copy this over here and see if it is running fine. I'll rather run it. In the moment I run it, I see actually wrong. And this is where you can actually check
out where it is running. So they're saying they are giving two three five. Now you're saying what's

we just had one value. Now what they are doing is they're probably doing it something like this.
I'll explain. Two, it's saying the program runs for two times. Then they're giving something like three and five, right? Three and five. So what they are saying is basically, hey listen,
this time we'll take test cases. Our pattern function should run these many times. And what they do
is they go ahead and probably do it something like this. T and i++ and inside this, inside this, they take this particular n and they call the function. So basically two times you will have to print patterns. First time for three rows, the next time for five rows. If you run it in the local compiler, see what happens. Let's run that in the local compiler. So the moment, sorry my bad,
so the moment I run that in the local compiler, you will see what happens. First,
test got printed next this. So this is the meaning of test cases. Test cases means your program internally
is run for multiple cases that they will never run it for one test case. In the real life or in the coding world, there will be thousands of test cases. Any program that you run will be run on,
any program that you write will be run on thousands of test cases. Like over here, your program is run
on two test cases and this is how the backend looks. In the backend, they will write something
into main. They will take the test cases and they'll call your functions for every other test case and your function has to perform the task for the first time three rows for the next time,
five rows. Now you might be thinking, okay, but I was right. Why was it wrong? If you see, they're clearly showing that there has to be a space because if you look at it, they won't space,
what you can just give is there's a space and once you've given the space, you can go ahead and
run it again. The moment you give a space and run it again, you see that it is running fine. You
can go ahead and submit this code and you will see that this is giving you a correct answer. So
from now onwards, you should always practice on online coding platforms because when you go for
interviews, this is where you will have to give your online exams or whatever it is. So please start practice. So I hope you've understood the test cases stuff and how the backend looks in an
online compiler. What you have to write, you don't have to write the entire code, you just have to
write the functions snippet because the other things they will be taking care of. Let's look at the
second pattern. Let's try to again implement this same number of steps. So the second pattern states,
okay, we have one, two, three, four, five, five rows. The first loop is going to be a dead straight forward. So let's do one thing. Let's quickly copy paste this stuff and yeah, let's do it pattern number two. I will do it. I'll do it print pattern number two. I'll probably just yeah. So what is the first loop? First loop is saying again, end rows. That is very obvious because we are seeing that it
is for one, two, three, four, five rows. There's something which we are definitely seeing. Let's look at
the step one and step one. The outer loop is done. Let's look at the inner loop and try to connect it
somehow. So if I look at for the zero throw, I'm just printing one star, right? And I say for the zero

throw, I'm printing one star. For the first row, I'm printing two stars. For the second row, I'm printing three stars. For the third row, I'm printing four stars. For the fourth row, I'm printing five stars. So can I see, can I see, I have to run it for once for the first line and run it for twice in the second line, run it for twice in the second line. So can I see, if it is line zero, it runs for or if it is line one, it runs for once, once, twice, price, four, five times. Very simple, isn't it? So can I say, if I write the inner loop as, and dj equal to zero, and j lesser than equal to y. And then inside this, if I give something like c out of star space and over here, I ended with this run. This has to run, isn't it? This has to run. So I've given the test cases everything. Will it run? Let's give it a try and see if it is running fine. Let's run this task.
So if you see over here, this is, this did not run because why?
Because we are calling print one. So we will call print two, my bad. It's called print two, and let's see if it is, if it runs, it did. It's very simple, isn't it? When i is zero, it runs from zero to zero. When i is one, it runs from zero to one. When i is two, it runs from zero to two.
So we just provide, yes, the step number two, we just looked at the step number two, and it was
pretty simple. The step number two clearly stated, connected somehow to the rows, and we were seeing zero
one row. It was pretty simple. Just run it for zero one time, two time, three time, and that was very
evident to be connected to the, out of eye loop. And the third step was print them inside the loop. So the three steps were done, and i was easily able to print this particular pattern.
Again, let's, for the online compiler, let's take this, for the online compiler, let's take this, and try to take it over here, and try to run it and see if it is running fine.
It is, and let's submit this. So for some of this, this is running fine. Yes, it is. So
that was the pattern number two. Now let's go to the next pattern. That's the pattern number three.
It's something like one, one, two, one, two, three. Again, step number one, for the out of loop, count the number of lanes, one, two, three, four, five. So it has five lanes. So the out of loop is going to be very, very straightforward. So let's again, take this and copy this for the pattern number three. So this is going to be pattern number three, and the out of loop is going to be pretty straightforward, zero to n. Yeah, let's take this off, and this is going to be three.
So the out of loop is very simple. But now let's try to connect the, the inner loop, like the columns to somehow the rows. Now I know this is row number zero.
For the order of simplicity, let's take this as row number one. For the row number one, I'm actually printing from just one for the row number two,
I'm printing one two for the row number three, I'm printing one to three,
for the row number four. I'm printing one to three four. So can I say for every row number, I'm actually printing from one to a number. Can I say for every row I'm printing from one to one, second row, one to two, the fourth row, one to four, fifth row, one to five. Can I say for every row because I do the outer loop runs for every row so can I say for every row and we are
taking simply city like one so we will do one lesser than can I say for every row the columns are going from one till the row itself and then g plus and what are we printing we are printing nothing but let's look at this what are we printing one two three that's nothing but the inner loop increment that's it let's go ahead and print j if you print j well it works it ideally should let's quickly run this and see if it is running fine run task and compile it let's see so we are seeing here it does it does run fine so what I'll do is I'll take it to the online compiler paste it over here and see if it is giving us the correct answer it is some of this and it will be giving you the correct answer so the third pattern is also done now let's look at the fourth pattern let's look at the fourth pattern what difference do you find it is quite similar to the previous pattern but the thing is on every row we are printing the row number we're not incrementing it like one two three can I say we're just printing the row numbers so if I go

back to
our stuff and do the pattern print of four let's copy paste the same thing this yeah it's
copy is the same thing and close this does this pattern four and instead of this can I say on
every column or on every row I'm just printing the row number in itself I'll try to now run this
and see if it is running fine if I do run this you will see that yes yes no it is not because
why did oh sorry print four my bad let's do this run task and see is it running fine it is
very simple you see one one method and all of them are running absolutely fine I'll just go
over here
and paste it and really try to submit it number super confidence I'll just directly submit it
it is running fine the pattern number four is done now let's talk about the pattern number five
when I talk about the pattern number five step one very simple for the outer loop count the
number of
lines one two three four five that's n lines so without thinking anything go ahead and copy this
and do the pattern number five it is this and let's be like okay this is running for five
for sure the outer loop is done let's now analyze for the inner loop for the inner loop can I say
on the row number one there are five it's printing five times let's let's analyze one prince
five ten the first row prince five step the second row kind of prince four steps the third row
kind of prince three stars the fourth row kind of prince two stars the fifth row kind of prince
one star so if I have to somehow somehow connect them to the rows can I see can I see this
the total
rows minus the row number plus one is what you're printing on every row again observation
very simple
observation can I see this I can why because if it is one and the total are five five minus one
plus one is five is what you're printing here if it is two five minus two plus one this is nothing
but four is what you're printing so this is what you're printing these are the number of stars
you're
printing so can I say the inner loop is going to be very simple j equal to it could be like zero
and j can go till end minus i plus one j plus plus again I'm starting from one so my formula
is this if you start from zero your formula might be something else what matters is how many
time
you're running okay see out of star and then go ahead and see out of end I will not be
submitting
this time on the online compiler all the problem links you will be finding it in the description
or it did not work again print five let's quickly compile this I'll not be submitting this code
on the online compiler anymore all the problem links you can find it in the description and
you can
go ahead and submit them by yourself when you're practicing so the pattern number five is
also done
you see one rule solves almost all the problems let's look at this again very similar it's like
one two three four five one two three four so it's like printing from one to that row number
instead of this I have to print the numbers so it's pretty very easy this copy paste and it
goes like pattern six over here I can do a pattern six and this time you can just go ahead
from here
and this is like this and you can be like okay j and this should be fine let's quickly run this
for pattern six again very simple so we do this yeah it does run by right so if you see by the
way let's for easy understanding let's do one test case really seven yeah that should be
give I should give you an easy understanding so if you run this you see everything is running
absolutely like this so this is also done now this one is where you have to think of it so
probably pause the video and think on how you can solve this so what is the first rule the first
rule is very simple count the number of rules that's five first loop is definitely going to be
something from like i equal to let's say zero and till five which is like you can run it till four
so this is something which I know so if I try to write down the row numbers it's like zero one
two

three four that's very obvious what's the second rule the second rule states for the inner loop focus on the columns what is happening and connect them somehow to the rules first thing that
else is how many columns are so can I say it's like one two three four five six seven eight nine
so can I say in total in total there are nine columns right and if I carefully observe what are we printing we're going to printing space stars space space space stars space space space space space space
space space and stars space space space so we're printing nothing but space stars space so if you logically think inside it if we can print space and if after that we can print stars and after that if we can print space I think our job will be done now let's analyze for every row how many space we have to print how many stars we have to print and how many space we have to print
if we are able to do this I think we will be able to write our inner loops very easily isn't it
so can I say but the zero zero column what are we printing printing one space two space three
space four space it's like four space one star four space right okay can I say over here it's like one two three three and three three comma three comma three can I say over here it's two comma
five comma two okay can I say over here it's one comma seven comma one can I say over here it's zero
comma nine comma is it nine it is zero can I say we're printing these many space these many stars
and these many spaces yes we can now we need to find what's somehow connected somehow to the rows
I think we can we know how to connect if we look at this this is nothing but going in the reverse
direction of the number of rows the number of rows are from zero to four it's pretty can I say the
formula for the first space is nothing but n minus i which is just imagine for the first case i will be zero so if n is five that those are the five rows five minus i minus one is what this will be very obvious so I figured out the space to be this you can just do some maths and you'll be
able to figure it out assuming n is five over here so this is very simple five minus zero minus one
for the first time next time five minus one minus one next time five minus two minus one so so space has been computed but now we have to think about the stars how do we compute the stars
because the stars are like one three five it's somehow increasing for zero it is one
for one it is three for four it is nine it's something like one more than twice can I say it's something like two into i plus one is what it is can I say this two into i plus one
two into zero plus one is one three two into three is six plus one is seven so can I say the stars
is two into i plus i can I've kind of figured out the relation somehow connected to the rows just run the inner loops the only differences this time the inner loop will first print the space then the stars and then this so there will be three different inner loops will we force this in a loop then this in a loop then this in a loop this one for space this one for star and this one for space very simple I just broke down the problem into simpler one once I've done this I think my job will be done so let's quickly write the pattern number seven right let's write the void print pattern number seven let's take n so I know one thing I will be starting from zero again I'm doing it in terms of zero based indexing you can go ahead and do it whatever based
indexing you feel like one basis let's print the space then the star and then the space okay so

this one will be for j equal to you know the formula is n minus i so j zero j lesser than equal to n minus i minus one and then j can go as plus plus very obvious okay let me just quickly check okay j lesser than because this should be fine and the similar thing will be at the last as well and for the stars is very obvious that you go from sorry j goes from zero j goes on till two into i plus one and then j plus plus why do you see this will be space so you can
just go ahead and print it something like this this will also be space and this will be stars so go
ahead and print the stars and once everything is printed you write before moving to the next row
you move on to the next line so let's go ahead and print the part number seven
let's print the part number seven let's go to the terminal and run the task and see where it's running by so if I run this you see that this particular pattern is indeed getting printed so the main task is to break down the problem into this into parts and then follow the rules that I taught you okay this one pretty similar they inverted what can you do it in the inverted fashion I think you can what is the difference and this time you start from this will be zero space this will be one space this will be two space so it goes in the opposite way and the stars are in the opposite fashion as well so think on this pause the video and think on this and let's see if you can do this so coming back to the pattern what's the first law the first law is very simple zero one two three four so the first loop is going to be super easy to predict now we know
we are printing space then we are printing stars and then we are again printing space this is something we know for sure now the question is how many space initially if we look at this initially
for zero we are printing zero space and we are printing like nine stars and then zero space one
it is something like one space seven stars one space for two it is like two space and then five and then two for three it is like three this is three and this is three for four it is like four this is one and this is nothing but four so the spaces one is pretty simple it's pretty simple so we just go back and we know the pattern seven is kind of similar we'll just copy paste and we'll just take this off and make this pattern something I know for sure is
the space one is equivalent to the row if there is the zero through there'll be zero spaces so this
won't run this is something which you know for sure the only problem is the stars the formula is
going to be super simple if n is five for zero it will be like two n minus two i plus one you can try doing it and if you apply this formula for the zero through it is going to be like two into five ten minus two into zero zero zero plus one it's like ten minus one nine for this one it will be like two into five ten minus two into two which is four plus one which is five ten minus five is five so it's very simple what I did was in the previous problem I knew this was nothing but two i plus one and I saw it is reducing so reduce it from twice of n it's quite simple so what I'll do is I'll go over here and I'll say two n minus two into i plus one again you can do it as as you prefer there's no hard and fast rule that you have to do it in this way only you can assign variables reduce them and do however you wish like so I'll go and print this print
eight terminal run task clear and run
hey looks like it is not getting printed in this way okay so it has to be two into n
no I think you should print run task clear and run okay it does print so this is how easily we can do the pattern number eight that's time to do pattern number nine where the fourth rule will
come in suppose the video and maybe think how you can do on this pattern number nine looks
a combination of seven and eight isn't it so what you can always do is can go ahead and

combine both
of so imagine I see let's first print print number pattern number seven according to this particular n and then print pattern number eight so if I do this will it run
hi logical it should and you will see it does so sometimes you have to be smart it's not necessary that you always write patterns you can always combine them as well then combine two
different patterns to generate one as well so keep that in mind and that's actually a good way to write patterns combining so the pattern number nine is completed time for the pattern number
10 again this looks quite similar to one of the ones that we did isn't it but is it a flip pattern like this one was a complete flip pattern like the same lines if you see this line and if you see this line they were similar right over here we have all we have a right angled line but the bottom
one is like there are two same lines of equal number of stars here there is no so you cannot merge
one like straight like straight triangle right angle triangle and the inverted right angle triangle you cannot do that over here so you can do it with some fluctuations but I'll still
say let's look at this pattern so n is given as five first thing to observe is the rule number four symmetrical now this pattern is a symmetrical pattern observe symmetry where does it get this is the this is the point where it starts getting symmetrical so whatever logic you have to write will be okay till here and then you have to flip the logic so observe symmetry you have observed symmetry that is done what is the outer loop let's see one two three four five six seven
eight nine nine rows is what you have to run nine rows is what you have to run so can I say if you
are running nine rows it's very obvious to saying you're running two n minus one times that is what
you outer loop runs so going back to the code the outer loop is pretty simple pretty straightforward
two less than two again you can go one base indexing as well that is your choice done one less than
two and minus one so this is what the outer loop will be so let's quickly write this one two three
four five six seven eight nine so we have written the outer loop now let's talk about the inner loop now
when I see inner loop what does it mean it means this one for the first time I'm bringing one stars
second I'm bringing two stars third I'm bringing three stars fourth I'm bringing four stars
fifth I'm printing five something I'm sure is till here till the symmetrical portion the number of stars that I'm printing is definitely equal to the row number so let's let's say number of stars that you're printing is equal to the row number which is i so we can go ahead and say i and it j equal to one j less than equal to stars is what you're printing so go ahead and say see out of star it's actually correct right and if you run this let's see what we'll have we go ahead and try to run this this will act print but this will print something like this whereas from the row number six you have to break the symmetry from the row number six you have to break the symmetry
let's try to analyze a formula for row number six we need four stars seven we need three for eight we need two for nine we need one I think is ready evident the formula the formula is crystal clear to n minus i as a formula isn't it those many stars is what you require if you exceed the row if you exceed the fifth row because this is your symmetrical symmetrical position
from six stone rules these are the number of stars is what you will be printing so can I see if the row exceeds the n-throw if the row exceeds the n-throw then the stars that you will be printing

will be nothing but two into n minus of the row number right can I say this or means i oh yeah can I see this I think I can if I go and run run the task well this print this yes it does
very simple if I try to write it in a much simple like you get you can just write it in other ways as well turn your operators whatever you wish to but this is how you can easily print denested for loops so we are done with the pattern number 10 as well let's look at the pattern number 11 okay
we have done a similar pattern over here the right angle triangles
I hope I don't remember if you call that so row numbers are like five that is quite easy so going
ahead void print 11 i and t n and you can say I know one thing i equal to 0 i less than n i plus plus this is definitely the outlook there is no doubt with the audio let's look at the enaloo how's working can I answer through the zero throw starts with for the second row it starts with one for the
third row it starts with sorry for the fifth so it's like so can I say for the zero through
it starts with one for the second row it starts with one for the fourth row it starts with one can i say for all the even rows for all the even rows it starts with one okay so I'll be like what's the start point? Let's keep start as one and can I say this if I am an even row
if I'm an even row the start is actually one or else the start is actually zero can I say this maybe you can define this somewhere here and assign it some dummy value if you wish to so can I say if it's an even row the start is one as the start is zero that is something which you know for every column now what happens it prints for one time two time three time by flipping
flipping flipping one zero one flip flip flip so can I do this I think I can it's very easy what I'll do is I know how to print the right angle triangle that's something like this this this this this this and you just print the start isn't you just print the start I'll post this you'll print and
then but if the start is one it stays as one at the next step it has to be zero it flips you can say flip flip is very easy you can write start equal to one minus start this line will flip one to zero and zero to one you can do maths and you'll find now let's uh print pattern eleven yeah
let's try to print it and see if it is running fine you see those three or four rules
will actually print everything did we print it properly it's quickly have a check
I should have started from one looks like the first line was not printed okay my bad it should be
this j less than equal to i so runtaus and clean and run this will be printing it one zero one the pattern is printed pretty easy isn't it let's look at the pattern twelve uh you can probably pause the video and try out the pattern yourself this pattern looks so this pattern uh so this pattern
kind of looks similar to the pattern that we did uh if you remember yeah where there was space
stars space and over here can I see it is like number space numbers so if I try to write all this pattern can I say that if I can figure out the way of numbers space numbers I think my job will be done that's crazy but for the outer loop that's very easy you just figure out
how many can I say I will be just running the outer loop for four times the outer loop is very simple
it runs for four times the outer loop will be running for four times right so if the outer loop is running for four times let's look at this for the first row can I say I'm printing one number right and one number where between that I'm printing some species let's count the number of species one two three four five six six is the number of species that I'm printing of it let's look at the next one I'm printing two numbers two numbers and let's count the number of species
one two three four okay I'm printing four species let's count the next one three three and the spaces are one two next four space zero and then four something I'm sure is the number and this one

are very easy because it's equivalent to the row number so what I'll do is I'll go ahead and first
I try to write this pattern and then we can figure out the other one I'll take the INTN back into a I can just yeah so print 12 and I know one thing you can just start from one again you can
do zero based indexing that is your choice so first is numbers then a space and then is numbers so can I say the numbers are going to be very simple it can go from one till the I and J++ they could go ahead and say J and this not this yeah perfect and similarly you can just
do it in the opposite fashion because if you look at this it's like three to one it goes in the opposite fashion so it's sky easy you just go from I to one and you do J minus minus so this is what you do for the numbers but space is what you have to exactly determine the next thing that I
need to do is I need to figure out a formula for space because if I can figure out a formula for space it can be very easy so can I say the formula is start from six and then do you see a change
it reduces by two it always reduces by two for sure right so if you can somehow figure out the first number and then you can just reduce it by two at every step I think the job will be done
so can I say the first number first number is nothing but two in two whatever is n minus one that's the first number very simple because if n is four it's definitely six right if n was three it would definitely be two sorry four so two into n minus one and then at every step you just reduce it by two why it's simple so what I do is I say he listen I know the space initially that you have to give is this sorry my bad to into n minus one there's something which we know why don't you go ahead and say okay please go ahead and print those many species so it's like
space then you can do j plus plus and then you can do c out perfect and write up to every step
you definitely are going to do an endel but at the same time you say space minus equal to which is equal to space equal to space minus two and you can go ahead and say print pattern number
and over you can go to the terminal and say run task clean and run
if you clean and run you actually see that this pattern is getting printed for the fifth one as well very easy isn't it so with this we will be wrapping up the pattern number 12 as well the next pattern is I think very easy you can try it on yourself so the pattern number 12 is done
now time for pattern number 13 again a right angle triangle so should not be a big issue five rows so we'll be running it for five times for sure so let's go ahead and do it for pattern number
13 so void print pattern number 13 i and d n everyone knows what's the outer loop going to be
it's going to be something like from let's see from one yeah we know it's going to run for n times let's look what's happening first it goes like one then like two then like three something for sure is for the first row we need one for the second row we need two elements for the third row we need three elements for the fourth row we need four elements for the fifth or fifth row we need five elements something for sure is the out the inner loop is definitely going to
run for the number of row times if it is first row one time if it is the second row two times if it is third row three times that is something which each of us know but coming back what am I
printing like one two three four it's a number which starts from one so I know the printing is going
to happen somewhere here this is where we're going to print something for sure and this is

go where
so we know the number starts from one so why don't we keep the number which starts from one and
we can print it something like this and at every step it's getting printed we just increase it by one
does this work first time one will come in print i'll go to every time the printing occurs one two three four
white simple and this is pattern number 13 for you so if I go ahead and say terminal run task and compile and run this will actually run it does and if you want some clarity probably we can give
a space and if you give a space this is going to have much more clarity yeah white simple pattern
number 13 down let's look at the pattern number 14 again a right angle triangle so the outer loop
is pretty pretty pretty it's going to be till end so what print pattern number 14 i and tn
and the same will go zero based indexing again you can go one base zero based the certain code will change as long as you're printing it it doesn't matter this is something which we will write now let's look at what are they printing it inside at every row they're printing one
12 material so number of rows equal to number same as writing will try to do a starting with A and we're going till three if it's a third row starting with A we're going till four if it's
the fourth row that is something for sure so can I say we can start with A now actually we can run
uh something like character CH equal way and we can say CH lesser than equal to A plus of i and
we can do CH plus plus now a lot of you might not understand so it starts with A and it says A plus
i so imagine i is something like two that's basically saying if I take you back to the iPad it's basically saying start with A go till A plus if i is two A plus two what does A plus two means in computer what happens is A is okay two places ahead B C that's actually C that loops from A
till C exactly if you're saying A plus zero it loops from A to A it loops from A to A and
in in ask your computer it's stored like A B C D you can probably google search about ask and you'll get a better idea it's just to be sick uh so now if you give C out of character
and you say this and you go ahead and see C out of NL let's see what happens I go over here and
I just terminal run compile and run well this get print here it does because the looping is done on the character this time so so the pattern number 14 is completed time for the pattern number
15 it's similar to an inverted right angle triangle which I think we have done in the pattern number five is we have so it's we just have to print like A B C D E and then A B C D the first time it has to be for the zero throw it has to be N times then N minus one times then N minus two times then N minus three times then N minus four times so it's it's it's pretty simple
so what I can do is I so what I can do is I'll go over here you know the outer loop is definitely going to be something like I equal to zero and I lesser than N and I plus plus and about the inner loop it's something like character C H A C H lesser than equal to for the first time
if it is zero if you see it prints all the five it's like A plus five times is it no four why
because it's like if I just write it it's like A B C D E is what you're printing so this is A plus four it has to be A plus four not five be careful about this so it's like
this is N next time it will be A B C D that's like A plus three got it that's like A plus whatever N is minus this is the zero throw this is the first row N minus row minus one please please got it into the match of towards it's like A plus N minus this this and then
you can do a character plus plus you can go ahead and do this and then this and then you can go

ahead and do C out of N del this once you've done this go ahead and print this and you will see
this is giving you a correct top so with this we have completed the pattern number 15 as well again the pattern number 16 is a right angle triangle but this time it's like A B B C C C D D D D D
and then five E's very simple you start with A and then you go to B and then you go to C it's like for the zero throw the character will be A for the first row the character will be B for the third row the character will be C it's like A plus the robot number right isn't it because
for the first row if for the zero throw it's A it's something like A plus i if the first
row is just B it's something like A plus i because A plus one will be B correct so again very simple
I'll just go ahead and write void print 16 i and D N and then four i and D i equal to zero i less than N i plus plus and the character that you will be printing will be definitely A plus i and how many times in a right angle triangle I don't need to say it anymore now everyone knows this now
C out C H and then you go ahead this and then you go ahead of N L once oh my bad it's like zero
once you've done this you can go ahead and say pattern trend 16 and this will get printed oh sorry
I think A did not get printed why because this not you did A V B C C C triple A triple A done
so the second the pattern number 16 is done let's look at the pattern number 17 the pattern number
17 is again similar to if you remember I think yeah this one where you have space star space it's like space alpha bits space but it has some symmetry so you have to observe symmetry
and then you have to print it so the outer loop is again very straightforward what will be the outer loop obviously zero one two three four the outer loop is going to run for N times the outer loop runs
for N times what are these spaces spaces spaces spaces spaces so it's like first you have to figure out
spaces then alpha bits and then spaces again so if you remember from the pattern number as I said
it was this one pattern number seven let's go to pattern number seven and look at the formula so it was like the spaces was N minus I minus one so I can say the spaces is N minus I minus one
that is something which we can copy probably let's copy this entire thing and paste it
and this will be pattern number 17 so instead of this we have to somehow figure out the characters
because as of now we have printed the spaces we've printed the rows we have to figure out the
characters so if we go to pattern number seven we know the number of characters are going to be
see that's two I plus one the number of characters are going to be two into I plus one that's for sure because this is one character because this is one character this is three characters this
is five characters which is like two into two plus one we know the number of characters I say Can I say one thing for sure, if I am having 5 characters in this room, out of which if I start with character A what happens, if I have 5, if I have 5 symmetry of some symmetry, if you observe symmetry, it breaks at half, right, it breaks at 3, so can I say, if it has 5, 5 by 2 plus 1, till this I will have a plus plus, I will have a plus plus, post this I will have a minus minus, isn't it, if I start with character A, I print it for whenever I am at the first time, whenever I am at the first time, I print character A, then I do a plus plus, whenever I am at the second time it prints B, then I do a plus

plus, whenever I am at the third time, it's C, and then I start doing minus minus, so that it becomes B, and it gets printed at the fourth, again do a minus minus, at the fifth it prints A, so can I just, can I just assign a character, I think I can, so what I will do is, I will go and say, hey, can you please assign a character to yours, it will be like, okay, let's do it, if you are assigning a character, can you go ahead and, you know, how many times you have to print, that's 2 into I plus 1, that's the number of times you have to print, so let's at first print CH, and see if it is getting printed correctly or not, it's part number 17, I will go ahead and see, as of now we have not decremented it, it's printing correctly, the space and everything is correct, if I do a character plus plus, and if I try to run it, it will be printed correctly, but it's getting incremented, so I need to decrement it, so I know, I know what is the break point, the break point is this, what is the number of points, it's like 2 into, these are the number of characters, y2, till here it increases, if you carefully observe, if you carefully observe, it is 5 by 2 is 2, till to 8 increases, post 2 have to decrease, so that in the next step, it is decreased, so can I see, break point will be this, so if your character J, your character J, again probably just to have a better understanding, you can just do it like this, 1B is the next thing, if your character J is lesser than equal to the break point, it's fine, do a character plus plus, if it is not, then do a character minus minus, quite simple, and now I'll go ahead and say, can you now run it, and see it is running fine, it is, it is running absolutely fine, so very simple, you just have to do a bit of mathematics, bit of observation, bit of subject, and the four rules that works, so we are done with one more part, nice, let's look at the next part, pattern 18, it's like E, and D, CD, let's start from E, and then it's like D, and it's like CD, then it's like B, CD, and it's like A, B, CD, so how do you do it, try to pause the video and think it because this is interesting, so that's it right under triangle again, is that out of doubt, no, no, it's the inner loop that we have to actually prepare, so can I suppose the zero, can I suppose the zero, it's like going from E, for the first, we're actually going from D to E, for the second, we're going from CD, E, can I say it's very easy, we actually going from E, like E is definitely the last one, and before this we're going from E minus i, isn't it, like if we do one, if we do E minus one, that actually brings you to D, for two if you do E minus i,
it brings you to C, done, again, very simple, what I'll do is, this is pattern number 18, so I'll go ahead and code, void print 18, IDN, and I know the order loop is this,
I know the inner loop will be from character, CH will be E minus i,
character will definitely go into E, character plus plus, and you can definitely print the character,
and going ahead, you can definitely do an NL, so print 18 is what you have to do,
and I'll go ahead and run this, and see if it is running fine, it does, so I can say one more pattern,
done as well, let's come to the next pattern, which is pattern number 19, now this is again an interesting pattern, but I guess you as of now have an idea how to do it, so if you carefully look
at this, we see a symmetry, the symmetry is something like this, this is an upside one, and this is
the downside one, so let's first solve this one, and then we can solve the downside one, okay, so I'll just be drawing the symmetry line, so above the yellow line is what we will be solving at first, and then we can solve the bottom one as well, so above the yellow line what are we printing,
let's look at this, we are printing stars, right, we are printing spaces, and we again printing stars,
and the outer loop is definitely very obvious, if n is 5, that's the number of times the outer loop
will run, stars, spaces, stars, so that is something which you have figured out for the pattern number

16, sorry 19, so we can go ahead and say i and dn, and we know that this is going to run for n times for sure, so we can be like first we got to print some stars, then spaces, then again stars, that's for us to look how many stars, how many spaces, and how many stars, so if I look at
this for 0 through, I'm actually printing 5 stars and 5 stars, which is for 0 I'm printing
5, 5, and 0 spaces, for the first column, we're actually printing 4, 4, and 2,
we're printing 4, 4, and 2, for the second column, we're actually printing 3, 3, and 4 spaces, isn't it, for the third we're printing 2, and 6 spaces, and 2, for the fourth,
so 0, 1, 2, 3, 4, 4, we're printing 1, 1, and 1, 2, 3, 4, 5, 6, 7, 8,
that's easy, that is super easy, isn't it, what is this, it's like n minus i,
so can I say the stars is n minus i, number of stars is what I'm printing, i am n minus i is the number of stars I'm printing, and can I say 4 spaces, we can start with 0, and we can keep on upgrading them with plus 2, as simple as that, so I'll go back to the code, and I know for
stars, it's something like i and dj equal to 1, and we just now figured out n minus i is what we are printing, so go ahead and say star, so we can just go ahead and control c, stars is done, for spaces, we know initial spaces is nothing but 0, so you can go with initial spaces, and there will be no command line, i and dj equal to 0, and probably j less than any space, and g plus
plus, and you can go ahead and say 4, rather see out space, and once everything is done, my bad,
once everything is done, you can just increase the spaces by equal to 2, and you can do a cout
of n l, by the initial spaces, what do you have to increase, perfect, once you've done this, this is pattern number 19, let's see if this gets printed, the terminal run task,
compare it, let's see, so if you try it, so this will be j by the way, that there was a type of 4, if you try to run this, we will actually see that it's getting printed properly, the first thing is printed, now let's look at the second, and that's again easy, that is again easy,
so if i try to again write the star spaces combo, stars, spaces, stars, it's like one star,
or again we can restart it, we can just start it from 1, 2, 3, 4, 5, it's like one star,
one star, one, two, three, four, five, six, seven, eight spaces, there is six spaces, two stars, two, three spaces, four stars, you can just do the second half as well, second half will
one base indexing will be a better way, and then we know the initial space, we can again, this initial space equal to eight is what we can keep, and then we can just copy paste this entire
stuff, because it's symmetrical, and over here instead of n minus i, this will be exactly i, and this will be minus equal to 2, because it's reducing by 2, I'll go ahead and better run this, and see if it is running fine, you see this pattern, so what i can say is the pattern 90 is also completed, now it's about the next pattern, which is the pattern 20, so can i see over here, it's again the same pattern, but this time, can you break down into symmetry, you cannot because
there is just, if i, if i carefully show you, there's just one line, over here you could because it was exactly symmetrically opposite, but this is not symmetrically, there is just a single line, had this line been twice, we could have done it accordingly, but no issues, maybe we can figure
out a V, and we can do it, because these are nothing but similar to writing, isn't it,
it's similar to writing, so we will first have a look in over here, assume it's fine,
that's like 1, 2, 3, 4, 5, 6, 7, 8, 9, something I know for sure is, 2n minus 1 times is what the outer loop will run, 2n minus 1 times is what the outer loop will run, this is something for sure,
so if i look at this, we actually bring in like 1 star, a lot of species, how many let's count,
1, 2, 3, 4, 5, 6, 7, 8 species to start of it, 1 star, 8 space, 1 star, 2 star, 2 star, 6 space,
3 star, 3 star, 4 space, 4 star, 4 star, 2 space, and then 5 star, 0 space, 5 star, and then
4 star, so can I say, after, can I say after the 5th row, can I say after the 5th row,

there will be a shuttle change, the moment you are at the row, 5 or maybe n by 2, the moment
you are at the row n by 2, on the next step there will be a change, and the change is very short, the space will increase till here it was decreasing, the space was decreasing, but whenever
we reach here, the space starts to increase, the space will start to increase, and what happens
to the rows, it is printing 4, then 3, we had done this, yes we have done this, if we go ahead, we have done this in the pattern 10, so we will go ahead and look at the pattern 10, how are we
doing it, we were saying the stars will be 2 n minus i, the moment it crosses the nth guy, right, so this is what we will do over here as well, the moment it crosses this guy,
it is going to be nothing but the number of stars, till here it was very simple, whatever is the row number, that is the star, at the moment it crosses it will be 2 n minus i again very obvious, for the 6, 25 minus 6, that is 4 stars, done, so I have figured out all the formulas, everything, that is time to go and quickly code this up, it is like void print 20, i and t n, and I know one thing for sure, this is going to run till 2 into n minus 1 and i plus plus, there is something which we know, now what are we printing, stars at first, spaces and then stars, let us print the stars,
idle stars, the number of stars are very simple, the number of stars depends, like it will be rows,
but if the i has crossed the n, then the stars will be through in the sense i, stars will be
2 into n minus the row number, we can just print the stars now, for i and t j equal to
1, we can go ahead and print those many stars, and we can go ahead and do j plus plus, and see out of star, that is it, and over here you can just copy paste the same thing,
and you can see this, but this will print all the stars, what about the spaces, you know the spaces, how they increase, maybe you can keep the spaces initially as, if you remember the initial
space was 8, so instead of keeping it as 8, you can say 2 n minus, 2, I think we did a mistake by keeping the space as, initial space as, okay it was 0, this will not be 8, instead of this, this will be 2 into n minus 2, my bad, I did a mistake there, but nevertheless, so
okay, 2 n minus 2 is the initial spaces that we start off with, and then while going we can do j equal to 1, and we know these many spaces, the j plus plus, and you can just go ahead and say
space, at the end of the day you can say c out, and l, and if i is lesser than equal to,
like lesser than sorry, lesser than n, then you say spaces will be increased by 2,
l spaces will be decreased by 2, this is what you will do to the spaces, and this is pattern number
20, so let's go ahead and print this, and see if it is running fine or not,
okay, we have an issue, looks like there was an issue while printing it, why did it happen, let's quickly have a look, as i made it a mistake, space will be minest to here, and this is where
the space increases, okay, let's run it quickly, let's compile and see, okay this time it's perfect, so, so another pattern done, isn't the pattern printing super easy, so we are done with the pattern number 20, we will start with the pattern number 21, the image is wrong, i will get it fixed,
the correct image is this, because you have to print a square, so if you look at the pattern, you're given 4, such a 4 size square, square, so for a square you know the number, the outer loop
is very easy, the outer loop is going to print for n times, that is for sure, but if i look at the square, the fill stars, where does it have stars, obviously in the boundaries, that is easy,
fill the stars in the boundaries only, can i say i'll run for all the times, i'll run for all the times, that means i'll run for 4 times every time, i'll run for 4 times every time, i start from

something like 0, and then go on till 1, i'll run it, and then i'll have a loop, which again runs from 0 to n, i'll have a loop which runs from 0 to n, so can i say this, that i'll just fill the boundaries, and if i try to, if i try to write the index x, let's write the index x, it's very simple, if i try to write down the index x, it's like 0, 1, 2, 3, 0, 1, 2, 3, so can i say, whenever i is 0, which is this, or whenever i is equal to n minus 1, which is this, or whenever j is 0, or whenever j is n minus 1, can i say these are the cases of the boundaries,
these are the cases of the boundaries, because whenever i is 0, it runs j runs 0, 1, 2, 3, whenever i is 1, j runs 0, 1, 2, 3, whenever i is 2, j runs 0, 1, 2, 3, whenever i is 3, j runs 0, 1, 2, 3,
so can i say these are the cases, when we know we have to fill the stars as simple as that, so,
so what are you, so what are you waiting for, let's go ahead and print this pattern, it's like void print 21, id n, and then you say k, i know i have to print these we need, id j equal to 0, and then you say, hey listen, and if you have this, or it is, or you are this, or you are this, then only i will print a star, if not, then i know, you will have a space, and over here i can say, end, as simple as that, and i can say, 21,
fantastic, let's run, and this is,
this will be space, now let's run it, quickly run it and see if it is running fine,
yeah, yes, it is printing a square of the size 5, so, what i can say is, i'm done with the pattern number 21, as well, the last pattern, finally my, my voice is giving up now,
so, the first step that i will do is, again, it's pretty observation,
it will not come to you for the first time, so to not worry if it doesn't come to your head,
so what i'll do is, i know there are 7 rows, and i know there are 7 columns, or something like n equal to 4, so, what i will do is, at first i will subtract 4 from every value, and i'll get a new metric, the new matrix will be like, 0, 0, 0, 0, 0, 0, and then 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
So once I have subtracted four from everything,
like four minus one, this gives you three.
Four minus three, this gives you one.
So you just subtract four from every value.
So let's solve this and once you get the answer,
you again subtract four to get this very, very standard process.
So what I've done is I have subtracted n from it,
which is nothing but the four, the value
to get the new matrix, okay?
Once you get this new matrix,
you can again subtract the new matrix to get the value, right?
It's nothing but like this was the current value.
This is the current value and this is the new matrix.
So again, if you just take it to the other side,
it will be n minus new matrix equal to current value,
pretty, pretty obvious, pretty obvious.
So I need to generate this, generating this is very easy.
Let's understand how.
So I know in the matrix, it's like zero, one, two, three,
four, five, six, zero, one, two, three, four, five, six.
2D matrix, back step 1.1, I've taught you 2D matrix.
This is how the indexing goes like.
Seven rows, seven columns.
So the outer loop runs for seven times,
the inner loop runs for seven times.

This is something very short because you've got a print,
seven, seven.
How is seven related to four?
Very, very standard maths.
2 into n minus 1 is how it is related.
So if I go back to the code, I know the inner,
I know the outer loop, which is 2 into n minus 1.
I know, sorry, it'll go from 0 to 2 into n minus 1.
And I plus 1.
And I know the inner loop will go from 0 to 2 into n minus 1
and G plus 1.
This is something we know for sure.
Now let's have some observation.
What is this value?
Something to observe is, if I take this, the distance is 1,
the distance is 2, the distance is 5, the distance is 4.
So you take always the minimal distance, which is 1.
And you place it here.
Let's take one more example.
3, the distance is 3, 3, 3, 3.
Getting the minimal from this portion.
When I say this portion, this portion is nothing
but the row number 3, the column number 3.
And in order to find it, you find the top distance,
left distance, right distance, bottom distance,
and hitting the minimal of it.
And once hitting the minimal, you actually find this value.
This 2 is nothing but the distance from the bottom.
This 1 is nothing but the distance from the bottom.
The 0 is nothing but the distance from the top.
You have to find the distance.
And you know how to find the distance.
It's very, very simple.
So if I talk, my bad, if I talk about this guy,
what's this distance?
It's nothing but the row number.
So can I say the top distance is nothing
but the row number i.
Can I say the left is nothing but the column number j?
Why?
Because if I'm talking about this, this 3 is the distance,
isn't it?
And if I talk about the right, what's right?
If I'm standing here, this is the distance.
It's like 6.
The last index is definitely 2n minus 1 minus 1
is the last index.
Like 7 minus 1, 6 is the last index.
And then in order to find the distance, whatever is j.
And the bottom, similarly, the last index
is 2n minus 1 minus 1 minus i.
Why this distance?
If this is the case, take this 6 and then you mine.

Once you've got this, you can actually get all the values.
Once you've got all the values, you subtract n
and you get your answer.
So we know one thing for sure.
The top distance is i.
The left distance is the column j.
And we know the bottom distance or the right distance
is 2 into n minus 1 minus 1, which is minus 2.
So you can be minus of the column number j.
And we know the bottom distance or the down distance
is 2 into n minus 2.
That's the last index minus i.
And once you've got this, you know what you have to print?
The value will be subtracted by n
because previously you also subtracted.
And the min of all of these, the min of top down
and min of left right, the minimum function only
accepts two variables, that's why.
And you can do a n l is it right?
Let's go ahead and see if it is running fine.
That is, it does very easy, isn't it?
So we have completed 22 and all the patterns are finally
completed.
So guys, I hope you enjoyed the entire lecture.
It took everything out of me.
So please consider hitting that like button
and to follow our ritual, do comment to understood.
So that I get an idea that you understood everything.
If you have any doubts, put it into the comment section.
I will be replying back.
Or the community will definitely help you.
If you need to watch our, what are you waiting for?
Hit that subscribe button, follow strivers A to Z DSA course.
Share your learnings using the hashtag strivers A to Z DSA.
And yeah, with this, I'll be wrapping up this video.
Let's meet in the next lecture.
Till then, bye-bye, take care.
Whenever your heart is broken, don't ever forget your golden.

Complete C++ STL in 1 Video | Time Complexity and Notes

https://www.youtube.com/watch?v=RRVYpIET_RU&list=PLgUwDviBIf0oF6QL8m22w1hIDC1vJ_BHz&index=6

Hey, we are welcome back to take you forward. So in this video, we are going to learn about C plus plus STL. What is STL?
STL is basically standard template library. Now in order to write codes in C plus plus, assume you're you're going to use a container or an algorithm. A lot of times what happens is you don't need to predefined the container or write the long,
long codes for that container or for an algorithm. So STL is basically a compilation of algorithms,
containers, iterators, functions in a minimized version so that you don't have to write lengthy lines of code and you can use that STL and you can get access to a container or to an algorithm.
So in this video, I'll be teaching you all the STL that is actually required to get started with DSA.
So please make sure you watch this video till the end to understand or to learn C plus plus STL in depth.
So without wasting any time, let's get started. So let's understand the skeleton of a C plus plus code.
So you can see this is the entire skeleton of a code and the first line is hashing load bits slash STD C plus plus dot H.
Generally, this is nothing but a library and you'd have learned about something like math dot H.
So in order to include all the algorithms in math library, you have to hash include math dot H.
Similarly, in order to include all the libraries in string, you have to give string dot H.
Similarly, if you're wanting to add anything, you have to actually write hash include the library dot H name for sure.
Now, imagine you're writing an algorithm or a program where you require a bunch of libraries.
So you cannot actually waste a lot of time in including all the libraries individually.
So what C plus plus tells us, okay, listen, I have all the libraries inside this STD C plus plus dot H.
They just include this one and all the libraries are automatically included and I don't need to individually add them. Now, this is the into main where you actually write the entire lines of code.
Now, what is this actually for using namespace STD? Now, imagine just for a case imagine I write
scene of A and A is an integer. So this works, but if you just omit this line, this will actually give error.
Now, if you don't add this line, you actually have to write STD double colon scene of A or STD double colon
C out of A. This basically takes an input into A. This basically prints the A into the screen.
So if you don't write using namespace STD, this you have to write every time. So that's a again,
hectic process. So that is the reason what we do is we write using namespace STD and we omit this.
But if you want to know about more in depth, you can definitely read a lot of articles. I'll be leaving a link of an article in the description below. You can check that out. So before moving to
the STL part, let's understand functions. So there are a couple of functions. So one is the void.
So if you write void and probably print and over here you give C out, let's say my name Raj and I
call the print function. So what happens is this print function calls and this will output Raj in

the screen. Now, this is a void function. What does a void function means? It will not return you
anything. Okay, so that is one kind of function and the other kind of function is a return type function.
Assume I write int of sum a int of b and I say can you please return a plus b and over here, I say int s equal to sum. Can you just do a sum of one and b? So what happens is it calls this function passes one into a passes five into b and returns a plus b. Yes, and it returns a plus b. So you get a plus b that is basically one plus five six into the s. And now if you do a C out of s,
what happens is this actually prints six into the screen. You need to understand this.
Now this is a return type function, right? Now this can be a double. This can be anything like you
can use any data type as you wish to. So these are the basic stuffs about the C plus plus skeleton
of record. Okay, so before moving to the next part of the video, I'd love to thank the sponsor of this video, which is coding ninjas. According ninjas is India's leading at tech platform.
If you have a computer programming, machine learning, data analytics, web development, and web development. If you have any course technical field, you can definitely check out the courses at coding ninjas. Why coding ninjas? Because in their courses, they are very well structured so that you can learn the path of it. And also the courses are very well curated because these courses are prepared by people who have been at IITs as well as Stanford and by the people who have been at Amazon Facebook and Google. Now if you
don't believe it, you can check out the Facebook as well as the Google rating of coding ninjas. The best thing about them that I find personally is the doubt resolution time.
Like the average doubt resolution time in the last one year has been 10 minutes. Like if you're raising a doubt, it gets solved within 10 minutes. So that's the amazing, like that's the most amazing thing that they do provide. So if you are looking out to buy any of the courses, you can
check out the link in the description. You can easily get an additional 20% discount on whatever
price that has been going around. So guys, make sure you check out coding ninjas. The link will
be in the description. Now it's time that we move into the actual part that is the C++ STL.
So you need to understand that C++ STL is divided into four parts. The first being algorithms, the second being containers, the third being functions, the fourth being IITs. So as of now, we will be learning about containers. It can be vector, it can be q, set, map, and a lot of other things. And we, during the course of learning containers, I'll be also teaching you what are IIT readers. So these couple of things we'll be learning at first. After that, I'll be talking about different algorithms and different functions that do exist in C++ STL. So before moving on
to containers, you have to actually learn about pairs. Now, what is pairs? Pairs is a part of utility library. Okay. Now imagine I say that I want to store couple of integers. Like I want to store one and I want to store three. So the only way that you can do is you can store it in a pair,
now how is pair defined? You define the pair stuff. And then you see the first stuff that you want to store is of integer data type. The second stuff that you want to store is of integer data type. Then you enclose them into curly braces. So what happens is this actually stores everything
into a variable P. So this variable P is now having one comma three is that is the meaning of this particular line. So that's how you define. Now in place of this int, you can also have something like double string character. The data type can be anything. Okay. Now if P is storing

something like one comma three. Now if I'm accessing P, it's actually a pair. What if I want to access this one? What if I want to access this three? So it's very simple. What do you say is, you just write P dot first P dot first and that will go and access this particular one. And if you write P dot second, that will go and access this particular three. So that is as simple as it can get. So if you're printing P dot first, one gets printed. If you're printing P dot second, three gets printed. I hope this is clear. Now imagine you're wanting to store. So as of now, you know that if you declare an integer data type or a string data type or a character data type, you can always store a single variable like if I want to store i and t a equal to two, that's fine. And I also know if I want to store two variables, I can use something like pair. But what if the question comes up and they see that when I store three variables, four variables,
five variables, can you do it? Yes, we can do it. We can use the nested property of pair. Yes, we can use the nested property of pair. Now imagine I say you that you're going to store one, three, four. So what you'll see is, okay, I have one pair and I know this pair can store two guys. So this is the first guy and this is the second guy in a pair. So what I'll say is the second guy can be of a pair data type. There by the second guy stores two guys in itself. That is how you can do it. If you're wanting to store three variables, got it like pair of integer. And in the second guy, you say that I'm going to store a pair in itself. So basically, one comma and the second guy is a pair. So again, a curly brace and three comma four. So that's
how you can also store three. And eventually if they're asking you to store four variables, you can go nested nested nested nested nested that you can improvise. Okay, so that's how you
actually store more than two variables in a pair. But now if you want to access this one, now if you remember well enough, I told you pair contains two guys, right? And this guy was called first.
If you remember well enough, this guy was called first and this guy was called second, correct?
Now, but this second guy now stores two guys. So can I say this guy is nothing but first, you get second. Then seconds first. So seconds first is what this particular guy is and what is this guy? This is nothing but the second dot second. So this is second dot second as simple as
that. So this is how you can easily access. So if I ask you what is p dot first, it's one. If I ask you what is p dot second dot second, it's actually four. If I ask you p dot second dot first, it's actually three. This will be the output. If I write c out, okay? Now, what if I declare now, as of now, you would have been declaring arrays like integer array, right? Or character array, or something like a string array, but can I declare a pair array? Yes, you can't declare a pair array. The data tag can be anything. Pair can also be data type. Now, this is the index zero. This is the index one. This is the index two now. So you're storing pairs in your array indexes as of now. So Pair can be treated as a data type. It generally lies inside the utility library. Yes, it lies inside the utility library. Okay. Now, if I'm if I'm trying to access array one. So that's basically this dot second. Now of this guy, which is the second, that's five. So if I'm trying to print this, this is going to print five. So this is the entire knowledge of pair.
That is required in order to get started with data structures and algorithms. And the next part we'll be learning about vectors. So the first container that we will be learning is vectors. Please understand every possible function about vectors because these functions will be similar
in all the other containers like queue, list, map, set. So please, please make sure you understand
all the functions in vector in depth. So till now, if I ask you to store five values,
the normal thing you would have done was you would have declared an array of size five. And that will be giving you access to five possible indexes, right? If you remember well enough,

this gives access to five possible indexes. The first one being zero, the second one being one,
two, three, four. Now, but afterwards, if you want to modify it, like if I want that, I want to enter one more element, I cannot modify the size of this array because this array has been declared
of size five. And I cannot modify the size because arrays are constant in size. So this is where something like vector comes in vector is a container, which is dynamic in nature. Like you can always
increase the size of the vector whenever you wish to. So if there is a requirement where you do not
know the size of a particular data structure that will be required, that's when you think of a vector.
And that is the best place to use vector. So vector is a container only with stores, elements in a similar fashion as the array does. In order to declare vector, it's very simple. You just give the vector name, then you give the data type, it can be integer, double, care, string, anything. And then you declare the data type name. Like over here, it's V, it can be large, it can be striver, it can be vex, it can be ABCD, it can be anything.
So right after that, there is a function as pushback. So if I say pushback of one, what it does is
this line basically does is it creates an empty container. Remember this, it creates an empty container.
And the next line, it says pushback of one. So this empty container says, okay, I'm empty. So I'm going to take one into it. So pushback of one will do this. I'm going to take one into it. Now there is another function as in place back. What do you mean by employees back? It's a similar
to pushback. It is similar to pushback. So the moment you do employees back to, it dynamically
increases its size and inserts two at the back. It dynamically increases its size and pushes two
in the back. Now, generally, employees back is faster than pushback. Now you can find the reasoning
why and Cora. I'll be leaving the link in the description. Okay. Now this is how you can define vector. Now can we define vector of a pair data type? You can again define vector of a pair data
type. As I said, so what do you just need to do is you need to change the data type declaration
inside this. So if you just change it, you can always. But over here, there's a trick. As I said, there are a couple of ways in which you can insert elements into vector. One was pushback. The other
one was in place back. So if you're using pushback, you have to insert like one comma two. You have
to give the curly braces in order to enter like in order to insert a particular pair. But if you're using M place back and you write without curly braces, one comma two, M place back automatically
assumes it to be a pair and takes it as an input and inserts into the vector that you have defined.
So that is how M place back is different from pushback in terms of syntax. Now what if I want to
declare a container with a lot of elements already filled. So imagine this is the size. Now this is how you can also declare something of a size. Like this is the size that you can give. So what
happens is a container containing 100 like containing five instances of 100 is already defined. It's a container containing five instances of 100 is already defined where this is the 0th index.

This is the first index. This is the second index. This is the third index and this is the fourth index. So container of size five is already defined with five instances of 100. What if I don't want to declare with 100? You can just declare of this. If you do this, what happens is a container
of size five with five instances of 0 or any garbage value is declared. Now this depends on the compiler. That's how you can easily do it. So similarly if I do something like V1 of 520,
this will declare a container of five instances of 20. Now what if I want to copy this container into some other vector. So you just need to declare another vector V2 and you need to pass on
this particular vector V1. So V2 will be the similar container but a copy of it, like a copy of it, not the same one. It will be another container of five instances of 20. So this is how generally the declaration is. If you know these kind of declarations, it does work.
You don't need to declare any further. You might be thinking, but strive to define the size of the vector to be five. Can we increase it afterwards? Yes. Even after this, you try this line, push back
one. After this line, if you try this, it increases its size and inserts one at the back. Yes, even if you would try this line, after this, it will increase its size and expand to a size of six. So it has allowed its dynamic in nature. You can always increase the size of vector,
even if you are pre-defining the size to be five. Remember this. Now, how do you access elements
in a vector? One of the easiest ways is, imagine your vector is like having 20, 10, 15, 5 and 7. Imagine this is what your vector as of now is, the container as of now is and this is the 0th index as I told you. This is the first index, second index, third index and fourth index.
The best way to access them is you can say V1. So if you say similar to array, if you say V1, it means N. This actually means N. If you say V3, it actually means 5. So you can actually access it in the similar fashion as you do for an array. That's how you can access it. That's one of the ways. You can write directly V0 or you can write V.Add. Generally, people don't use this. So you can just avoid it. You can simply use the stuff that you use in array. What's the other
way? The other way is an I-trader. Yes, the other way is an I-trader. Now, I was telling you that through the lecture, we will be learning about containers and I-traders. So let's understand
I-traders. What is an I-trader? Imagine the vector like, let's see the same vector only 15 and 6 and 7. Imagine this was the vector. Now, if I write like for I-trader, you have to write this. Whatever data type you have taken, vector, that data type, double columns. And if you write I-trader,
and this can be anything, this can be I-t, this can be byt, this can be anything, this is just a name.
But the syntax is the data structure, data type, double colon, I-trader, this. And you write V-dot begin. So I-trader basically means it points to the memory address because these guys would be
stored somewhere in the memory. Like, this 20 would have been somewhere stored in the memory,
right? And that memory can have any address, the address can be 8,567, something, something,
the memory address can be anything. This 10 will be right after that, this 15 will be right after that, right? So all these possible values are actually stored in memory. So what we do is,
if you write V-dot begin, it actually points to that memory, it points directly to the memory, right? Not to the element, it points to the memory, understand that. V-dot begin means,
it's pointing to here, but on the memory, okay? So if you're trying to print V-dot begin,
you're printing the memory address, not the element. And in order to access, like if you have read about C++, in order to access anything that is in the memory, we just write star. Like,
if I write star V-dot begin, understand this. V-dot begin is going to give you the memory,

this portion. And the moment you write star, the element inside this is accessed. Now, let's understand again. If I'm writing V-dot begin, this is actually pointing to the memory, where 20 is, okay? The next time I'm doing, I'm saying, hey, I traitor, if we just move ahead. So this begin does is, this goes here. So now the traitor, instead of begin, is right at the next memory address, because I've shifted the memory, and inside all of these are contagious memory locations. All of these are contagious memory locations. So if you're shifting it, ++, it moves to the next memory. Again, if you do IT++, moves to the next memory. After that, if you're doing star of IT, what will happen? What will happen? As of now, it was 20. So if you do star of IT, this actually prints 10, because it has shifted to 10, and now you're saying star, access the value at that memory. So you get 10. I hope that makes sense, okay? What if, right after this, I do an IT++2. So basically, I'm saying shifted by two positions to this portion, shifted directly by two positions to this portion, which has a 6. And if I try to print this, it will print 6. So this is how you can easily use the I-trader. I-trader is nothing but points to the memory where the element is lying. I hope that makes sense, okay? So we are talking about I-traders. Now you must be thinking, do we have any other I-traders apart from begin? We do have, we have something like end, reverse end, and reverse begin, okay? Imagine I have something like 10, 20, 30, 40 as the vector, okay? So this is what the vector is. So if I am saying vector dot end, like where does this point do? Remember this end will not point to this portion, end will not point to this portion. Instead of that, end will point to somewhere right after 40, the memory location that is after 40. Now if you on this I-trader, do an IT-minus-minus. Then this I-trader will move to 40. Then only this will move to 40. So end points to a memory location that is right after the last element. Please understand, this is very, very important in terms of I-trader. So you understood about end, but not about something like reverse end. These couple of things are never used, but just know it like it's never ever used. What is reverse end? Reverse end basically means I am going to reverse this, I am going to reverse this. So apparently the reverse is 40 at first and 30 than 20 than 10. So now the end is 10. So right after end, so that's this position is where it will be pointing. That's this position is where it will be pointing. Reverse end, end means right after right, and reverse begin will be pointing to this. Reverse begin will be pointing to this. There is a specific thing about this. It moves in the reverse way. If I try to do IT plus plus on this, if I do IT plus plus, IT as of now is pointing here, IT plus plus will move here. Yes. And after this, IT plus plus will move here. So it's a reverse I-trader. You have to think it in the reverse way. Like if we just think this array in the reverse, it's 40 vector rather, 40, 30, 10, 20, 10. And now if I talk about end, this is end. And if I talk about begin, this is begin. And if I say begin plus plus, it moves to 30. So it's that way. Reverse. Everything is in the reverse order. Never used. We need to know. Just know it for the sake of knowing. No one is going to ask you. We have discussed about V of 0, V of add of 0. What is V dot back? As the name recommends, if the vector is having something like this, V dot back means the element which is at 30, is the element which is at 30. That is the meaning of this. Now, if I'm if I'm wanting to print the vector, there are a couple of ways. Imagine I have 10, 20, 30 as the vector. And I want to print it. The simplest way is I know the indexes are 0, 1, 2. So I can directly loop from 0, 1, 2.

Like I can just go across and say I'll look from 0, 1, 2 and print it. That's a very simple way. The other thing is I say I'm going to do it iterator wise because I know this guy is begin. So I read begin and I know the last guy is end right after the last guy is end. So I'm going to run
the iterator. It does not reach is the last guy. I'm going to do an it plus plus. The first time, I get 10. Next I move it. I get 20. Next I move it. I get 30. And I will print every time star of ID. And this is how you can print the entire vector. This is how you can print the entire vector. But but but there is a shortcut. Now you must not be like no one wants to write these long syntaxes because STL means short. Like standard template library, but it means everything in a very simpler fashion. So STL gives you something like auto. So if you write auto,
it automatically assigns it to a vector iterator. You don't have to say that this is a vector iterator. You don't need to define the data type. It automatically defines the data type. Now even if you write integer a equal to 5. So you're defining a to be of integer data type. Even if you write something like auto of a equal to 5,
computer automatically says that this is an integer. So this a will be of integer data type. So if you write auto, the data type is automatically assigned according to the data. According to the
data, the data type is automatically assigned. Like if this would have been something like Raj String and I would have written auto of a, a would have been automatically string. So auto means
auto assignation. So that's that's the beauty of C++. For some time, you don't know the data type. You can just write auto C++ will take care of it and it will automatically assign the data type for you. The other way to print the vector is using the for each loop. So if I use this for each loop, which is IT on V. So it's basically means if the vector is 10, 20 and 30, I'm saying IT.
First time ID is here. Next time ID is here. Next time ID is here. And again, simply print the IT. Auto means on the data type. Please, I treat on the data type. First, I treat on the 10, not in not an iterator, not not an iterator. This means over here in because it is of integer data type.
Automatically, I treat 10 than 20 than 30 and automatically prints it entirely. So that's about how to declare a vector, how to use it reader in a vector, how to print a vector. Now let's understand the deletion in a vector. So imagine I want to delete something. So there is something as
an erase function. And if I had the vector like 10, 20, 12, 23, I say begin plus one. Now in order
to use the erase function, there are a couple of things. Either you give the iterator that click, the location of the address, the location of the address that you want to delete, that this is the address that I want to delete. Please, please delete this address. So I'm saying, okay, begin plus one. What does this mean? Begin plus one means 20. So if you just do erase of this
address. So the vector will be now 10, 12, 13, the vector will be reshuffled. The vector will be reshuffled into 10, 12 and 13. That is how the reshuffling will go on. So that is one way to erase.
So we have understood how to erase single element. But what if I have a vector like 10, 20, 30 and 40
and 50. And I say, striver, I want to delete these couple of elements. I don't want to go single. Do you have something? I say, yes, I have. And that's like erase. And I say, give me the starting
address and give me the end address after the element. Very important. End address after the
element. So starting, if I want to delete 20, can I say that's nothing but begin plus one? I can see because right after begin, that is where 20 is. I want to delete 30. That is begin plus two.

But I said, right after what do you want to delete? So that's begin plus three. That's begin plus three. Three is pointing to here. So I have to delete this portion. But you have to give this end after after the guy that you want to delete after right right after 30. You have to give the address.
So apparently it will start and this is something like this end is not included. The start is included. Got it. So please make sure you give something as which is not included right after one
right after that. Okay. So that's how you can easily delete it. Okay. So for this example, we have
10, 20, 12, 23, 35. What do you mean by plus two? That means 12. What do you mean by plus four?
Plus two plus three plus four. So plus four is here. So apparently this and this will get deleted and you will be left out right and 20 and 35. I hope that makes sense. Now I'm going to learn about insert function. Insert. Again, if you want to insert something, it's very simple.
First of all, if you declare this, this creates 100 like two instances of 100 in a container.
Now if I want to insert a 300 right at the start, right at the start. So I'll say not begin
and please insert 300. So this does this inserts right at the beginning. Okay. 300. Now imagine
you had like 10, 20, 30, 40. Okay. And I want to insert something here on insert of five here.
So do you write? This is the first position. This is the first position. So instead of begin,
you have to write begin on the first position. Can you please insert five? So if you write this, this five will go here and get inserted right at the first position. That's how you do insert.
Now that was for single element. You inserted single element. What if I say that you have 10, 20,
or 30, 40 as the vector? I don't want to insert two fives. Imagine I want to insert two fives.
How do you do that? So you say I want to insert at the first position. The number of elements that
I want to insert and the number that I want to insert. So this will do is this will take five
and five and insert it right after 10. So this makes it 10, 5, 5, 20, 30, 40. So if I say over here, you had 30, 100, 100. So I'm saying begin plus one, which is right after 300. Two positions, like two, two occurrences of 10. Two occurrences of 10 inserted. Understood? Very simple. So that's
how the insert function does work for this. Now what if you have a vector and you want to insert it
into some other vector? Now imagine I say that. Okay, I have a vector like for this example, we have 50, 50. So I had a 50 and 50 vector. Okay. Now this vector is named as copy. So this is
the line that declares that. Okay. Now I already had this vector 30, 10, 100, 100. And now I want to
insert this 50, 50 somewhere. So at that somewhere, I give that address. And I say please enter this
entire vector. Please enter this entire vector. So it will easily take this entire vector
and incur it. If you want to have a portion of this vector, you can give that starting portion and you can give this after end portion and that will also do it. Again, not required. What is required is arrays and this portion. This is hardly required. If you just know arrays and insert about a single element does. Okay. That's how you can easily do about vector. Now what
are the other functions in vector? V dot size will give you. How many elements are there in the vector
as of now? Dot pop back. If this is the vector pops out the last element. Dot swap is very simple.
If this is a vector V1, this is a vector V2. It swaps the vector as the name recommends. V dot clear doesn't matter how big your vector is. Trim sit down to an empty vector. Trim sit down to

an empty vector. Erase is everything. Okay. And V dot empty says does your vector? Like if your
vector also has like a minimum of one element, it says not empty. Not empty. But if the vector has
nothing, it will say true empty. So these are the functions that are generally required in a vector.
So the next container that we will be learning is list. Now list is exactly similar to vector,
but the only stuff in list is it is it gives you front operations as well. Now list is a container
again dynamic in nature. Same way of declaration. You can push back to you can replace back four.
So this is the list that will happen if you push back to and four. And after that, if you say push
front, this front goes over here five, like it directly pushes it into the front. In vector,
you had to use the insert operation. And if you're inserting somewhere that does take a lot of time.
Like insert function in a vector is very costly. Like we will we will read about time complexities
in further data algorithmic lectures, but as of now just remember an insert in a vector is costly.
And in list, since internal operation is a doubly linked list. Like a doubly linked list is
maintained for a list and for a vector as single linked list is maintained. So thereby something
like push front is very, very cheap in terms of complexity time complexity wise when you compare it
to a vector. Okay. And there is a place front as well. All of the functions begin R and reverse
and size clear MP. All of the functions are similar to vector. So I'll not be explaining that.
Okay. That is about list. Now the next container that we will be talking about is DQ. Again,
similar to list and vector. You just declare it push back, push front, pop back, pop front,
back front and all of the functions are similar. I'm not going to explain this as well. It is exactly
similar to list and vector. So the next container that we will be learning is stack. Now stack is
something as leaf. Oh, leaf. Oh means last in rim of this last in first out. The guy who went
in last is the guy who will come out at first. So generally, you can just imagine a stack to be
a data structure like this. And this is how you declare a stack stack the data type and the variable
name. Okay. So I've declared the variable name like this. I'm saying push one. So I push one.
I'm saying push two. I push two. I say push three. I push three. I'm again saying push three. I
push three. I say in place is similar to push. So I say push five. Now these are the push operations.
Right after that, if someone says stack. So as I said, who's the last guy who went in the last guy
that went in, you know, is nothing but five. So this will print five is this will print five.
Now realize this over here indexing access is not allowed. You cannot say this is index zero. This
is index one. Something like this, you cannot say in stack, there are only three functions. One is
push. One is pop. The other one is stop. All other, they're like size clear are there. But these are
generic three functions that you have to deal with. So if I'm saying top, it gives you five,
but the five is still in the stack. The five is still in the stack. Now the moment I say pop,
it deletes this from the stack. Now five is not in the stack. Now if I'm saying top,
right before five was there three said print three. If I'm at this woman saying stack dot size,
there are four elements. So I print four. When saying stack is the stack empty. The answer is
false. It has four elements. Now if I'm saying swap it to some of the stack, I declare another stack
one. And I say, can you please swap it? So swap is very understandable. Both, both the guys will
swap. So I hope pushes clear, pop is clear. Pop means delete. Pop means just tell me what is at

top. You don't need to delete. Just tell me what is at the top. So that is how the stack STL works
like. Now talking about complexity wise in stack, all the operations are we go off one operations.
Everything happens in constant time. So let's learn about the next container now. Now the next
container that we will be learning is a queue data structure. Now a queue data structure is similar to stack, but over here it is FIFO. Okay. FIFO means first in the guy who gets in first comes out first stack was last in. Okay. Now you can just think it in this way. Like if you
if you're forgetting names, you can say like you go into a platform and if you're buying a ticket, what do you generally do? The person comes in. The first person who comes in stands and the next
person who comes in stands. So the first guy who gets the ticket, this guy, then this guy gets the ticket. And if someone is coming, it's a queue that happens. So that is where the concept comes
in first in first out. So if I'm saying push one, I push one, push two, I push two, push four.
So I pushed one, two and four. And the next line, I'm saying queue dot back plus equal to, but back will mean four. It does not means this guy. Back will mean four only. So over here, I'm saying add plus five. This makes it nine. I'm not saying queue dot back. So if I'm saying queue
dot back, it prints the last guy nine. Okay. If I'm saying queue dot front, prints one, just prints, just not deletes. If I'm saying queue dot pop, deletes, deletes, deletes the front guy, first in guy, queue dot front is now two. So I prints two. It is similar. And now all the operations are much more similar to stack size and all these things. That's how queue works. Again, all the operations are happening in constant time. Now the next thing that we will learn is priority queue.
Now as the name recommends priority, the guy who has the largest value stays at the top. It's similar to queue, but there is some, something that happens inside which you learn probably
after a couple of years when you are appearing for interviews. But as of now, just understand the logic.
Okay. So remember, if you're declaring priority queue like this, the maximum element stays at the top
or the largest element. If you're using character, the largest character, if you're using integer, the largest integer, if we're using the string, the lexically largest string will stay at the top.
Okay. So I'm saying queue dot push 5. So you push in 5. I say queue dot push 2. So you push in 2.
I say queue dot push 8. So you push in 8. Now I say queue dot push 10. So you push in 10. Okay.
Now the moment I say pq dot top out of all these elements, which one is the largest? 10, said prints, 10. And that is the guy who will be at the top. Now if you're trying to insert something like three, three, the three will go right here. The three will go right here.
And this is not a linear data structure inside of it. A three data structure is maintained, which you learn in the later half. Now understand one thing. The data is not stored in a linear fashion
in like inside. At inside a three is maintained. Okay. Which probably you learn someday. Okay.
So as of now, if I say pq dot pop, the top most element is popped and is popped. I say again, a pq dot top is the top most ADS. So this is how the priority queue works again push
top and pop main functions. The other ones are size and md. Size and md is very simple.
And so app is also very simple. So these are the functions. What if I want a priority queue?
It stores the minimum element at the top. Then this is how the syntax is priority queue, data type,
vector and this greater ink. You have to write. And if you give this data type,

and now if you push five, five is there, if you push two, two is there, and if you push eight, eight is there, if you push 10, 10 is there. But this time if you try to access pq dot top, two will come out. So that's how you maintain a very simple minimum priority queue. And generally, it's known as mean heap. And this is known as max heap. Remember this term that you learn in DS algo as you move forward. What is the time complexity of push? Push happens in logarithmic of n. Top happens in we go of one and the pop which is the deletion again happens in logarithmic of n. So this is how it happens. If you don't know what is logarithmic of n, no issues. Just keep this in your mind as you move forward. You will understand in DS algo. Now the next container is very, very interesting. And that is nothing but the set container. Now what is set? Just remember one thing. It stores everything in the sorted order and stores unique. Just remember these couple of points. And you know, what is the set? Everything in the sorted and just unique. Just two points and you're done. Let's understand. So imagine this is the container. And I say insert one. So you insert one. Imagine you say emplaced two. So you insert two. Imagine you say insert two unique. So will it store two again?

No, it does not. No, it does not. It does not stores two. If I say insert four, will it? Yes. If I say insert three, it will. But it will insert it here again. A very important thing. Sort it. Stores in a sorted order. At first, it will have one. Then it will have two. Then it will have three. Then it will have four. Everything in the sorted fashion. So it stores everything in the sorted fashion. So this is how the set will be storing. Again, a container. Is it a linear container?

No, a tree is maintained. So I'm just explaining you this via this bucket. But inside of this, there is an entire tree which is maintained again, which you learn as you move forward. So insert emplaced works in a similar fashion, sorted and unique. Now there are functions. If I say set.find3, and this is the set. So it will return an iterator. Remember this. Returns an iterator, which points to this tree, which points to this tree. It returns an iterator, which points to this tree. So basically, this is an iterator. Remember, iterator points to the address. Perfect. Now if I say set.find6 is 6 here. No. If an element is not in the set, please hear me out properly. If an element is not here in the set, it will always return set.end. That means an iterator, which points to right after the end. Imagine the set is having 1, 2, 4, 5. And you did set.find3. And you don't have a tree. So you will have an iterator end. This is the iterator, which points after 5. That's where.find will turn the iterator to be. It points afterwards. And after this, there is set.irase. Very simple. Erases this guy 5. Erases this guy 5 out of it. Like if this is the set, if this is the set, it will delete 5. It will simply delete 5. You don't have to think anything. Deletes 5 and maintains the sorted order.

Deletes 5 and maintains the sorted order. Now, as I said, set is nothing but unique and sorted.

So if you're trying to count, if it exists, if it exists, it will only have one occurrence, because it does contain unique. And if it does not exist, it will have 0. So if one is there in the set, it will give you 1. Like 1 occurrence. If it is not, it will give you 0. That's simple as that. You can either give the element to be erased or you can give that. This is the address or the iterator. Please go and erase this iterator as simple as that. Now, in vector, we did learn about erase start comma and similar thing also works over here. If you want to erase everything, yes, if you want to erase everything between 2 and 4, imagine you had something like this. So 2 is here, 4 is here. If you get the first guy, if you get the second guy.

So if you do find, you'll get the 2 side iterator. If you do find a 4, you'll get the 4 side

iterator. So it deletes 2 and 3. Remember this, it deletes 2 and 3, not 4. 4 is this bracket. And this is this, please remember this. Now, in set, there are other functions like size, empty, swap, everything is similar to vector. There is begin, all of these are similar to vector. So I will not be explaining them. The most important are find, count and insert. These are the most important
ones as well as erase. Now, they have something as a lower bound and an upper bound. So I will be
linking a video in the description, which explains lower bound in depth and upper bound in depth.
So please go back and watch lower bound and upper bound. Once you have seen that, you will actually understand how does this lower bound and upper bound work in a set. It's the exact same that will be taught in the video, which is in the description. So please make sure you watch it. Now, in set, everything happens in logarithmic time complexity. If you're inserting,
it takes logarithmic. If you're erasing, it takes logarithmic. Everything happens in a logarithmic time complexity. Again, if you don't know, login, no issues, please remember this in your head.
Now, we did learn about set and I said sorted and unique. That means it will just contain one occurrence of two. You can insert thousands of occurrences of two, but it will just store one occurrence of two. But there is something as multi set. If you define multi set, it only obeys sorted and it will not obey unique. It will store multiple occurrence. Like if you try to insert one, one, one, one, stores all the occurrence. And we try to erase one, but all the occurrences
are erased this time. But if you do an erase one, it erases every one. And this time count will count
you the number of ones in the multi set. But if you want to delete, imagine your multi set is containing three ones. And I just want to delete one occurrence of one or two occurrence of one
or three occurrence of one. So, what I can do is I can just find out the first occurrence of one. So, I'll just do multi set dot find because I know find points to the iterator.
And I'll say erase that iterator. Instead of saying erase element because if I say understand, if I say erase element, it erases all the elements. But if I say erase address, it only erases that portion. It only erases that portion. And I don't know, it is like two, two ones. So, I say find one and go till two. Go till two. So, it will erase both of them, both of them. So, either erase element, erase address or erase starting address and right after the end, that's it, die traders. And rest all functions are same as set, stores everything in the sorted, but not unique. Stores multiple occurrence like one, one, one, two, three, three, four, it can store multiple occurrences as well. So, that is the definition of multi set. Now, we did learn about set. We did learn about multi set. Now, there is something as unordered set.
Everything is similar to set. The only thing that is omitted like it stores unique. The only thing omitted is it does not stores in the sorted order. We don't know how it will store.
It has randomized order. It has randomized order. Like if I put in one, after that I put in five, after that I put in two, after that I put in three, after that I put in six, it can have this order, it can have any order in the world, but it will just have unique elements. Like if I try to insert one,
again, it will say I have one. I have one. And in most of the cases, the time complexity is b go of one. Everything like all the operations are same. Insert, erase, all the operations are same, but only the lower bound and upper bound function does not work. All the operations work,
but the lower bound and the upper bound functions do not work. Remember this, all operations are
similar to set and they do not store everything in sorted order. So, all the operations are

generally in a b go of one constant time, but in the worst case, which happens once in a millennium.
Like if the data is pro like possibly given in such a way that they want you to explore the worst case, which does not happen, then the unordered set goes for a b go of a linear time. It goes for the worst case. Again, it does not happen every day, happens once in a blue moon,
the time complexity goes till b go of and just write them down in notes. You will understand these
things when you move across or when you grow in experience. That's how the unordered set works.
So, the next container that we will be learning is a map container. Now, you can think this as something like just take a task where I say in your college, there will be like there might be multiple people with Raj name, but how do you distinguish themself? By rule numbers,
one Raj might have a roll number of 23. The other Raj might have a roll number of 25. The other
Raj might have of 28. So, you know this one guy is of roll number 23. The other guy is of roll number
25. The other guy is of roll number 28. So, generally this is in in your class, this is store like this roll number one, roll number two, roll number three and so on. If there are 50 people, you store
it like this. So, map you can think this as a data structure or a container, but say the roll number
is my key and over here the value can be the name. So, this is what the data structure means.
You store unique keys because you can't have 23 roll number twice, you can have it just once.
So, the keys are unique, the keys are unique, but the values can be like, over here they can be
a Raj, over here they can be a Raj. So, there is one guy with key three who is Raj, there is one guy
with key 50 who is Raj, they can be duplicate values, but it has to be a unique key. So, you can
think map as a container which stores everything in respect of key and values and very important
thing. This key can be of any data structure, it can be integer sorry any data time,
it can be integer, it can be double, it can be pair, it can be anything. Similarly, this value can be anything. So, how do you define map? This is key, this is value, key is integer, value is integer, over here key is an integer and the saying value is two integers, over here they are saying
key is two integers and value is one integer. So, you can define it as you wish to that is on you,
this is how you define. Now, if I am defining the map to be this for an example, assume you are
defining the map to be this. Now, I say one equal to two, it means on the key one, can you please
store two? So, this is what it stores internally in the map, internally in the map, it stores one comma two. Next, I say I don't want to store it like this, and place three one. So, it stores three is the key and the value to three is one, it has the similar thing stores it into the map. Again, I say insert, you can also use insert two comma four goes and stores two comma four and this is how you can store all these three variables. So, in this way, you can actually store for this particular declaration and remember one thing, map stores unique keys very, very important,
map stores unique keys in sorted order, something similar to set data structure, map stores unique

keys in sorted order, something similar to set data structure. Now, this is the declaration, the second declaration for this declaration, sorry, this declaration, this is the key. So, you have declared the key like this and the value is a single integer. So, this is for this, it will be storing like key is two comma three. So, this is stored and the value corresponding value is 10 perfect. So, this can be stored like this, that's how it generally stores, I hope that makes a lot of sense. I told you that everything is stored in a sorted order. So, this will at first store one comma two, then it will store two comma four, then it will store three comma one, this three lines will be storing like this. So, again, if you want to explore or if you want to iterate on the map, one of the ways is you start from begin iterator and you go on to end iterator, similar in the vector, all you do is you say IT, you run a four each loop, first time ID is here, so it stores in a pair, next time ID is here, stores in a pair, next time ID is here, it stores in a pair, if it is storing in a pair, this is ID dot first and this is ID dot second. So, first time prints one two, next time goes your prints two four, next time goes your prints three one. So, if you try to do this see out, this is how you can actually traverse in a map and everything is stored in a sorted order of key, in a sorted order of key, not value, sorted order of key, remember this sorted key is how it stores, no duplicates, all uniques, okay. Now, if I want to access map of one, if I want to access map of one, it says a value two, it says a value two, because at one you are storing two, but if it tries to access five, will it find five, there is no five, so what happens it, it says null, but if you want to print it, it actually goes on prints zero or null, okay, because it does not exist. So, if something does not exist, it gives you zero, okay. So, that is how you can easily access for a key. Now, if you want to know the i-trater, imagine you want to know where the key to lies, the address of it. So, again the find function will come over here and say map dot find three, so this is where you get it, the i-trater. And in order to access this, you give a star. So, this access is this, and if you want the value dot second, got it, it gives you the i-trater to the this three comma one, okay, this IT is this, so if you give a star, it's the element, and if you give a second, it is the one, that's how you can easily access this as well. Now, over here, if you try to do dot find five, and five is not there, it points to nothing but dot n, n means after the map, after the map, okay, it makes sense. And again, the low bound and upper bound functions, if you have seen the video in the description, you can understand how low bound and upper bound works, okay, and all the other functions like array, swap, size, empty are safe, so I'm not going to explain it again. Next thing is, multi-map, similar to map, only thing is you can store duplicate keys, you can store duplicate keys, something similar to set and multi-set, as I told you, right, duplicate keys, but everything is a sorted order. This time you can store like one comma two, and then you can come across, and again store like one comma three, right, so you can store duplicate keys over here, unordered map, again unordered map is similar, only this portion will go across, it will not store in sorted, it will be randomized, it will be randomized, but it will it will not have duplicate as well, it will just have unique keys, unordered map will have unique keys, but it will not be sorted, and the difference is like the map works in logarithmic of time, and unordered map in the, in almost all cases works in constant time, in the worst case it goes for be go of n, again this worst case happens once in a blue moon not always, in almost all the cases be go of one is what appears, so as of now I can see that I have completed containers, and I treat this, now this is not required, not required, no need to learn it, just omit this,

now I will be telling you all the important algorithms, like which are mandatory, like you should know,
and all the other algorithms which I will not teach, you will eventually learn while you could, but that are not like those are not important, so as of now you can just leave it, eventually with time
if it's required, learn at that moment, no need to learn it now, so let's move across to learn some
algorithms, now if I say you that, hey listen, I give you an array of size like 1532, okay of size 4,
and I want you to sort it, so you will be like let's apply bubble short, moisture, selection sort, so and so, but in C plus plus S TL, if you just write the line, sort a comma, it's a four size,
so this actually means the first position, the first position of the first iterator,
the starting iterator, and this means the last iterator, A plus 4 actually means this portion, then portion, the last iterator, again similar to something like start, start is included,
and end is not included, so you write the starting iterator which is A, which actually points to this, and A plus 4 which actually points to this, so all the elements are sorted, right after this line, it will have 1, 2, 3, 5, so you don't have to actually use moisture, bubble short,
selection sort, it sorts that into one line, and if you're using vector, the starting is begin,
this is the ending, the starting iterator, and the ending iterator, so in this way you can sort any container, not map, all like all, not map, I'm talking about vectors and arrays over here,
okay, now what if I just wanted like, I had something like 1, 3, 2, 5, and rather let's keep it like 5, 2, okay, and I wanted this this portion to be sorted, so either A plus 2, because this is A plus 2 for sure, and I know A plus 4, right after this is A plus 4, so only this portion will be sorted, so this is how only this portion will be sorted, perfect, what if I want to sort them in descending order, imagine you have like 1, 3, 5, and 2, I want to sort them in descending order,
so it's very simple, give the starting iterator, give the ending iterator, the portion that you want to sort, and just write greater end, yes, just write greater end, and this is nothing but a comparator, an inbuilt comparator, which automatically sorts it like I'll teach you comparator, automatically will sort it in the descending order like this, in a descending order,
you just need to write greater end, and it automatically sorts it into a descending order,
now what if I won't sort it in some other fashion, because as of now we know how to sort it in increasing, we know how to sort it in decreasing, but what if I want to sort it in my way,
because going across, you will see that this my way has been used a lot, for an example,
we have declared a pair array, and these are the pairs 1, 2, 2, 1, 4, 1, okay, and I want you to sort
according to second element, like I want you to sort it in order of increasing second element, please understand, I want you to sort it according to increasing second element, okay, and if the
second element is same, then sort according to the first element, but in decreasing, but in decreasing,
what do I mean by that, so as of now, can I say the second element is 2, 1 and 1, so this 2, 1, and 4, 1 are the guys who should appear right at the first, okay, because the second elements
are 1 and 1, and after that I can say after that 1, 2 will appear because of this portion,
so can I say that I've sorted according to the second element, but I still have a problem,
now these couple of guys are having same second element, now if they have the same second element,
I want you to sort it according to the first element, what in descending, which means I want you to
have 4, 1 at first, and then 2, 1, that means among them, among them, I want you to sort it according
to descending, so first 4, then 2, and then you can write, so first sort it according to the second,

and if there is a group, which is having the same second, then among them sorted according to the
first, so this is my way, this is my way, it's something like a combination of increasing as well as decreasing, so this is where, generally you write the first iterator, the last iterator,
and a comb, and a comb, remember this, and a comb, now this is nothing but a self-written comparator, a self-written comparator, and this comparator is nothing but a boolean function, is nothing but a boolean function, so I've written this, but I'll just teach you how to write it,
it's very simple, you write boolean comparator, and this is the function, this has to return or to end of false, this has to return or to end of false, now go back and see, what is the data type that you just had, and the data type, if you see was pair of ended, just copy paste,
and have couple of guys, pair one, and pair two, that's it, have couple of guys, this is the first thing that I'll do, have couple of guys, now please understand this, that this is pair one, and this is pair two, so forget about the array, forget about the array, and now think of
this two instances, where you have two pairs, where you have two pairs, this is p1, this is p2, while writing comparator, just focus, what have you been set, sorted according to the second element,
so you say okay, okay, if my p1 dot second is already smaller than second, that means it's true,
we are assuming, we are assuming that the pair one lies before p2, lies before p2, that is what
the assumption is, the pair one lies before p2, and that is okay, if the second guy is lesser than
this second guy, I'm okay, this should actually, so they are in the correct order, this comparator
says are two guys in the correct order or not, and I'm saying they are in the correct order, they are in the correct order, because this guy is smaller than this guy, and that is what I was said, so if they are, they are in the correct order, but I know one thing, if it's the opposite,
if this is the case, I will say they are not in the correct order, because if I have two guys, okay, I imagine this is five, and this is four, and I'm saying p1 occurs before p2, this is wrong, this is wrong, if this second is actually greater than this, I know this is wrong, so I say they are not in the correct order, so what happens is comparator, internally says p1 and p2 are not in the
correct order, and you please swap them, so apparently four comes before five, this is what happens,
so they do internal comparisons, and this swap, so I told them false, they are not, so they will swap internally, but do we have any of the conditions, so we have, what if they are seen, because that's the only condition that is left, that is the only condition, because if these two conditions do not happen, I know we will come to a point that they are same, I don't need to
write a if, because that's the only condition that is left, they are same, if they are seen,
we'll again try to evaluate, I know if they are same, this is p1, and this is p2, and this time it is descending, so this guy is greater than this guy, it's okay, otherwise it's not, so can I say if p1 dot first is actually greater than p2 dot first, it's okay, because this is what I was looking for, else I say it's not, if they equal it's fine, even if I swap or do not swap, it's okay,
so can I say, if it's great, it's okay, that's what I'm looking for, whatever it's not,
that's false, please swap it, if it's not, I need in descending, can you please just swap it,
so you just analyze everything, whenever you try to write a comparator, always analyze everything
in terms of two pairs, don't think in terms of arrays, just pick up, yes, I repeat, you just need to do one thing, just pick up one pair, just take another pair, and try to analyze p1 and p2,
that is your job, so we have learned about comparators, so anytime there is my way, my way, sorting, you can write the comparators, just need to write this, and you should be done,

just focus on the data type, and try to just have evaluate two data types, and write it,
nothing different, okay, so this is again, one more STL, which is very important, which is built in
popcorn, so if number seven, what is the binary of seven, it's one, one, one, that's the binary of seven,
so if I say built in popcorn, it says, okay, this has three bits as one,
generally a number seven means zero, zero, zero, zero, zero, zero, like these are the 32 bits inside the computer, in 32 bits it's zero, zero, zero, zero, zero, zero, and one, one, one,
so built in popcorn says, how many ones are there, or how many said bits are there,
so it will return three said bits, if this number would have been six, which is nothing but one, one, zero, six is nothing but one, one, zero, so built in popcorn would have returned to the number of said bits, okay, if the number is long, long, then built in popcorn becomes built in popcorn LL, built in popcorn LL, if the number is long, long, integer will not surprise there, perfect, now the last thing is, not the last second last thing is next permutation,
now if I write a string as one, two, three, and I want to have all the permutations of it,
all the permutations, so what I can say is, okay, listen, I'm going to have the string,
and I know the next permutation is one, three, two, I know the next permutation, like if I talk about
dictionary order, two, one, three, the next permutation is two, three, one, the next is three, one, two, the next is three, two, one, can I say, these are the dictionary, like these are the six permutations that you can have, three factorial is six, so if you want to print them, what you do is,
you first print the first string, let me say, can you have the next permutation, please,
can you have the next permutation, it takes you, the string which was this,
becomes one, 32, and I printed, again, the string now becomes, two, one, three, and you print it,
again the string becomes two, three, one, you print it, again the string becomes three, one, two,
and you print it again the string becomes three, two, one, and you print it, right after this,
it goes to null, it says, no more permutations, it returns a false, if there are no more permutation,
It returns a false, and it returns a false, a while loop breaks, and this is how you can print all permutations of a string.
Here's a catch.
What if the string was 231?
Then it would have started from 231, and the next permutation of 231 is 312, and the next is 321.
So it would have just printed like first tense, then this, then this, right after this, no permutations.
So it's very important that if you want to print all the permutations, you start from the sorted guy.
Like you just start from the sorted guy, and you can easily sort it.
You know the STL now in this fashion.
You start from the sorted guy, and that's how you can easily print it.
Now the last one is max element.
Imagine you have an array like 1, n, 5, 6, and you want the maximum element.
If you give max element, start iterator, and iterator, it gives you the address.
And if you give the star, it gives you the element.
Similarly, main element is also there, right?
So these are the algorithms that are generally used in DS algo in your day to day life.
All the other algorithms are there, but they are not widely used, and you will not be requiring them.
So whatever STL I have taught in this video is more than enough to get started with C++.

That's in case you have understood everything.
It's an honest request that get into the comment section, and just write one line comment because that is the only thing that keeps you motivated to make these kind of content.
And if you are probably the first time on this channel, please do consider subscribing because I have a lot of content regarding freeze, graphs, dynamic programming, DSA, that might help you in your longer run.
So please make sure you subscribe to this channel as well, and you can like this video.
And yeah, please do comment because that keeps me going.
With this, I'll be wrapping up this video.
Let's meet in some other video till then.
Whenever your heart is broken, don't ever forget your golden.

Basic Maths for DSA | Euclidean Algorithm | Strivers A2Z DSA Course

https://www.youtube.com/watch?v=1xNbjMdbjug&list=PLgUwDviBIf0oF6QL8m22w1hIDC1vJ_BHz&index=7

So this video is going to be another video from the strivers A to Z DSA course and this is India's most in depth DS Algo course.
Why do I see that?
Because this course has 455 modules.
I can guarantee you that you can take any paid batches, any free courses.
None of those courses will have 455 modules.
This is an extremely in depth DS Algo course that can teach you everything in breadth about DS Algo.
And in the previous videos we have covered step 1.1 and 1.2.
And regarding step 1.3, I've added a video on C++ STL on the playlist.
You can go and watch it.
Regarding Java collection, I'll be adding a video in the future, not now.
If you need a video now, you can go to YouTube and you'll find a lot of other resources from where you can study.
So in this video, we will be discussing about basic maths.
Why am I teaching just the basic maths as of now?
They are starting off as of now.
Your brain will not be that matured.
So if I teach you the advanced level concepts, you might understand, but it won't be that convenient for you.
That is why my teaching way is very different.
What I do is I usually start off with the basic stuffs.
I give you a lot of time to absorb it and then we move on to the advanced part.
This is why all the basic stuffs are initially there.
And then we move on to the DS Algo and in the step 8, we have a section as advanced mathematics.
We'll be covering everything that is related to advanced mathematics that might be asked in interviews.
But as of now, we will be learning basic maths.
Now, these are the problems which we will be solving.
But before that, let's learn some basic maths concepts.
So before solving all the problems that are listed under basic maths, we'll be starting off with the basic maths concept.
We'll be teaching you the concepts initially and then we can solve all the problems listed under the section.
The first concept that I'll be teaching you is the digit concept.
Remember one thing, this is a very, very important concept because if you know how to play around digits, then you'll be able to solve most of the problems in basic maths.
So let's understand the digit concept.
Imagine, I give you a number like 7, 7, 8, 9, this is the number that I am giving you.
Now I ask you to perform extraction of digits, I ask you to perform extraction of digits.
So let's learn the extraction of digits and after that you will see how we can implement the extraction of digits in order to solve most of the problems, okay.
So when I say extraction of digits, what does it mean?
It means I read 9, I read 8, I read 7, I read 7, I read all the digits individual, okay.
So what is this digit, 9?
Can I see this?
If I do a module 10, I'll actually get 9, you might ask, why?
If I ask you, the numbers that are divisible by 10, what are they?

10, 20, 30, 40, 50, 60, 70, so on, 100 and so on, do you see a pattern?
All the numbers that are divisible by 10 are actually ending with 0, isn't it, right?
So can I see, if I'm doing a module 10, what is the meaning of module operator in B sequence?
The module operator says, I will divide the number by 10 and whatever is the remainder, that is what I'll give you.
So if I see, if I'm dividing this number by 10, what is the nearest number?
Obviously that will be 7, 7, 8, 0, if I divide it by 10, this is what the nearest number will be and can I see that the remainder then will be 9, because if I divide it by 10, then this is where the division, like this is the number which will get divided by 10, and after that will be left out with 9, that is why when you do a module of 10, you always get the last digit, so you get the last digit at as 9, so this is how you get the 9 digit, and if I ask you, can I get the next digit 8, how will you get the next digit 8, very simple, you say, okay, this number, let's divide it by 10, so if I divide 7, 7, 8, 9 by 10, can I see, I'll get 7, 7, 8, 0, 9, can I see this, if I divide the number 7, 7, 8, 8, 9 by 10, I'll get 7, 7, 8, 0, 9, and if I take an integer round of, if I take an integer portion of it, the integer portion is 7, 7, 8, so what you will do is, you don't have to go to the next step, you will say divide by 10, you do a division by 10, you'll actually get 7, 7, 8, 0, 9, but you just take the integer part, that is why you get 7, 7, 8, so once you have 7, 7, 8 with you, if I need the last digit, which is 8, how do you extract it, again the same way, you say, can I divide, what do you lowerize with 10, if I, to a mod of 10, I'll actually get 8, why, because the nearest number will be 7, 7, 0, which is divisible by 10, which will still leave a remainder of 8, so I get the digit 8 as well, now can I see if I require the next digit 7, can I again do a division by 10, if I do a division by 10, can I say I'll get 77, why, because by right, 7, 7, 8, by 10, I'll get 77.8 and the integer portion is 77, I get 77, again if I have to extract the last digit, can I say I'll do a modulo rise, a modulo of 10, I'll get 7, I will, 10, 7, again extracted, 7, now if I need the next extraction, I again divided by 7, so divided by 10 and I'll get 7 because 77, because 77 divided by 10 is 7.7 and the integer part is 7, so I get 7, if I do a modulo rise of 10, I'll actually again get 7, where will you get again 7, very obvious because the nearest number that is divisible by 10 to 7 is 0, thereby you get a remainder of 7, so you get 7, after that, if you again try to divide it by 10, this time you'll end up getting 0, because if I take 7 divided by 10, it'll be 0.7 as the integer part is 0, so can I say if I get an integer part is 0, can I say I've extracted all the digits, I have and if you see the extraction has been done in the reverse order and all the digits have been extracted as simple as that, so this is how you can easily extract all the digits, so if I try to write the pseudo code, how will the pseudo code look like, can I say if I have the n, I can take it from the user, I can take the n from the user and imagine I'm asking you to print all the digits, extract all the digits like 9, 8, 7, 7 and you can print it, so how will you do it, it's very simple, I will be like okay, while I know what is the last step, last step is the extraction goes on from n when it is 7, 7, 8, 9 to n till 0, so I'll be like I'll go on till n is greater than 0, which means till n doesn't become 0, right and can I say the extraction is very simple, the first time n was 7, 7, 8, 9, if I had to do an extraction, it's very simple, can I say the last digit is nothing but n module or i10, if I do an n module or i10, I'll get the last digit 9 and in order to get the next digit, what I do is I say n is n by 10 and this is how I can do it, so what will happen, let's do a dry run, at first imagine I give the user gives n as 7, 7, 8, 9, so this says 7, 7, 8, 9 greater than 0, which is true, so the last digit happens to be 7, 7, 8, 9, module or i10 and the last digit is 9, if you want to print this last digit, you can definitely put a print operation, in C++ it is Cout, in Java it is system.out.printl, so you can go ahead and print the 9, once you've done this, can I say if you do 7, 7, 8, 9 by 10, then the n will reduce itself, what will be the value of n now, can I say

the value of n will be nothing but 7, 7, 8, this is how the first iteration, first iteration will happen and then it will reach here, then again goes here, when it goes here, it will be a new iteration and can I say this time the iteration will be 7, 7, 8 because n has changed itself to 7, 7, 8 and 7, 7, 8 greater than 0, what I will do is I will quickly raise this because it is the next iteration, so let's quickly raise this and the next iteration what will happen, it will see 7, 7, 8, module or i10, the last digit this time will be 8, again you can print that last digit and this time it will be 7, 7, 8 by 10, hence n will become 77, again the iteration will go and it will be 77 greater than 0 and this way all these steps will be performed at the end of the day, the value of n, yes the value of the n will be 0, ends the while loop will be false and I can say that the execution has been completed and you have successfully extracted all the digits in the reverse fashion, very important in the reverse fashion, got it, so this is what is the concept of extraction of digits and this is going to help you solve a lot of other problems as well, so now let's look at the first problem, it states count digits, let's understand the problem, given the number n, find out and return the digits present in a number, very simple, it says 156 is the number and the number of digits is 3, imagine the n is given as 7, it has just one digit, so it will be given an n and it will tell me the number of digits, so if I go back to my iPad, can I say, if I give you the number 7, 7, 8, 9, this has 4 digits, can you solve this problem using the extraction of digits, can you, it will like this is super easy, why, because you know the extraction of digits, you know 1 digit, 2 digit, 3 digit, 4 digit, the digits are extracted 4 times, so can I say, I can keep something like a counter variable over here and you know the number of times
the extraction happens, that is the number of times the digit will be, so can I say I can put a counter equal to counter plus 1 on the logic of extraction of digits, if I do this, can I say I will be able to count the number of digits and eventually if I print the count over here, can I say that I will always have the count of digits of any given and I can.
Usually in coding rounds or in your interviews, you just have to code the function, the function is an int function that means you have to return the count of digits and they will be giving you the input, so you are given the variable, you just have to return, you have to just write the code inside the function, int main and everything will be written on the back end, I have already discussed about this in the pattern video in case you haven't watched it, please go back and watch it, so this was the code that we discussed on the iPad, right, so the count
stores the count of digits, I will just return the count and then I will go ahead and run the code and I will see that it is running absolutely fine and then I will go ahead and submit this, so this is how you can easily solve this particular, now remember one thing, this last digit does
not have any significance, so you can remove it, so that was for extraction of digits but this is kind of reducing the numbers, so the number of times it is divisible by 10 is the number of times
the digits are, now since I have removed the module operation, we observe something, can I say the number of times it is getting divisible by 10, the number of times it is getting divisible by 10 is the count of the digits, it is and this is where something like logarithmic, log base 107789, if you do this in your calculator, it will actually get something like 3.89 something,
so this is the value that you will get, if you do a log base 10 of the number and then if you can add a one to it, this will be 4.89 and if you take an integer of it, that will be 4,
so this is another way to find count of digits, what you do is very simple, you say count is equal to
log 10 the number plus one and you are saying take the integer or you can just auto-cast it like type

cast it to integer, this is how, whatever you get, this is converted to an integer, if you are getting 4.89, it will be truncated to poor, and now let's run this and see if it is running fine, okay, so it says out of scope, log 10 was not declared, so if you find such errors, what you can
do is you can go to hash include, that's basically because in the backend, they might not have added
all the directories, they can go ahead and add all the directories and that will start working fine,
once you have done this, you can go ahead and compile and you are seeing this, you are seeing this
that this is also running fine, so this is one of the other ways to count digits as well, but the primary concept is extraction of digits and that is what you should focus on, now if I discuss the time complexity over here, what will be the time complexity, the time complexity will be nothing but log base 10 n, this is the bigot of time complexity, while log base 10 n, the reason was
very simple, you saw this is getting divisible by 10, how many times is the loop running, the number
of times it is getting divisible by 10, this is why you will see, the time complexity is near about log base 10 n, this was 3.89, it near about 4, the number of times this loop did run was 4,
yeah, you can avoid these operations, these are single operations because imagine the number being
very large, these will be considered as unit operations, that is why the time complexity is log base 10 n, got it, whenever there is division, remember this, whenever there is division, if the division is happening by 10, you say log base 10 n, if the division is happening by 2, you say log base 2 n, if the division is happening by 5, you say log base 5 n,
this is how you compute the time complexity of like, this is how the logarithmic time complexities
are, so whenever you are writing a logic where the number of iterations depends on division and you are dividing, dividing, that is when something like logarithmic will come into the time complexity, that time the time complexity will not be we go often, if the number of iterations is based on division, time complexity will be logarithmic, remember this always, so you solve the first
problem, count digits, the next problem is reverse and number, let's go to the problem,
it states write a program to generate the reverse of a given number, print the corresponding reverse number, if a number has trading zeros, let's reverse will not include them, for an example,
the reverse of 104, 00 will be 401, instead of 00401 and these are some of the examples,
so let's get back to our extraction of digits concept, so according to the problem,
what they are wanting is, if I am giving you the number 7789, the reverse of this number will be 9877,
now we know that the extraction of digits happens in the reverse fashion, where we generate nine,
then we generate eight, then we get seven, then we get seven, somehow we need 9877,
which is the similar fashion, this is where the basic maths comes in, what do you do is,
you define a variable, sum, or maybe reverse number, reverse number equal to 0, and you say,
reverse number equal to reverse number into 10 plus last digit, remember this, this is what you
say, reverse number into 10 plus last digit, let's see how it works, so I'm saying initially,
reverse number is 0 to start off, let's do the step by step, first step nine gets generated,
so what am I doing is, 0 into 10, because reverse number is 0 at the first step, the first step,
seven, seven, eight, nine, modulo 10 will generate nine, and n would have, as of now become,

something like seven, seven, eight, nine, by 10, so n would have, as of now become, seven, seven,
eight, and reverse number says, reverse number is 0 into 10 plus the last digit nine, so the number
becomes, or rather the reverse number, as of now is nine, right, this is what the first iteration is,
let's do the next iteration, so the next iteration will be like, I'll just quickly omit this off,
so the next iteration, can I say it's seven, seven, eight, greater than speed, and seven, seven,
eight, modulo 10, hence the last digit is eight, I can't say it's, and this will be seven, seven,
eight, y 10, so n will become, seven, seven, this time, reverse number is stored as nine,
because you stored reverse number as this value, so this is nine, so what do you do is,
it's a nine, into 10 plus the last digit eight, so what do you get is, nine into 10 plus the last
digit eight, which makes it 98, the next time you get seven, the reverse is 98, into 10 plus seven,
which is 98, seven, next time it is seven, the 98, 87, into 10 plus seven is 98, 77, so you've got
the reverse number there, why it's simple, why did this work, it's very easy, understand,
you're getting an last digit, you're easily getting the last digit, nine, and after that you're getting
the next last digit eight, and you somehow want to add eight to that nine, you somehow want to
add eight to that nine, and the easiest way is, if you can somehow add a zero to this nine,
it'll become 90, and then if you add eight to it, it'll become 98, similarly, if you want to add
seven to it, if you want to add a seven to it, make it 987, you're again add a zero, and then
a seven to it, it becomes 987, again if you want to add a seven, you're again add a zero, this is
why at every step I am doing reverse number, into 10, whatever you have generated into 10 that
will allow the last unit digit to be zero, then when you add a digit, goes and gets into that
place, as simple as that, so again you saw that extraction of digits is actually handy, so
I'll take the number, and I'll keep the reverse number equal to zero, and then I'll go ahead and
say n greater than zero, and I can say last digit is n modulo 10, I can say reverse number is
reverse number into 10 plus last digit, and I can say n is n by 10, and at the same time I can say
out of reverse number is what I need, perfect, and I'll quickly run the code and see if it is
running fine, it is, let's quickly somewhere this and this should be running fine, this step,
so reverse number is solved again with the concept of extraction of digits, next problem is check
palindrome, now when I go to the palindrome problem, let's understand the problem, it states write
a program to determine if a given number is palindrome or not, true if it is palindrome or false
otherwise, so overhead states palindrome are the numbers for which reverse is exactly same as
the original one, for an example one to one, because if you take one to one and you do a reverse
of it, one to one's reverse is one to one, that is why it is called as palindrome, so if I go to the
iPad and write some other palindrome numbers, it's like one double three one, if you write the reverse
of it, it stays still one double three one, something like 11, 11 in itself is a palindrome,
seven, seven in itself is a palindrome, whereas one to three, the reverse of one to three is three to
one, this is not a palindrome, so any number which on reversal is the same number is a palindrome

so the definition of palindrome number sees the reverse of a number, so if I somehow can generate
the reverse of a number, if I somehow can generate the reverse of a number which I've already did
and then compare it with the original n, if they come out to be same, can I say the palindrome?
I can, so over here if you remember we took a reverse number and at the end of the day the reverse
number was nothing but the reverse number and now if I can compare this reverse number with the
original number, with the original number which is n, and if they come out to be same I can say
this palindrome, like if this is true, I can say this palindrome or I can say it is not palindrome,
can I see it, but wait, wait, if you remember we were dividing n by 10 and at the end of the day,
n was 0, so does n have the original number, no because we were, we are doing operations with n,
which has made n to be 0 at the end of the day, so what I need to do is maybe I need to store a
duplicate of n in some top variable and instead of comparing it with n, I can compare it with the
duplicate because n will be 0 at the end of the extraction of digits, it's very important to store
a copy of n somewhere so that it can be used to compare it with the reverse of a number,
that is the only change that you have to do and that is the only thing that you have to keep in the
mind. So if I remember this was the code, so what I'll do is I'll just go ahead and say duplicate
equal to n and over here I see if dope is equal to reverse number, I can go ahead and print true,
which is what they want, which is true or else you can just go ahead and print box,
that is what they required, give it a small, small, quickly run it and see if it is running fine,
should be and let's quickly submit this, on submitting you see that it is running absolutely
fine, next one is gcd or hcf but before that we will be solving Armstrong numbers,
so what is the definition of Armstrong number, it's very simple, imagine you're given this number
371, you take 3 cube, 7 cube plus 1 cube, if taking the cubes of these numbers, like cubes of
these digits and adding them up, sums up to the number itself, that is what you call as an Armstrong
number, even if you take 1634, 1 cube plus 6 cube plus 3 cube plus 4 cube, if you sum them up,
you actually get 1634, but something like 35, if you take 3 cube plus 5 cube, this is not going
to be equal to 35, this is going to be equal to 134, but this and this are not theme, whereas 1634
and the summation of cubes of its digits is 1634, so you call 1634 as an Armstrong number
or 371 as an Armstrong number, so I hope you've got the definition of Armstrong number,
if you've got the definition of Armstrong number, you know how to solve it, I've already taught
you the extraction of digits, you know how do you extract 9, you know how do you extract 8,
you know how do you extract 7, just have to do a cube of it, so can I see this time,
instead of taking any such duplicate or reverse n, I can just take a summation because
I need to sum cubes and last digit is what I have to sum, can I say sum equal to sum plus
last digit multiplied twice, last digit into last digit into last digit, can I do this?
So first time 9 comes in, what happens? 9 into 9 into 9 gets added to sum, next time 8 comes
8 into 8 into 8 gets added to the sum, so everything is getting added, first time 9 into 9,
sorry 9 into 9 into 9 got here, next time 8 went 8 into 8 into 8, next time 7 came in 7 into 7

into 7,
next time again 7 came in 7 into 7 into 7, so can I say at the end of the day sum will be storing
the summation of digit cubes and after that you need to just compare it with your original end, so maybe again keep a duplicate variable which stores the end because at the end of the day,
you have to compare if this and the duplicate are same, if this is, can you say it is an Armstrong,
if this is not, it's not an Armstrong, again what logic worked, extraction of digits,
if you know how to extract digits, can play around with them and you can solve this problem,
so Armstrong number is completed and the next problem that we will be doing is print all divisions, so when I say print all divisions, what does it mean, imagine I take a number like 36,
and ask you what are all the numbers that divide 36, so you can say 1 is something which completely
divides 36, you can say 2 is something which completely divides 36, you can again say 3 is something which completely divides 36. 4 is something which completely divides, is 5 something
which completely divides 36, no, if 36 is divided 5 it leaves the remainder of one, so not 5,
6 is something which does it, 9 is something which in does it and 12 is something which does it
and 18 is something which does it and then 36 is something which does so if I talk about 36,
The divisors of 36 are 1, 2, 3, 4, 6, 9, 12, 18 and 36.
These are the divisors of 36.
The question is very straightforward.
You have to print all of them in this particular order.
Okay?
Now, how do I do it?
It's very simple.
One thing I know for sure is,
I'm talking about divisors or factors
that definitely going to lie between
one to the number itself.
Can I say all the divisors will be between
one to and itself because anything greater than n
will never divide n for sure.
So, if I know all the divisors are going to be between
one and n, can I just loop from one to n?
That's my first thought process.
Since I know the divisors are from one to n,
my first thought process is very simple.
Let's do one.
Let's start the loop from i equal to one.
i less than equal to n and i plus plus.
This is something I know for sure.
So, i is used to loop around.
Now, the first value of i is one, then it's two,
then it's three, then it's four, then it's five,
and then so on till 36 in this case,
if it is n.
So, how do you determine that this i is a part
of all the divisors?
It's very simple.

Can I say, if it is completely dividing,
if i is completely dividing n,
then it has a factor of all the divisors,
and what do you mean by completely dividing?
It should leave a remainder of zero.
When I say leaving a remainder of zero,
does it mean if I do a modulated of i,
if I do n modulo i,
the value should be zero,
because it's completely divisible by what I will do.
I will say, okay, if n modulo i is equal to equal to zero,
I will go ahead and print i,
and c plus plus c out in Java system,
dot, dot, dot, dot, print, l.
I'll go ahead and print out i.
But in this way, I'll be able to print all the factors
of a particular n.
If I talk about the time complexity,
ask you, what is the time complexity of this code?
It will be extra, but it's very simple.
Since the loop is running from one to n,
it's taking n iterations.
And this is an unit operation.
So let's not compute calculated there by the time complexity
of this particular approach is nothing,
but we go off and very simple.
Over here, they've given us everything.
I want us to write this print divisors function.
Let's write the print divisors function.
It takes an n.
And as I said, it's very simple.
You go from one, you go on till n.
And you say, I, this one.
And you know, if n modulo i is equal to zero,
you say, see out, I, and then you give a bit of space.
That's what you need to write.
And on submitting, you will see that this is running absolutely fine.
But the time complexity is B, go off n.
I don't want a B, go off n time complexity.
Can I do it in a much better way?
I can't.
But it requires a bit of mathematical observation.
Let's see that mathematical observation.
So for 36, I said that one was a factor.
If one is a factor, one has to be multiplied with something in order to get 36.
So one was multiplied with 33.
And if you carefully observe, if this is one and the number is 36,
the other number will always be n by one.
If it is two, the other number will be n by two,
which is 36 by two.
That means 18.
So you get 18.
The next time it was three.

So the next time it is three.
So when I take three, it is nothing but 12, 26 by three.
Because 3 into 12 will be 36.
The next time I take four, it is 4 into 9.
The next time I take six, it is 6 into 6.
Next time, the next factor is actually 9.
And then you multiply it with four.
And the next factor is 12 and you multiply it with three.
And the next factor is 18.
You multiply it with two.
And the next time it is 36, you multiply it with one.
So if I have to write all the factors, these are all the factors.
These are definitely all the factors.
But you have a bit of observation.
If I draw a line at this portion,
I draw a line at this portion.
And I take this,
and I take this on the equal,
1 into 36, 36 into 1.
So can I say, even if I consider everything before the orange line,
I will get one, I will get two, I will get three,
I will get four, I will get 16, 36,
I will get nine, I will get 12,
I will get 18, and I will get 36.
Even if I take everything before the orange line,
even if I take everything before this orange line,
do I get all the factors?
I do.
So do I need to go beyond this orange line?
No.
What is this orange line?
If you carefully observe,
what are you doing?
A small number into a big number,
a small number into a big number,
a small number into a big number,
same number, same number.
And then, a big into small,
a big into small,
a big into small.
So can I see?
This is nothing but the square root of n.
Because when you take square root of n,
square root of 36 is 6.
Beyond square root, the numbers will grow,
the numbers will grow,
and it will nothing but a replication of the upper half,
the replication of the upper half.
So thereby,
this is nothing but a repetition of the upper half,
thereby I can say,
even if you loop till square root of n,
even if you loop till square root of n,

you actually can get your factors.
How?
If this is 1,
this has to be n by 1.
If this is 2,
this has to be n by 2.
If this is 3,
this has to be n by 3.
If this is 4,
this has to be n by 4.
If this is n by 6,
this has to be n by 6.
So can I see?
Now the looping is going to be very straightforward.
I loop from i equal to 1
till i less than equal to
square root of n
and i plus plus.
Can I do?
And can I see?
If n module i is a factor,
then print i as one of the factors,
print i as one of the factors,
what is the other factor?
We just now found out the other factor was n by i.
But we need to be careful.
What is the careful observation?
If it is 6,
the other factor might be 6.
So they are not two different factors.
So if you're taking the second factor,
if you're taking the second factor to n by i,
just make sure that n by i is not equal to i,
because the second factor might turn out to be the same factor.
It's very important.
The n by i,
which is the second factor,
must be compared with i.
And if they are not same,
you can say that maybe print.
That's your another factor.
That's it.
So first,
check if i is a factor.
Print it.
Now the other factor,
n by i,
with which,
the i will be multiplied.
Just check if this is not equal to i.
If it is not,
that's the second factor.
That's it.

So if you go ahead and print this,
the printing will be something like this.
First one,
and 36 will get printed.
Next two,
and 18 will get printed.
Next three,
and 12 will get printed.
Next four,
and nine will get printed.
Next one i is six.
Six gets printed.
But the other factor is six.
And it fails this condition check.
Thereby,
the other six doesn't get printed.
So all the factors are printed,
but they are not printed in a proper.
Yes,
they're not printed in a sorted way.
So what you can do is,
whenever you are getting all the factors,
probably you can store them into a data store.
And if you have seen the C++ STL video,
you know which data store you can use.
You do not know what will be the size,
or what will be the number of factors.
So the data structure that you'll be using
is a list,
a list in Java,
or a vector in C++.
You'll be using an undefined,
like you cannot define the size of the data structure.
You'll be using a list, okay?
And in that list,
you can store it,
you can store it,
store, store, store.
So everything will be stored in the list.
Once you have stored in the list,
sort the list.
And if you sort the list,
you will get everything in the sorted fashion.
So if I go back to the code,
what did I see?
We will be going till square root of n.
Right, we'll be going till square root of n.
We know this is a factor.
And we need a list now.
So let's take a list.
So this is our list,
a vector.
And we know

ls dot push
back of i,
a pushback i.
And we know the other factor is n by i.
If this is not equal to
i, that's the other factor again.
We'll see list.
Can you store the other factor?
And the other factor is n by i.
Once you have stored everything,
can you go ahead and say over here?
Sort ls dot begin.
ls dot n.
Once you have done this,
just need to print it.
You know how to print it?
This is how you print the list.
C++ STL video guys.
So I'll just iterate on the list.
I'll print the list with space.
So all of them are correct.
Now go ahead and print.
And it will be correct.
Why did I sort it?
Because they wanted us to print everything in the sorted order.
It's very important to sort the list.
If I talk about the time complexity,
what will be the time complexity?
Something before discussing the time complexity,
you're writing i lesser than equal to square root of n.
Instead of this,
because square root is a function,
and every time the function will be called,
because square root is a mathematical function.
In C++ STL,
this will be called every time,
which will take time itself.
Instead of writing this,
you can actually write i into i lesser than equal to n.
It will be like when i reaches 6,
it will be like 6 into 6 lesser than equal to 36.
It will work.
But the moment it goes to 7,
7 into 7 is not equal to 36.
So this will be false.
So this is the other way of writing for the square root.
This is what you can write.
So can I say this loop is running for,
we go off square root of n times.
Can I say this?
That this loop is running for,
we go off square root of n.
And then the number of factors,

whatever is the number of factors,
is sorting it.
The internal sorting function takes n log n.
What is the internal sorting function taking?
n log n.
What is n?
n is the number of factors.
n is the number of factors.
Right?
So can I say n is the number of factors.
n is not the original n.
It is the number of factors.
And then you are going ahead and printing it.
So again taking a number of factors time to print it.
Whatever is the number of factors.
So the overall time complexity in this case is,
we go off square root of plus we go off number of factors
into log of number of factors.
So number of factors into log of variety properly.
Number of factors got it quite simple.
And then plus this.
So we go of this plus we go of this plus we go of this
is the time complexity.
But the motive was to teach you that you can also find factors
in we go off square root of n.
Got it.
I can say I have also done print all divisors in both ways.
Now what is the next question at states?
Check for prime.
So what is the definition of a prime number?
A lot of you will say a number that is divisible by one
and itself.
This is the wrong definition.
Why?
Because according to this definition, one is a prime number.
Because one is divisible by one.
And one is divisible by itself.
Which is one.
It's a wrong definition.
Instead of this, the definition that you should always keep in mind is
a number that has exactly to factors one and itself.
That's a better definition.
One and itself.
A number that has two factors which is one and itself.
So if you remember, we just now computed factors.
So if you are given a number, something like 11.
Can I say 11 has a factor of one and 11 itself?
Any other number doesn't divide.
So 11 is a prime number.
If I say 13, 30 is a prime number because one and 13 divides it.
If I say five is a prime number because one and five divides it.
If I say four, four is not a prime number.
Why?

Because it is divisible by one.
It is divisible by two.
It is also divisible by four.
So there are three factors.
So four is not a prime number.
We take eight.
It is not a prime number.
Why?
Divisible by one, two, four, eight.
Not a prime number.
Something like 17 is a prime number because divisible by one and 17.
So what is the first?
The brute force.
What is the definition of brute force?
The algorithm which is the first algorithm or the initial algorithm that comes to your mind.
So can I see?
The simplest way to check is.
I will do one thing.
I will keep a counter equal to zero.
And I know it exactly has two factors.
So I will run a loop from one.
And I will go until n.
And I will say i plus plus.
And I will say, hey listen.
If it is a factor, it will be completely divisible by i.
And it will leave a remainder zero.
And I will do a counter plus plus.
And can I say, at the end of the day, if the counter turns out to be two, then I will say it's a prime number or else it's not a prime number.
So can I say this is the extreme brute force approach.
And if I write the extreme brute force approach, what will be the time complexity of the extreme brute force approach?
Can I say I'm running a loop n and these are unit operations so I can ignore.
And I see the time complexity will be be go of n because I'm running a l loop.
The check for every i which can be a factor and then I'm checking it.
And there is enough to that there are conditional checks which are unit operations can be avoided.
So this is be go of n.
And we know that factors are involved in the previous problem.
We did learn that all the factors can be found in square root of n.
Because if you remember 36, 36 out of factor one, the corresponding factor like the corresponding
other factor was 36, 2 is 18, 3 is 12, 4 is 9 and 6 is 6.
Even if you checked till square root of n, you could actually count all the factors.
And all the factors were 1, 2, 3, 4, 6, 9, 12, 18, 36.
You can count all the factors even if you loop till square root of n.
So why are you looping till be go of n?
Kindly loop till square root of n.
So what you'll do is you'll say i equal to 1, i into i lesser than n.
I've already taught you this and count to equal to 0.
And you'll say if n modulo i is equal to 0 that is definitely a factor.
At the same time, the other factor has to be different.
And if the other factor is different, then it will also be counted.

And then you can have a seam check.
If counter is equal to 2, prime else not a prime number as simple as that.
And if I talk about the time complexity, this loop ends up running for be go of square root of n.
If someone is coming up and asking you, how do you check for a prime number?
You say I know the square root method because I know the observation.
Every factor there is the other corresponding number with which it has to get multiplied in order to get the number.
Thereby I could just go up to square root of n, nothing beyond it.
Because still square root of n, I'll get all the factors.
So this is the code that I did right in the iPad.
Now look quickly, go ahead and submit this and see if it is working fine.
It is indeed working fine.
So we have completed the check for prime.
So apparently you have completed everything and the GCD or the HCFS left.
So let's go across and learn the GCD or HCF and then come back and take it over.
So what do you mean by GCD or HCF?
It's very simple.
Greatest common divisor or highest common factor.
Let me give you an example.
If I give you two numbers, n1 equal to 9 and n2 equal to 12.
So you need to find the highest common factor or greatest common divisor.
That actually divides n and that divides 9 and 12.
So where I down all the factors of 9, it is 1, 3 and 9.
Where I down all the factors of 12, it is 1, 2, 6 and 12.
Even 3, 4.
These are all the factors of 12.
So if I ask you the common factors, the common factors, if I go ahead and mark,
it is 1, 1, it is 3, 3.
So there are two common factors.
Out of these two common factors, which is the highest one, 3.
So I can say the GCD of 9, 12 is 3 because 3 is the largest number that divides 9 and 12.
If I ask you the GCD of 11 and 13, 12, it will be 1.
Because if we try to write down all the factors of 11, it is going to be 11 and 11.
All the factors of 13, 11 and 13.
So the common one is just 1 and that is the GCD.
So there will always be a GCD because 1 is the number that divides every other number.
So for every given two numbers, there will always be a GCD or HCF.
If I ask you, what is the GCD of 20, 30 or other 20, 40?
What is the GCD of 20, because for 20, 20 is the factor and for 40, 20 is also a factor.
So that's why 20.
So for two given numbers, one of them can also be a GCD of those two given numbers.
So that is what is the definition of GCD.
So all of you know how to find factors of two given numbers, a given N1 and a given N2.
So the last two problems, you have learned how to find factors.
So imagine, it's like N1 is 9 and N2 is 12.
So can I see, if I loop from 1 to 12 and for every number, I will check if they are dividing both.
Does 1 divide both?
Yes, 1 as of now is the largest factor.
Does 2 divide both? No.
Does 3 divide both? Yes, so 3 is the largest factor.
If I can check every number, and if every number divides both of them, if a number divides both of them,
I just replace that with my GCDs and so on.

The largest number that I get that divides both of them will be my GCD.
So if I write it, can I write this as i equal to 1?
Maybe I will loop till n1, maybe i plus plus, and I will see, if N1 modulo i equal to 0,
and N2 modulo i equal to 0, and I will say as of now, I know one thing for sure.
Any given two numbers, any given two numbers will always have a GCD of 1.
So GCD will be replaced by i. Can I see this?
Because i is starting from 1, so it goes from 1, then it goes to 2, then goes to 3.
So whichever number divides both of them, it just keeps on getting replaced.
The last number, which will be stored in GCD, will definitely be the largest because i is moving in the increasing factor.
You might ask me, but striver, what if N was given as 12 and N2 was given as 9?
Then the fall loop would have been running for 12 times, but i know one thing for sure,
that if i is 10, the 10 will not divide 9, there is no point in checking.
So can I say, instead of running till N1, I can actually run it for something like,
minimum of N1, N2, because i know if i run till minimum of N1, N2, like 9,
if i run till 9, that will suffice, running 10, 11 doesn't make sense.
For example, if the number was 20 and 40, if i run till 20, that will go,
because 20 is the largest factor i can have, running it for 21, 22 will not make sense.
So can i say, i can run the loop till minimum of N1, I can.
Thereby, can i say the time complexity will be, we go off minimum of N1, N2,
whichever is minimum till that i will run the loop.
So thereby the time complexity is minimum of N1, N2, now you might have questions in your head.
What's travel? We are trying to find highest and over here, what are you doing?
Going from 1, 2, 3, 4, and checking everyone.
But what if N1 is 20, N2 is 40, and i do the other way.
Yes, i do the other way and i see, i will run it from minimum of N1, N2,
i greater than equal to 1, i minus minus.
And i will say if N1 modulo i equal to equal to 0, and N2 modulo i equal to equal to 0,
i will print that as my GCD, and i will break, and i will break.
And you know what is the task of break?
Norwee's breaks out from the outer loop.
This is a conditional statement, this is not a loop.
For this break, which is the loop that is outside this break, this one,
it will break out from this.
How does it work?
The minimum of 20, 40 is 20, so 20.
20 modulo 20 equal to equal to 0, yes.
40 modulo 20 equal to 0, yes, print GCD, yes, 20, yes, break.
So the program terminates.
In this way, you will say, in this way, it will have a better complexity,
because the moment you are getting someone from behind, it breaks out.
You might give me this.
Yeah, definitely this might turn out to be a better one for a lot of cases,
but still the worst case will be minimum of N1, N2.
Imagine i give you two numbers.
N1 equal to 11, N2 equal to 13.
In this case, what will happen?
For 11 and 13, the highest common factor is 1.
It will start from 11, 10, 9.
It will not find anyone till 1.
It will eventually loop from 11 to 1.
So no matter what you do, the time complexity will be still this,

because if both the numbers of GCD as 1, it ends up running completely.
And you know when you determine the time complexity,
not you in the time complexity lecture, you always take worst case,
you always take worst case, the worst case is when you run it till 1.
So we saw the previous method was a brute force method
and was taking linear time complexity.
Now there is an algorithm known as equilibrium algorithm,
which is going to take much, much lesser time.
So let's learn about it.
What is equilibrium algorithm?
It states, if we give in to numbers N1, N2, the GCD of N1, N2,
whatever is the GCD of N1, N2, that's equivalent to the GCD of N1-N2, N2,
where where N1 is greater than N2.
Usually in books you will find them written as GCD of A, B,
is equal to GCD of A-B, B, where A is greater than B.
This is what the equilibrium algorithm states.
If you want a mathematical proof of it, you can definitely Google such
as a huge mathematical proof to it.
Again, we will not go deep into the mathematical proof
because it is not required in programming world.
You just need this concept.
So just have the concept.
Now, if I try to prove this with induction, it's very simple.
If I give you two numbers, imagine 15 and 20.
It basically states the greater number and the smaller number.
What is the GCD of 20, we know the GCD of 20, 15 is 5.
So it states the GCD of 20, 15, the greater number 20 and the smaller number 15 is equal to
it states greater minus smaller, 20 minus 15.
5, come at the smaller number 15.
So what is the GCD of 5, that is also 5.
So again, by induction also you can prove it.
You can take any two numbers and you will be able to prove that they are same.
So I can say that the GCD of 20, 15 is equivalent to the GCD of 5,
now you apply the equilibrium on the first two numbers.
And whatever you get, that will always be a truncated number.
You saw 20 getting truncated to 5.
Again, you will apply equilibrium to this.
If you try to apply equilibrium to 5, 15, it has to be the greater number at first.
So apparently you have to apply equilibrium to 15, 5 because 15 is the greater number.
So what do you do?
Let's say, let's apply equilibrium to 15, 5.
There will be like GCD of 15 minus 5, 10, 5.
Again, you can apply equilibrium to 10, 5.
If you apply equilibrium to 10, 5, it will be greater than 10 minus 5, 5, 5, 5.
Again, if you apply equilibrium to 5, 5, it will be both of them are same.
Anyone can be at the front, 0, 5.
So one quickly becomes 0.
The moment one of the numbers becomes 0, the other number is actually your GCD.
It's actually a GCD.
So the GCD of 20, 15 is this.
What do you do as you take two numbers?
Apply equilibrium to get it smaller.
Apply equilibrium to get it smaller.

Get it smaller, get it smaller.
Tell one of them is 0.
If one of them is 0, the other is 0.
So the algorithm is quite simple.
Start with A, B.
Keep on truncating.
Keep on truncating.
With A minus B, B.
And keep on doing till one of them, the greater number becomes 0.
But here's a catch.
This might end up taking a lot more time.
Imagine I give you a number like A equal to 52 and B equal to 10.
So it will be like, what will you do?
You will say GCD of 52, 10 truncated to GCD 42, 10 truncated to GCD 32, 10 truncated to GCD 22,
truncated to GCD 12, 10 truncated to GCD 2, 10.
And then you again take the bigger number at first 10, 2.
And then you'll be again like truncated to 8, 2 truncated to 6, 2, 4, 2, 2, 2 and 0, 2 and ultimately 2 is left because everything comes up.
So it's a lot of steps.
It's a lot of steps.
It might not improve the linear complexity by so much.
But there's a catch over here.
If you do 52 and it was 10.
And you're reducing it with 10 every time.
And you ended up at a place of 2, 10.
Isn't it equivalent to saying you're dividing it by 10?
And you were like 52.
If you're divided by 10, it's like 5.
You did it 5 times.
1, 2, 3, 4, 5.
And you ended up having the remainder because you are minus 10, minus 10, minus 10, minus 10 till it is possible.
It's equivalent to saying you subtract it till it was possible.
Or till it was greater.
But can I see?
Instead of subtracting it 5 times, you could have directly gone from here to here by saying 52 modulo 10, 10.
That's same.
2, 10.
That is same.
Instead of subtracting 10, 5, 10.
Directly get 2, even from 10, 2 to 0, 2.
You could have directly gone by saying 10 modulo 2.
Because that is what you did.
Indirectly, simple match.
So can I see?
The algorithm in a better sense will be.
GCD of A comma B, but definitely is greater than B.
Is equal to A modulo B comma B.
That's a bit of a.
And you go on doing it till it becomes 0.
That's a plus that.

You always take the greater now.
So the logic is very simple.
Forget about A and B.
The logic is very simple.
The greater modulo is smaller.
That's the logic.
And you go on till one of them is 0.
And if one of them is 0, then what is?
If one of them is 0, the other is GCD.
The other is GCD.
As simple as that.
If I try to quote this up, try to quote this up.
You're given A.
You're given B.
Can I say you go on till both of them are greater greater than 0?
Basement.
And I know one of them will be greater.
Either A will be greater.
In that case, I'll do A modulo B.
Or else if B is greater, I'll do B modulo A.
Instead of swapping, changing because I did not want to get into swapping, changing why?
Over here, it was 2 comma 10.
So A was this B was.
And he swapped it to this.
I did not want to implement that.
I just, I knew now 10 will be modulated.
So what I did was I implemented it in such a way, stating if A is greater, A will be modulated.
If B is greater, B will be modulated.
And once this is over, if A is 0, if A becomes 0, and the while loop is false, can I say that GCD will be B?
I will.
Or else can I say that GCD will be print of A?
Simple.
Simple as that.
Yes.
You can, you can just take this example, 52 comma 10, and you can do a dry run on this.
And you will see that eventually one of them will become 0, and that is when the looping ends.
And the time complexity of this equilibrium algorithm is B go off, log of phi, minimum of A comma B.
While log, if you remember, I clearly stated during the digit extraction or in the pattern videos or in the time complexity videos, that whenever there is division happening, whenever there is division happening, the number of iterations will be in terms of logarithm.
Over here, there is modulo, you are reducing the number by division.
The modulation operations are happening.
Then by the time complexity will be in terms of logarithm.
Now, why phi?
Why not something like log base 10?
In the digit extraction, it is always invited.
Over here, this A and V, it is always changing, fluctuating.
You are not sure what will be A and V.
Depending on different examples, it will fluctuate.
That is why they given them, given it a term, phi.

If you want to know what is phi in depth, there is a huge mathematical proof to it, not required.
You can read it but it is not required.
And minimum of A comma B.
Because that is the initial number where you start from.
That is the initial number where you start to do it.
So that is why the time complexity is this.
No one is going to ask you how and everything.
Just keep it in mind.
The time complexity is log phi minimum of A comma B.
This is the equivalent.
Read in the same code.
After if I have not given else because if this line doesn't, if this line executes the function would have end.
And this line will not execute.
That is why.
So now I will go ahead and run this code and it is running absolutely fine.
And if I will sum with this, we get the correct answer.
So with this, we will be wrapping up the basic match.
Next is basic recursion.
I have already made videos on it.
I will be attaching them to the playlist.
So watch this match video.
Those that go and watch the basic recursion video.
Once you have done this, then we will be going across to the next topic.
That is basic hashing and I'll be explaining hashing in depth.
But I hope for this video, you have understood basic match completely just in case you have.
Please hit that like button and follow reach will do comment understood so that I get an idea that you guys are understanding and you watch the video.
Till here, if you are new to our channel, what are you?
Please, please do consider subscribing to us because that is the only thing that each she motivated to make these kind of content.
And here with this, I'll be wrapping up this video.
Let's see in some of the video.
Till then Dubai.

Re 1. Introduction to Recursion | Recursion Tree | Stack Space | Strivers A2Z DSA Course

https://www.youtube.com/watch?v=yVdKa8dnKiE&list=PLgUwDviBIf0oF6QL8m22w1hIDC1vJ_BHz&index=8

So, before starting this video, I would love to thank the sponsor of this video, which is Scored Studio.
Now, if you are willing to practice interview problems, topic wise, this is the place where you should look for, because Scored Studio has over 2,000 plus problems and has all the problem solution in C++, Java as well as Python.
If you are looking for a guided path, then you can find for Python, for DBMS, for object oriented programming, operating system, computer networks, system design, web development, any
other thing that you are looking for, you will find guided paths for every other thing over here.
Also, also, if you are looking for any top company questions, let's say, if you are in interview at Amazon, if you are looking for Amazon questions, you can get all the top Amazon coding questions via a tag and all the solutions in C++, Java as well as Python.
Also, if you have an interview schedule, you can read their interview experiences, where there are 600 plus interview experiences from companies like Amazon, Microsoft, Adobe, Google, etc.
So, what are you waiting for?
Quote Studio has a lot of free resources for interview preparation, the link will be in the description.
Make sure you check it out.
Hey, if you are welcome back to the channel, I hope you guys are doing extremely well.
Recursion.
Yes.
One of the toughest topics for the beginners is what I am going to start today.
So, it's going to be an entire playlist.
Do you need to know any prerequisites?
Yes, there is a slight prerequisite.
You should be knowing what is a function.
Yes, what is a function?
Like, how do you write a function to add two numbers?
You should be knowing that much.
If you know how to write a function to add two numbers, that is more than enough to watch this playlist.
Well, this playlist being C++ or in Java doesn't matter.
Just watch this because I am going to write pseudo quotes and I am going to explain the entire
concept.
So, language will not matter C++, Java Python doesn't matter.
Okay.
So, to start of it, what is a recursion?
Let's read the first line.
When a function calls itself, very important.
When a function calls itself, that means if I am writing this function and inside that, this function, inside this function, there is this function being called again.
So, this is what is the definition of the first line.
So, in order to understand this first line in depth, how does it work and everything, let's take a super cake example to print one using a function, like using a recursive function.
And once you have understood how recursion works, then we can understand the second line

which states until a specified condition is met.
So, generally, there is a mean.
If you are using Java, there is a public static word mean.
If you are using C++, it is an int mean.
So, if there is a mean and I am saying a function, let us assume I write the function as f.
Okay, this is what I am calling and the f function is a void f function which says, hey, can you please print one in C++, it is going to be C out in Java, it is going to be system.out.printl.
And I call the same function again.
So, how does the program runs?
First the program comes to the int mean the execution, then it comes to the f and whenever you are calling the function.
It straight away goes and calls this function void f.
Okay, so I am going and calling this function void f.
Now, the void of goes and prints this line one.
Yes, it executes this line one.
So, I can say if I am having an output screen on that output screen at this moment, one will be print.
Okay, after this, this line is executed which states calling of the function, like calling of the self function f.
So, how will this work?
Will it again call back like this or how will it work?
It eventually calls back the same function.
But in order to understand in a much better way, what I will do is, I know if I just write it over here, there is a problem.
So, I will just create a replica of this function and I will write it over here.
Okay.
So, I am calling the replica of the function print one and again f.
So, this f again calls the same function.
So, basically it calls the same function again because this function is in your memory, right.
So, it calls the same function again and again this first line, yes, again this first line is executed.
So, again, there will be an output of one and again after this, this particular line will be executed.
So, how will this particular line get executed?
It again calls the same function in memory.
So, if I write down the f function again, print one, f again.
So, this f will now go and call it perfect.
So, again, this line will be executed print one.
So, again, you print it one and again, this line will be executed which again calls one more same function in the memory which is void f and again it prints one and after that again calls f.
So, this if you write the program, this keeps on going for every day, why?
Because there is no stop condition, yes, there is not a condition where you say, okay, wait, no more going to call me again and again and again, there is no such condition.
Thereby, if I keep on writing, if I keep on writing, this will keep, like, sorry, this will keep calling someone, again, this will keep calling someone, again, this will keep calling someone and this will keep on going forever.
So, it is a non-ending recursion or I can say an infinite recursion because you are calling the function again and again and again and there is no specified condition mentioned where you say that, hey, listen, now you are going to stop.
So, if there is no specified condition mentioned, it is going to fall under an infinite recursion,

goes and calling, goes and calling.
So, if I show you this in my code editor, by the way, this is a sublime text editor that I use.
If you want to use the same format, you can use watch my video, how to install a sublime text.
So, this is for sublime text, this is the int main, you can write it in Java as well, print.
Print is calling C out of 1, which is again a print statement and again print.
So, if I run this, if I just have a run on this, what will happen is it will print unlimited once, till it runs out of memory and throws a segmentation for it because it just, see, it did print so much, but after this, it did ran out of memory because the program cannot keep on running, keep on running.
So, this is what we call as stack overflow, what is stack overflow, let us understand.
Let us come back to the code.
So, if I show you the code, this was the main, correct, this was your main.
And this main, call this F, right, this main, call this F. And after this, this F, call this F. So, this F, yes, this F is not yet completed, this function call is not yet completed.
So, it will be waiting in the memory, correct, so the weight, so there is a stack, yes, there is a stack, you do not need to know any specific definition of stack, just, you can just take it as a thumb rule, there is a stack whenever you read data algorithms in the next chapters, you will understand what is a stack, but let us understand.
This function call, is this function call, calls this F, which indirectly calls this.
So, this function call is waiting, why?
Because this function called has called this, so this will wait.
Where does this function call weight?
I can say this function called, weights in line number 2 because the line number 2 has been called and it is waiting.
Now, this function called is called, over here you are calling this guy, if you see you are calling this guy.
So, apparently another function call with line number 2 is waiting.
Now, this function call is called, again this guy is called, so another function call at line number two is waiting. So if you just keep on calling them, yes, if you just keep on calling them again and again and again. So these particular function calls will be waiting in memory, will be waiting in memory because they are yet not completed. Now when do you call
a function to be computed? If the last line has been successfully executed but this has not been
because because this call this guy hence the last line has not been completely executed.
So this is the weight. This weight is what we call as stack overflow. Now remember if you keep
on calling, if you keep on calling like in the example in the sublime text, you kept in calling.
It went print one print one print one and at the end, it stopped at five two three zero three seven. Like it's it's a compiler independent. So in some compilers, it might go beyond that.
So it did stop. Why? Because this particular stack space would be keeping F. Yes, it would be keeping another F would be keeping another F would be keeping another F. But this has a specific
memory. It cannot just keep on keep on compiling up a lot of function calls because these function calls are yet not completed. So it cannot just keep on piling up in the memory. So that is
when the segmentation fault happens and it's commonly known as stack overflow. When there is
numerous function calls waiting due to recursion because you call the recursion. So one function

call was waiting. You got another. You got another. You got another. So it kept on waiting. This is
what we call as stack overflow. Got it. So this is why infinite recursions are not written because
because you'll end up having stack overflow. And this is the stack space like these function calls
are waiting. So in short, this is the definition or the first line definition when a function calls calls itself. I hope you have understood how the function calls itself. Now we are going to understand until a specified condition is met. What does this mean? Now if you carefully observe,
we call this. Then this guy called this. Then this guy called this. Then this guy. This guy called this. Then again, this guy called this. Then this it will keep on going. But there has to be a condition where you stop calling. Now that condition is something which is known as base condition. Yes, base condition. So let's quickly write down the function. So assume I'm writing
an f. And just assume there is a global variable f kept it as count equal to zero, a global variable.
And over here, I'm writing print count. And I'm doing a counter plus plus and I'm calling the function. Now if I keep on doing counter plus plus, what will this function like? I've already told you from the main, if I call this function, what will happen? Can you think what will happen?
Yes, I think you can. This function call will ultimately call this guy. Then the print of count will happen. And it's the value of count initially zero. So the output screen will print a zero. Correct. The output screen will print zero. Next counter plus plus will happen. So the counter value will change to one. And after this, this function will again be called to f of zero. And again print of count will be called. But this time counter value is one. So one will be printed. Again, you'll do a counter plus plus. So counter is increased to two. Again, you'll call this function. And again, this will call someone else. So it keeps on going like this and you'll be doing
print count counter plus plus again. And again, f. So you keep on doing and again print of count will
print the next two counter plus plus will make it three. And again, this will call this will keep on going. Now, you need to stop it somewhere. So the condition that you use to stop it is known
as the base condition. Yes, the condition that you use to say that here listen,
you're no more going to call it. You've got to stop. So assume I say that we're going to print only up till three. As you I'm saying, you're going to only print up till three. So how will you modify it? So you'll basically say function. Let's understand how this works. Right at the starting of the function, you're going to say initially the count is zero. You're going to say if count is equal to equal to four, I am going to return. Okay, I'm going to return. I'll explain you this as well. I'm going to return else. I'm going to print count. I'm going to do a count of plus plus. And I'm going to call the function. And in the mean, yes, in the mean, I'm going to call this function. So if I ask how will this work? Let's understand. So let's first keep the output screen. Let's see how this works. So I'm saying function. This function is called
what I mean. So it goes and calls this particular function. What's the first line? Count is equal to four. Is it? Count is zero. So count is not equal to four. Perfect. Next goes to the next line which says print count. What's the value of count? The value of count is zero. So you print zero. Correct. Next. Count is plus plus. That means counter becomes one. This line is executed.
Next function has been called. So in the stack space, this function will be awaiting this function
will await in the stack space. So in the stack space, this function, which is the line number,

line number one, two, three, four, five, in line number five, this function awaits. Why? Because it goes and again calls the same function in the memory, which this time checks is count equal to four. And then returns print again, the same thing count. Count the plus plus f. This time is count equal to four. No. Print count counts value is one. So to make things simpler, let's keep count as two. So that we don't have to print a lot of times of count as three. Count as three. Let's keep count as three. So that we don't have to print a lot of times. So count is printed again, count plus plus. So this line has been executed. Function has been called. So the moment you call the function, this guy is this guy goes into the stack space saying that okay, another function called at line number five awaits to be completed. So you again go and say f of if counter is equal to three. Is it counter is two? So it's not. So this line is still not executed. You will print counter. Then you will do a counter plus plus. So print counter will be executed. Prince counter plus plus makes it three. And then again, you'll call the function. Okay. Now this guy will again go into the stack space saying I'm still awaited. I'm still awaited to be completed.

And this time this function call will again call some other function. But this moment counter equal to equal to three. And counter is if you carefully observe counter is indeed counter is indeed three.

Counter is indeed three. So you can say that this line is this line will be executed. And you will say return. And you will say return. Thereby all these lines like all these lines will not be executed. All these next three lines will not be executed. And the moment you return, this function is terminated. Remember, if you're writing a return statement inside a function, the function gets terminated then and there. So this says, okay, I'm over. I'm over. So this function called call this, but this is over. Now the moment you come over here, this guy gets out of the stack space saying, okay, it's time to be executed. So this line is over now. And this time it comes over here.

And now it goes back. Why? Because the function call is over. If the function call is over, you can directly go back. Next it comes over here again. You can say this is done. Again goes back because this function call is again over. Now this is also completed. Goes back to the F and this line gets completed. So this is at the end completed. So what happened was you you called, called, called, this guy said, I'm over. Went back, went back, went back. This is what we call as a specified condition because you specified that counter has to be three only till then keep calling, keep calling.

The moment, the moment the counter was three, if you carefully observe over here, now the moment the counter was three, you said stop and return. It's like you stopped it. You no more call this function line because you stopped. And due to that stoppage, the other guys also completed their function calls. And this time there was no stack overflow because you did not call any further than three recursion calls. So this stop condition is what we call as base condition.

Now remember there can be a single base condition. There can be a multiple base condition. I'll be talking about all of them in the upcoming video. But I am assuming you have understood
the entire stuff. I'll try to code this up. So if you see the code, now this is what we will do is we'll keep a counter variable as zero. And what we are doing is we are printing count. And we
have something like count is equal to three then you can simply tell or else you can see how count and do a count plus plus. This was the code. Now let's run it this time. So if we run it this time, you'll see just zero one two being printed because print went on and call the functions. And this is the simple dry run. Now what is the recursion tree? Now there is something
in recursion which you'll be hearing a lot recursion tree. Now if you carefully observe, in order to explain you, I wrote the entire function, then I again wrote the entire function, then I again wrote the entire function, then I again wrote the entire function. But will someone do this every now and then? No. So this is basically truncated into a very simple thing. What we call
as a recursion tree. So basically I'll say the first function was called, which is specifically this guy. And then this guy called F. So this guy again called F. Then this guy again called F. And then this guy again called F. So again called F. But this guy returned this guy. If you carefully observe this guy, returned. So what you'll do is you return. So you went, you went, you went and you returned and you returned and you returned. So instead of writing the entire
function, we basically represent them in terms of F. And that is what we call as recursion tree. We'll be talking about depth and recursion. I'll be doing a lot more problems. So you'll understand what is the concept of recursion tree. So basically the same function call is what I've
represented in a smaller fund, a smaller representation. And that is what we call as recursion tree.
So in order to summarize the class, I hope you have understood everything. Like what is recursion,
like calling the function within itself, if I read recursion, after that you have understood what is a base case. The moment you say that, listen, you are not going to perform the below lines.
You go back. There's no more no more recursion being called go back. So that is what the base
case is. You also understood what is a stack overflow or stack space. Stack space is the space where the
remaining non executed functions like non completed functions rather because this function, this guy was called. So this function was yet to be completed. So stack space stores the yet to be completed once. And there is something which is known as recursion tree. It's basically the
diagram in which the function called someone, then that guy called someone. It's basically a shorter
version of the diagrammatic explanation that I gave you. So I hope you have understood all these
three things. Okay. So in the next video, I'll be doing couple, I think three or four problems. I'll be learning recursion in more depth. So just in case you have understood recursion in entire
depth, like a theory concept, please, please, please make sure you like this video. And if you new to our channel, please do consider subscribing. And yes, if you haven't yet checked our striver's SD sheet, the link will be in the description. Please check it out because a lot of people are getting pleased using that particular SD sheet. And now with this, I'll be wrapping up this video. Let's meet in the next one where we will be solving some problems from

recursion.

Re 2. Problems on Recursion | Strivers A2Z DSA Course

https://www.youtube.com/watch?v=un6PLygfXrA&list=PLgUwDviBIf0oF6QL8m22w1hIDC1vJ_BHz&index=9

So, before starting this video, I would love to thank the sponsor of this video, which is Scored Studio.
Now, if you are willing to practice interview problems, topic wise, this is the place where you should look for, because Scored Studio has over 2000 plus problems, and has all the problem solution in C++, Java as well as Python.
If you are looking for a guided path, then you can find for Python, for DBMS, for object oriented programming, operating system, computer networks, system design, web development, any
other thing that you are looking for, you will find guided paths for every other thing over here.
Also, also, if you are looking for any top company questions, let's say, if you are in interview at Amazon, if you are looking for Amazon questions, you can get all the top Amazon coding questions via a tag and all the solutions in C++, Java as well as Python.
Also, if you have an interview schedule, you can read their interview experiences, where there are 600 plus interview experiences from companies like Amazon, Microsoft, Adobe, Google, etc.
So, what are you waiting for?
Quote Studio has a lot of free resources for interview preparation.
The link will be in the description.
Make sure you check it out.
Hey, if you are welcome back to the channel, I hope you guys are doing extremely well.
So, today will be the lecture 2 of Recursion Playlist, which is basic recursion problems.
Now, in the previous video, we did learn about recursion in depth, what is, like, so please make sure you watch out the previous video and after that, only you start this video.
So, in this video, we are going to solve some problems.
The first one being print name 5 times, I am assuming you can do that, print linearly from 1 to n using recursion definitely, print linearly from n to 1, then print linearly from 1 to n, but in some backtracking, over here, I will teach you how can you do stuff using backtracking, kind of backtracking, and print from n to 1 again using backtracking.
Okay, so, I am assuming you can do the first 3, but still, I will recommend you to watch the entire video, because the concepts get more clear, the more you watch the videos.
So, let us take the first problem and do it.
That is print name n times, let us keep it n times.
So, the generic way is definitely to run a fall loop and print the name n times, but I want you to print your name n times using recursion, yes, that is a catch here, using recursion.
So, how can you do this?
You know, there will definitely be a main, which will be calling print, okay.
So, I am saying n times, so what you can do is, you can take the n from the user, yes, you can take the n from the user, so, input as n, in Java, it will be integer dot pass the n, n, and then you can say a function with probably 0 and n.
So, over here, I am going to teach you something, which is very important, 1 and n, so this is the first time, and I am saying till n, you print it, okay.
Over here, I am going to teach you something, change of parameters, which is very important.
So, I am writing a void f, okay, no global variables this time.
I am taking an integer i, and I am taking an integer n, okay.
Over here, I know the base cases, if I start from, yes, if I start from 1, I have to go till n, that is for sure, like I have to print it n times, so I know the base case is definitely going to be till I have not printed n times, so if at any moment, I exceeds n, I am going

to say return, because I am going to just print it n times, that is very simple, coming from the previous class, it is very, very simple, this is what we call it as the base case, grammar.
So, base case is decided on the problem statement, so this is the base case, okay, what after that?
After that, I will be like, okay, let us print it, so you will simply say print, let us assume my name is Raj, so I will print Raj, and then I will call f, I have printed it once, I did game up as 1, so the next time I will call it as i plus 1 comma n, okay.
So, let us understand, how does the flow works, so that you get a better perspective, so initially you will be taking integer n, assume n is given as 3, assuming n is given as 3, and again I will put the output screen over here, assuming n is given as 3, so n becomes 3, function says 1 and n, so this guy calls this particular guy with i's value as 1, n's value as 3, so the function call, yes, the first function call that you make is of f of 1 comma 3, that is the first function call you made, is the base case true, no because 1 is not greater than 3, 1 is not greater than 3, no, thereby you say print Raj, so Raj gets printed, okay.
Next you are saying function of i plus 1, so this guy calls a function with i plus 1, i was 1 over here, i gets incremented by 1 and calls the function by 2 and n is still 3, and over here you write the same line of code i greater than n because it is the same function which is being called, so in memory it is the same one print Raj and then f of i plus 1 comma n, okay.
This time let us understand if this base case is executed, i is 2, n is 3, so this will not be executed, print Raj, again you print Raj, remember this function of 1 of 3, call this f of 2, 3, so in recursion tree it will be f of 2 comma 3, this is how you create the recursion tree, now f of 2, 3 this is printed and this portion, so this guy goes and again calls f of i plus 1, so 3 comma 3 and again you write the same lines is i greater than n, then you say return, definitely this is false, so thereby you again go and say print Raj and is f of i plus 1 comma n, so this is false, so this gets executed, so Raj is again printed, f of i plus 1, so this time f like f of 2, 3 called f of 3, 3, so I can just draw that f of 2, 3 called f of 3, 3, now f of 3, 3 calls i plus 1 which is f of 4 comma 3 and this moment I can say if i greater than n then return becomes true, why? because i is 4, n is 3 and this guy comes out to be true thereby you are returning, so the moment you are returning these lines will be of no use any more, these lines will be of no use any more, so you return, now this is done, now remember f of 3, 3 called f of 4, 3, so you can do it in the recursion tree as well, f of 3, 3 called f of 4, 3, okay, so this is done, this is over, so I can say this function is over, thereby you can return, this is done, so this function is over, thereby this is over you can return, thereby this is over you can definitely return back to where it was called and now the function the program terminates, so this is how
easily the recursion did work and we did it in the parameters and not using any global variables,
so I was able to print my name n number of times, yes, I was able to print my name n number of
times and at f of 4, 3 I returned, so this got completed, this returns, so this got completed, this returned, so during the course this guy printed 1, this guy printed 1, this guy printed 1, this guy printed 1, and this guy hit the base condition and returned, so this is how you can easily do the first problem which is print the name n times using recursion, I hope you have understood this, so if I talk about the time complexity, space complexity, it's very very simple, I can say the time complexity to be be go of n because inside the functions, yes, inside the functions these lines are like be go of 1, like they don't take any time, so apparently you're calling one function, that guy calls one more function, that guy calls one more function, so you're
calling n functions, if you carefully observe 1, 2, 3, 4, so you're calling n functions, like

n functions, so I can say the time complexity to be be go of n, yes I can say the time complexity
to be be go of n, and what is the stack space? So in space complexity, we generally assume the stack
space, now this guy would have been in the stack, when you called f of 1, 3 would have been in the
stack, then f of 2, 3 would have been in the stack, then f of 3, 3 would have been in the stack,
So they were waiting in the stack till this base case, return, they were waiting to be completed.
So thereby I can say the space complexity is also be go of n. Now this space complexity is hypothetical like you're not using an array internal. The computer's internal memory uses the stack space. So thereby making the time complexity and the space complexity to be be go of n
and be go of n. I hope that makes sense. So guys we have done the first problem which is print the
name n times of 5 times. Now let's do the next problem which is print linearly from 1 to n.
So basically if n is given as 4, I want you to print 1, 2, 3, 4. This is what I want as the output.
So how will you do this? Again, very much similar to the previous problem. If I write down the function it's going to be i, n and it's going to be i greater than n and you're saying return
and this time the print statement will be executing i and then you can call f of i plus 1n.
It's a similar one just in place of the name you can print i right so that in the main, yes so that in the main when you call the function like you can take the input of n you can take the input
of n and then when you call the function with 1, n first time prints i which is
1 next time goes i plus 1 so 2. Similarly you can just go on printing till whatever your ns.
So there's quite similar to the previous one where I'm easily able to print 1, 2, n correct.
Now it's a time to think it in the reverse order and to the next problem which is print in the opposite fashion print in terms of n to 1. So like if I give you n equal to 4 you're going to print me 4, 3, 2, 1. Now can you do this? Can you do this? I think you can, yes you can.
It's very simple. As of now the code that we wrote started from i equal to 1 and then you did i plus 1 then you did again i plus 1 and you kept and call it. So if I'm if I have to print from 4
I can understand that the first i has to be 4 and the last has to be 1 after 1
before 1 you don't print. So I can say I'll just write the function and the opposite order.
It's going to be same i comma n but this time it's going to be if i becomes
lesser than 1 then i don't need to print and i can print the i and this time i can call it i
minus 1 comma n and during the start yes i repeat during the start of main i can take the input
of n definitely i can take the input of n and i can say f of n comma n to be called yes i can say f of n comma n to be called. So that if if assume n is given as 4 the first time or assume the n is given as 3 to have a better understanding. The first time you call it 3 comma 3 and this goes
calls it like 3 comma 3 correct this line doesn't execute print 3 so the output screen will definitely have something like 3 and then you call it as f of 2 comma 3 and again then you will call it as f of 1 comma 3 and again at the end you'll call it f of 0 comma 3 and this 0 comma 3 will be executed and you'll come back you'll come back and you'll come back. So automatically you
print 3 to 1 as simple as that. So it's the main stuff is you just change it over here where you say I'm going to do i minus 1. This is the main stuff where you change where you say I'm going
to do i minus 1 correct. So again we have also done the next problem which is print in terms of n
is 2 1. So three problems have been done. Now these were very basic. I want you to change it

and think it in the reverse order. What if I want to print from 1 to n but I don't want to do i plus 1 instead I say that the recursion call starts from n comma n usually like better brief let me just explain. I am saying print from 1 to n, print from 1 to n but I am not going to allow you
to use plus 1 like generally from 1 to n was meaning f of i plus 1 comma n was the function called
was the recursive function called that you are making. I'm not allowing you to do that.
I'm not allowing you to do plus 1. So how do you do this? So over I'm going to teach you something
as backtracking. So let's understand backtracking. Since I'm not allowing you to use plus you have
to use minus. So what I'll do is i comma n as usual. This time it will be like if i at any time goes lesser than 1 you return f of i minus 1 comma n and right after that I'm writing print of i. So over here I'm writing the main function over here I'm writing the main function and what I'm writing is let's take the input of n and let's call f of n comma n. So basically what I did was I said that you cannot use this plus thereby you can use minus 1. Why because I wanted to
I wanted you I wanted me to teach you what if I write the print line after the recursion call. What happens? Let's understand. So again let's quickly give the output screen
and assume n is given as 3. So the first call is 3 comma 3. So thereby the first call goes as 3 comma 3. The first call goes as 3 comma 3. Now let's understand this. Does this line execute
i lesser than 1? No because i is 3 and it doesn't execute.
Of it. Next I'm saying f of i minus 1. So this function call will go. Remember this the print line will not be executed. The function call will go. Okay let's call the function call and I'm saying i minus 1. That's 2 comma 3 and again let's write if i is lesser than 1 then return
and it's right f of i minus 1 comma n and then let's write the print of i and then this. Okay so let's see if the first line executes again no because i is 2. This time again you call the function with i minus 1 which is 2 minus 1 1 1 comma 3 and this time you write if i is lesser than 1 you return and you say f of i minus 1 comma n print of i okay let's see what happens this time f of 1 comma 3 i is 1 doesn't execute f of i minus 1 goes and calls this with f of i minus
1 means 0 comma 3 if i lesser than 1 return I'm not writing the remaining couple of lines i is 0 so this line this time is indeed true is indeed true and if this line is true what happens you basically say return executes so you return so this function line has been executed
correct now the print comes over here because after this the line of execution comes over here
and it says print i what's the value of i 1 so the output will be 1 next the execution line comes over here which means the function has been executed correct go back went back this line has been
executed thereby the function line comes to the next this print is i what's the value of i 2 printed next the line of execution goes here which means this function has been executed go back this line this has been executed you go to the next print i i's values 3 thereby this line has been executed go here hence go back so even by starting from 3 even by starting from 3
i was able to output in the linearly increasing order so it's not necessary that you have to start from 1 in order to print from 1 2 3 you can start from the back and you can do the task after the function call so basically by doing this what you did was you made it mandatory that this line this print line is executed at the first because you kept it after the function call you kept it after the function call so unless until this function call is over this line would have never been executed so you made sure the last guy has been executed first why how

did you
made sure that the last guy because this was the last guy has been executed first how was this
function completed first by making sure the print line is after the function call because this will only be executed if the base case is true thereby this gets executed this then this gets executed
then this let's see opposite way you made it by writing after the function what it so this is how you print from 1 is to n now i will be asking you to do the other one so this is what i've done but by backtrack this has been done i want you to do the other way i want you to print from n to
n basically if i give you as n equal to 3 i want you to print 3 to 1 but i don't want you to use this i minus 1 comma n i don't want you to use this okay so i not solve this i want you people to answer in the comment section about this let's see if you have understood the entire concept you
should be able to do it yes you should be able to do it so remember if i write this as a recursion
tree it's very simple you started with f of 3 comma 3 then you went to f of 2 comma 3 then you
went to f of 1 comma 3 then you went to f of 0 comma 3 which indirectly stated i'm over then the print line executed of 1 then you said after this the print of 2 was executed and right at so this the print of 3 was executed so this is how the recursion tree will look like for this particular stuff so i hope that is you have understood all the problems and you have understood the recursion
in depth for the basic kind of problems of one one the recursion so just in case you did please please make sure you like this video and if you're new to this channel please do consider subscribing
and if you haven't checked out our strius is d sheet the link is in the description please make sure check it out with this let's wrap up this video and meet in the next one where i'll be discussing some other interesting problems bye bye

Re 3. Parameterised and Functional Recursion | Strivers A2Z DSA Course

https://www.youtube.com/watch?v=69ZCDFy-OUo&list=PLgUwDviBIf0oF6QL8m22w1hIDC1vJ_BHz&index=10

So, before starting this video, I would love to thank the sponsor of this video, which is Scored Studio.
Now, if you are willing to practice interview problems, topic wise, this is the place where you should look for, because Scored Studio has over 2000 plus problems, and has all the problem solution in C++, Java as well as Python.
If you are looking for a guided path, then you can find for Python, for DBMS, for object oriented programming, operating system, computer networks, system design, web development, any
other thing that you are looking for, you will find guided paths for every other thing over here.
Also, also, if you are looking for any top company questions, let's say, if you are in interview at Amazon, if you are looking for Amazon questions, you can get all the top Amazon coding questions via a tag and all the solutions in C++, Java as well as Python.
Also, if you have an interview schedule, you can read their interview experiences, where there are 600 plus interview experiences from companies like Amazon, Microsoft, Adobe, Google, etc.
So, what are you waiting for?
Quote Studio has a lot of free resources for interview preparation.
The link will be in the description.
Make sure you check it out.
Hey, if you are welcome back to the channel, I hope you guys are doing extremely well.
Today, we will be solving a very, very interesting problem in recursion, which is the summation
of the first N numbers.
Now this can be done using a very simple for loop, but I want you to do this using recursion because that will create your base of recursion very, very strong.
I am going to teach you a couple of ways.
The first way being of the parameter wise ways, where I will be showing you how to do this using parameter.
The next one will be functional way, where the function itself returns the answer.
Okay.
So, to start off with, I hope you have understood the question.
Some of the first N numbers assume N is given as 3, then the summation will be definitely 6 because 1 plus 2 plus 3 is equal to 6.
Some of the first 3 numbers is what the question states.
So, to start off with, let us learn the parameterized way.
Now, when I say parameterized way, what does that mean?
That means assume I write a function is assume I write a function and I say I and I carry a variable sum.
Okay.
And that is the recursive function that I am writing.
And I say that, okay, I am going to think this as a loop.
So, in the previous lecture, we saw that we started from 1, we went on to 2, we went on to 3.
So, we are going to do this till we do not reach N or probably we can do it the opposite way from N to 1.
So, let us assume we do it from N to 1.
So, we can say if at any moment, I is lesser than 1.
Yes.

If at any moment, I is lesser than 1, I can definitely print, yes, I can definitely say let us print the summation, okay, and then try our return, make sense.
And apart from this, what I am going to say is, okay, f of i minus 1 sum plus i is what I am going to pass.
So, this is what my recursive statement will be.
If the main will be very simple again, the main will be taking an N as the input and then we can just call f of N, comma, initial summation as 0.
Now, this is what I have taken.
Now, for an example, let us assume this is the output screen and let us assume N is given as 3.
So, what I am calling for the first time is, I am saying i to be 3 sum to be 0, right?
That is what I am saying.
i to be 3 and sum to be 0.
This line is not executed.
This line is executed and it thereby calls another function in the same way.
The same function is called in the memory and it passes i minus 1 which is 2 because i was 3.
So, it ends down to 3 minus 1 which is 2 and sum plus i sum was 0, if you see sum was 0.
So, 0 plus 3 will become 3.
So, it goes as 3 again is the if line executed, no, the if line is not executed.
So, the functional line will be executed and that is i minus 1 which means something and then sum was 3 plus i this time is 2.
Now, this time this function again calls a function which says function i minus 1.
So, i was 2.
It becomes 1 and the summation 3 plus 2 becomes 5 is the if statement executed, no because
the if statement if you carefully observe it is i greater lesser than 1 and that is not the case.
So, it is not executed, correct?
Thereby it goes again and calls the same function with i minus 1 comma sum plus i.
This time i minus 1, i was 1.
So, this time the functional call is made to f of i minus 1, 1 minus 1 becomes 0.
Then summation, summation was 5 plus i, i was 1.
So, it calls it as 6.
What is the if statement?
If statement stated i lesser than 1, i less than 1 is that the case.
Yes.
So, what it does is print the summation, summation was 6.
So, on the output screen the 6 is printed and it says return.
So, thereby this functional line is no more executed and it directly returns from here.
So, if it returns this function is done and now it returns thereby saying this function is done.
So, this guy again returns thereby this function is done thereby this guy returns.
So, we did a parameterized function where we can say that our recursive tree was 3 comma 0 where this being the i and sum, we went on to f of 2 comma added this 3, we went on to f of 1 comma added this 2, we went on to f comma 0 added this 1.
So, whenever we ended up at 0, we just printed this because we were adding the i, adding the i every time to the parameter and that is what we printed.
This could have been done in the other way where you could have done i plus 1 as well, but I am just teaching you the parameter way in how can you actually carry the parameters.
Yes, because over here we increase the parameters and at the end of the day, at the end of the day, we came here and print this particular parameter which is 6.

So, this is how the recursion tree will be and at the end it came back came back and went back.
This is how you can do it using parameterized.
Now, if you understood the parameterized, it is time to understand the functional.
How do you generally write the functional?
Because functional means you do not want the parameter to do the work, you write a recursive
function where you say that, hey listen, this is my value of n, nst and give me the summation of 6. So, if I am asking you to write such a function, that is when you cannot use parameterized
because you want the function to return the answer, you do not want it to print, you want it to return.
Why?
There will be a lot of cases when you learn dynamic programming that you want the function to give you back something instead of printing, right.
So, basically to understand the concept before writing the recursive code, let us, so these are all patterns that I am teaching, it is not that I am just writing the codes, these are patterns I am teaching.
If you learn these patterns, if parameters are there, then you have to do it in such a way, if there is return, then you have to do it in such a way, to learn these patterns, once you are very confident about these patterns, you can solve any problem in recursion, that is my word, okay, to go with functional, let us understand when I say functional word does that mean.
Now, if n is given as 3 for a reason, if n is given as 3, can I say this, can I say this?
I can say I will do a 3, I will do f of 2, where I know, where I know f of n means summation of first n numbers, okay, I know this, so if I write 3 plus f of 2 make sense, then when I am calling for f of 2, can I say, okay, that can be written as 2 plus f of 1, and when I am at f of 1, can I write this as 1 plus f of 0, and everyone knows that f of 0 is nothing but 0.
So, I can think it in such a way that, okay, whenever there is an f of n, I know n will be there, I know n will be there, which is 3, because if I am saying f of 3, I know the summation will contain 3, but I need the summation of the rest of the 2 numbers, so this is the idea that I am going to follow, so what else is, okay, let them give me the n, if at any moment n is equal to 0, I am going to return a 0, because if f is n is 0, I know the summation of the first 0 numbers is 0, or else I am going to return simply n plus f of n minus 1, okay, and I am going to simply turn n plus f of n minus 1 done, now question might arise, how will this work, let us understand, let us write the main function again, so if I write mean, and I say n is given as 3, and I call f of 3 or f of n rather, and I write print, print this f of n, so what is being done, this f of n goes and calls this, so let us call this, with what value, with the value 3, f of 3 is called, is this line executed, no, n means 3 plus f of n minus 1 is called, remember this, there is a line incomplete line, 3 plus something, incomplete it, that incomplete it will be called in recursion, till it does not comes, I am waiting 3 plus 3 plus something, something I am waiting, got that, so f of 2 n goes, it states if n is equal to 0, it says no, so I am like okay, return 2 plus f of 1, this time 2 plus this line waits for f of 1 to be done, so I go and call f of 1, if n is equal to equal to 0, again I say no, remember this, you could have written n equal to 1, that is your choice, you can write it, I am just teaching you patterns, that is it, return 0, it doesn't works, return 1 plus f of 0, again this line awaits, I am awaiting, let him come back, so let us go, f of 0,
when you go for f of 0, let us understand what happens, if n equal to equal to 0, return to 0, this guy comes in with a value 0, returns to 0, this time I am returning something, I am giving back something, so it is returning to 0, I hope that makes sense, so whenever this function is called, this function is called, it came back with 0, came back with 0,

so can I say, now instead of writing this f of 0, I can actually write this 0, because this function yielded you a value 0, so thereby 1 plus 0 is 1, hence return this guy returns a 1, 1 plus 0, returns a 1, so can I say now instead of writing f of 1, can I write 1, because 1 has been returned, thereby 2 plus 1, can I say this guy returns a 3, indeed I can say, the f of n minus 1 goes away and I can write 3, so thereby 3 plus 3 is 6, thereby this guy returns 6 and you say print, so output is 6, this is how, yes this is how you can write a functional stuff, where the problem is broken down into smaller parts, yes the problem gets broken down into smaller parts and once it is broken down, it easily gives you the output, so coming across to the code, the code is very very simple, you say identity sum, you can call identity n, and you can simply write, if n is equal to 0, please return a 0, else return an n plus sum of n minus 1, and over here probably I can just keep n equal to 3, you can take this as input as well, see how sum of n, so if I just write it, you will see this perfectly works, it is all at perfectly, that is a very simple one, okay, so I hope you have understood how to get the sum of first n numbers, now if I give you a very simple task, task is factorial of n, what is factorial of n, whenever I give you n equal to 3, it is 6, whenever I give you n equal to 4, it is 24, basically 1 into 2 into 3, this I am it is 1 into 2 into 3 into 4, correct, multiplication instead of addition, multiplication instead of addition, so how will you do this, I mean that is very simple this time, very simple, if I write the code, so I will leave the parameterized, you can do it yourself, I will try to write the recursive code, okay, now you know one thing for sure, in the recursive code, this will become a factorial, correct, and this will become into, because 5 into 4 into 3, and this will become n minus 1, but we need to understand
will the base case change, will it be n equal to equal to 0 this time, let us see, if I write this and if I run this, what happens, let us understand, if I just give it a run, see, okay, sorry, let us give it a run, so if I, let us change it factorial, now let us give it a run, so if I give it a run, you see the output being 0, you see the output being 0, why, due to this base case, let us understand, how, why did this base case give you 0, so if I write f of n, and I write the base case is n equal to 0, and I say return 0, and I am saying return n into factorial of n minus 1, if this is the code, and I am writing the main function as n is given as 3, and I am saying print f of n, so basically this guy goes and calls this with 3, not executed, 3 into f of 2, not executed, if line is not executed, so return 2 into f of 2 minus 1, which is 1, so this is gone, f of 1, again you are writing n equal to 0, not executed, return 1 into f of 0, f of 0 is going, and this time the f is executed, and you return a 0, and you return a 0, which is absolutely wrong, because the moment you return a 0, this guy returns a 0, and if I just strike it off, instead of f of 0, you end up writing a 0, which means 1 into 0 is 0, you get a 0, thereby this is strike of 0, 2 into 0 is 0, 0, this gets 0, 3 into 0 is 0, you get a 3, so what should you have returned, yes, you should have returned ideally 1, or or or, you should have done this, this is also correct, or even if you keep this, this is also correct, you should return 1, because in case of multiplications, if I returning 0, the overall product gets multiplied to 0, thereby giving you a wrong answer, so this is what I wanted to explain you, again, what about the time complexity, you are basically doing one call, two call, three call, till n calls,
so I can say big of n is the time, and the space complexity is an auxiliary space, stack space, because this awaits, then this awaits, so there will be big of n, n functions will be awaiting to be completed, so stack space will be taken, so in stack, this is what the time and space complexity will be, why big of n is the time complexity, first time this function is called, next this time, and these functions are all unit functions, you do not take a lot of time inside these functions, thereby the number of function calls will be equal to the number of time complex, it is equal to the time complexity, so guys, I hope you
have understood the entire lecture, just in case you did, please, please make sure you like this

video, and if you need to the channel, please do consider subscribing, please, please do consider
subscribing, and yeah, with this, let's wrap up this video and meet in the next precaution video,