**Design and Analysis of Algorithms**
**Prof. Abhiram Ranade**
**Department of Computer Science Engineering**
**Indian Institute of Technology, Bombay**

**Lecture - 2**
**Framework for Algorithms Analysis**

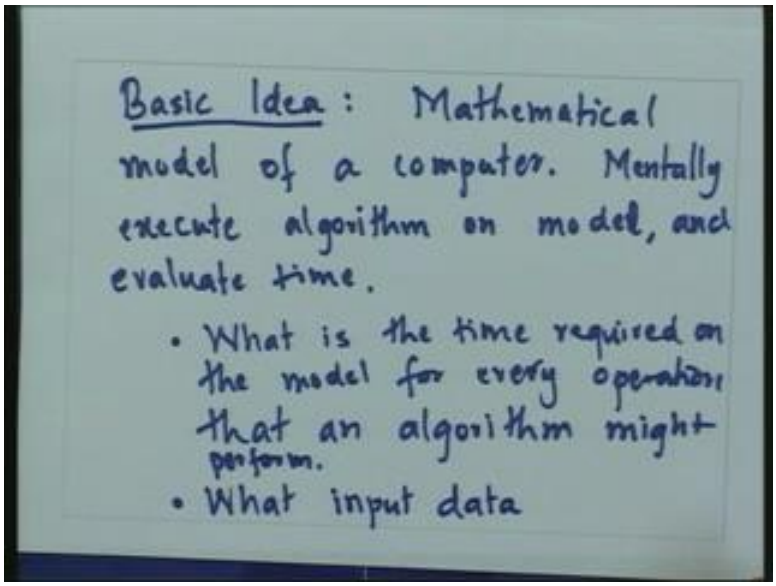
Welcome to the second lecture of the course on Design and Analysis of Algorithms. In today's lecture, we are going to develop a Framework for Algorithm Analysis. In this course, we will be designing many algorithms for solving many different kinds of problems. We will want to compare these algorithms and even just plain evaluate them. And for this, we will need some sound mathematical bases.

And that is, what we are going to do. We are going to design a formal framework using which we can evaluate algorithms. And we will and also compare them. This is going to the topic of this lecture and also the next few lectures. The framework that we designed could be used not only for comparing the execution time of algorithms, which is what we primarily mean; when we say algorithm analysis. But, it could also be used for comparing other resources that an algorithm might use.

For example, an algorithm might use varying amounts of memory. So, we could use essentially the same framework that we are going to discuss very soon. And use that framework to formally compare the memory requirements of different programs or different algorithms. The basic idea in designing in the framework that. We are going to discuss today, is actually very related to the kind of analysis that we did in the first lecture.

Except we are going to make it a little bit more formal. Essentially we are going to make a mathematical model of a computer. And then, we are going to take our algorithm and mentally execute that algorithm on that model. And then through this execution and by mentally executing, we will be able to tell how much time. The algorithm takes and that is essentially going to be involved in doing the analysis. That is what the analysis is going to have.

(Refer Slide Time: 02:58)



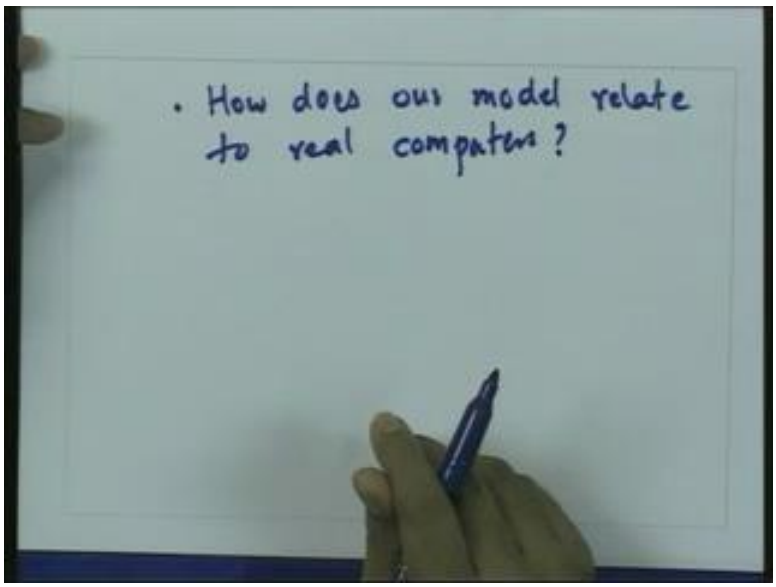

So, let me write that down the basic idea. We will make a mathematical model for computer. And we are not going to execute our algorithm on any specific real computer. But, will execute it mentally will imagine it is execution on this mathematical model. So, let me write that down as well, mentally execute algorithm on computer model and evaluate the time. This is the basic scheme; this is the basic idea that we are going to develop.

In order to develop it, we need to answer several questions. So, the first question naturally is what is this mathematical model going to be? Which essentially is the same thing as saying how long should be assign, what time should, we assign for each of the operations. That is comprised that is used in our algorithm. So, we need to answer questions like. What is the time required on the model for every operation that an algorithm might perform.

Then, we also said that we need to execute. We need to mentally execute the algorithm on this model. However, every algorithm or most interesting algorithms will require data. Some input that needs to be given to this algorithm. So, an important question that we need to answer is, what data should we be giving, what should be the input data. This is an extremely important question, because the time of execution in general will depend upon the input.

So, when we say we want to estimate the time taken by an algorithm. We have to be very clear in saying what input is being given to that algorithm. We may make mathematical models. And we may develop them and we may estimate the time taken on those models. Of course, there is the important question, which we need to answer, which is how does all this relate to real computers?

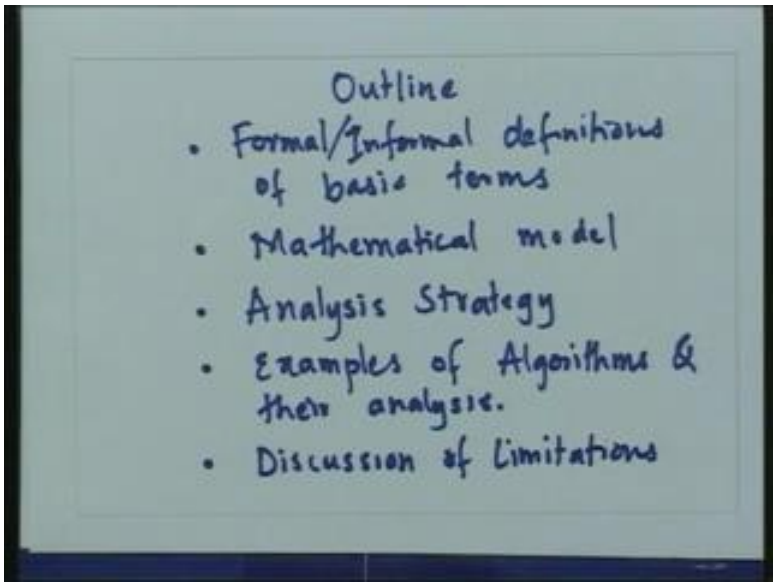
(Refer Slide Time: 06:17)




So, how does our model relate to real computers? If our model is terribly different, then our conclusions for the model might not be too useful for real computers. And of course, we do not really care that much about the mathematical model. We want our conclusions to eventually apply to real computers. And therefore, this is an extremely important question that we need to consider.

Over the next few lectures, we are going to answer these questions and also many of the other relevant questions. And you will see that all these questions can be answered nicely. And in that, we will be we will finish our development of our framework.

(Refer Slide Time: 07:32)

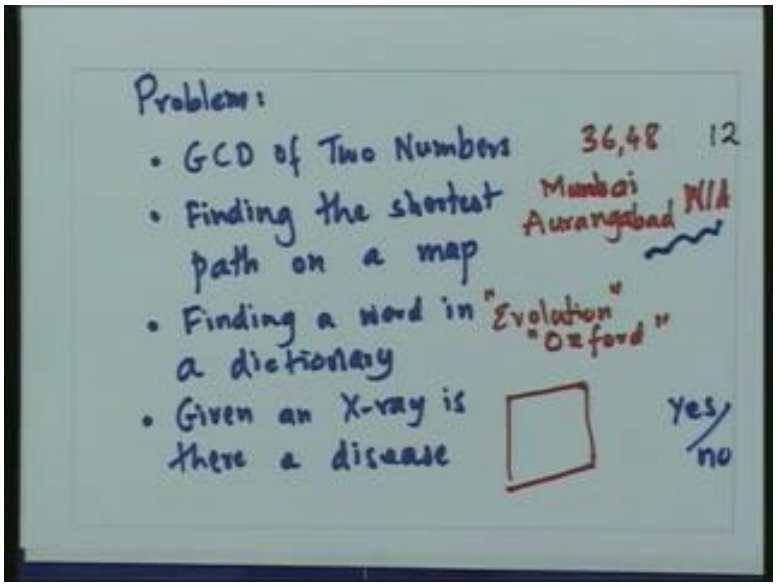



Here is roughly what I am going to talk about in the next few lectures. I am going to start by discussing some fairly basic terms. So, we will try to formally define or at least semi formally some basic terms. Then, we will present our mathematical model. After that I will discuss, the general overall analysis strategy which we are going to use.

These will for example, answer questions like what should be the input. We will also be taking a number of examples of algorithms and their analysis. And finally, we will conclude with a discussion of the limitations of the model. This will essentially be an answer to the question of how well do our conclusions to the conclusions that we draw for the mathematical model relate to real computers. And of course, I do not strictly I would not strictly discuss these points in the order I have written them.

I will discuss examples and I will discuss limitations and may be alternate a little bit. But, this is basically going to be the gist of this lecture and the next. Let us, begin now with some basic terms that we are going to use. In day to day life, we often use the same term to mean different things. In scientific discussion, it is important to fix the meanings for every term. So that we do not confuse ourselves later on and we do not end up with fallacies of any kind.

(Refer Slide Time: 10:06)




So, let us start by discussing the very first very common term that we are going to use which is problem. Before, I give a definition of a problem I would like to give some examples. And from those examples, I will try to motivate this definition. When I say problem in this course I will mean, what we usually mean is, in the sense of the problem of computing the GCD of two numbers.

Or say something like the problem of finding the shortest path on a map or maybe say finding the meaning of a word in a dictionary or may be even something like given an X-ray determine if there is any disease. You may notice that when we are talking about a problem. There is typically certain input which is which needs to be supplied and a certain output that needs to be generated.

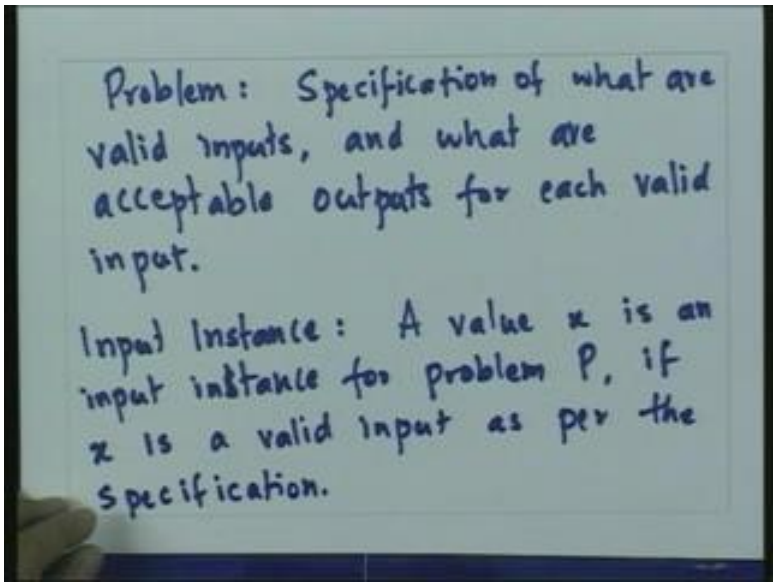
Let us take an example of this. So for example, if you are asking about the GCD of two numbers. We could say that the input consists of say numbers like say may be 36 and 48 the GCD of which will obviously, be the number 12. Say for the problem of finding the shortest path in a map may be the input could look like say name of a city may be Mumbai and say a city say Aurangabad.

And we would have to supply which map we are going to use. So maybe, we use the western India Automobile Association map and that will also have to be supplied. That will also that map will also have to be supplied as a part of the problem definition. For finding a word in a dictionary may be we have to supply the word.

Say for example, we take the word evolution. And we will also have to name what dictionary we use, say may be the oxford dictionary or something like that. For the last problem, we will have to supply an actual X ray. Say some actual picture and in this case the output would be something like either there is disease. Say we just a yes or no.

For the shortest path, the output would be say the actual map, the actual path on the map. For the evolution for finding the meaning of the word evolution the output would have to be the actual meaning that you would get while after looking at the dictionary. At this point, we have I think we have a good sense of what a problem is and we can write down a reasonably simple definition.

(Refer Slide Time: 14:19)

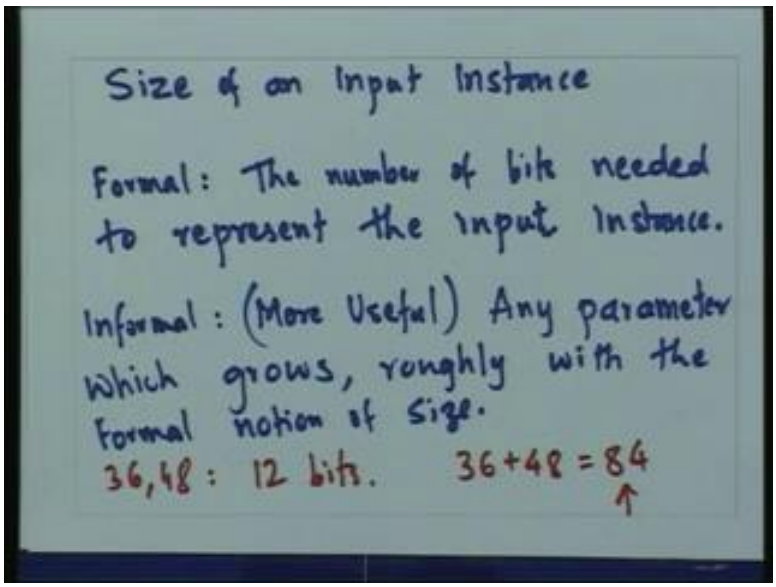



So, let us do that. So, when we say problem in this course. We will mean a specification of what are valid inputs and what constitute acceptable outputs? Acceptable outputs for each valid input. So, we looked at this earlier. So for example, for the GCD problem 36 and 48 constitute valid inputs. And for these the acceptable input is 12.

Finding the shortest path names of two cities in the map constitutes a valid input and acceptable output would be the description of the path and so on. Of course input which is valid for one problem need not be valid for another problem and typically is not. So, numbers will not make sense as input for say the dictionary problem or and words will not make sense as inputs for the GCD problem; obviously.

We often use the phrase input instance and this is nothing but a valid input value for a given problem. So, I will say that a value x is a input instance for problem p if x is a valid input as per the specification. So, 36 and 48 are 36 and 48 together constitute an instance for the GCD problem. Mumbai Aurangabad and map constitute an instance for the GCD problem for the shortest path problem and so on.

(Refer Slide Time: 17:17)




Another important term that we need is that of a size of an instance. We will often not necessarily use the term input instance, but we will just stick with instance. Instance will always mean input instance or we could even say problem instance. So, when we say the size of an input instance, we mean in a formal sense we will mean a following. Will mean the number of bits needed to represent the input, the input instance. Let me just clarify that. So, a specific input instance will have a certain specific size.

So, again let us go back to our examples. (Refer Slide Time: 10:06) So for example, if you look at 36 48 which constitutes the input instance for the GCD problem, then we will have to ask the question, how many bits are needed to represent 36x and 48? So, here there is a question of how we represent numbers in the first place.

So suppose, we say numbers are going to be represented in binary. Then, 36 will require 6 bits. And so will 48. So in this case, the input instance will have length 6 plus 6 or 12. As far as, the shortest path in the map is concerned, somehow or the other we will have

to represent the map. There are various ways of this representation we will see some of them later on in the course.

In general a map can be thought of as a graph which you have probably seen in the prerequisites for this course. And a map could be represented as a matrix. And a matrix could be represented as an array bits if you like and in that way we can represent maps as well. This definition that, we have given. This formal definition that we have given is often a little bit inconvenient for directives.

So often, we settle for a somewhat more informal definition. But in fact, this typically is something that is more useful. And informally we might say the size of an instance and we might mean any parameter which roughly grows with the official definition of the size of the instance.

(Refer Slide Time: 17:17) So, let me write that down. Any parameter which grows roughly the growth may not be exactly predicable with the formal notion of size. So, let us go back to our GCD problem. There we said that the size was the size for 36 48 was 12 bits. But, instead of making this is the definition of size. We could say that the size is simply the sum of the numbers. So, 36 plus 48, which is 84.

So, we could think of 84 itself as our notion of size of the input instance rather than 12 bits. In fact, if you go back to the first lecture you will see that this was the parameter that we used when we analyzed the GCD algorithm in the first lecture. So, we said that the size the sum of the numbers u and v will keep on decreasing.

And in fact, this is really the reason why we are interested in the notion of size. Usually we will expect that the time taken by an algorithm will increase with the size of the instance. And therefore, if you are going to evaluate an algorithm it is only natural and it is only fair in some sense that we also mention what the size of the input instance is.

So, if an algorithm takes a long time on a large instance on an instance of large size. Then, that is but if it takes large time on an instance of a small size. Then, we should potentially say that that algorithm is not a good algorithm or at least it is not a fast algorithm.

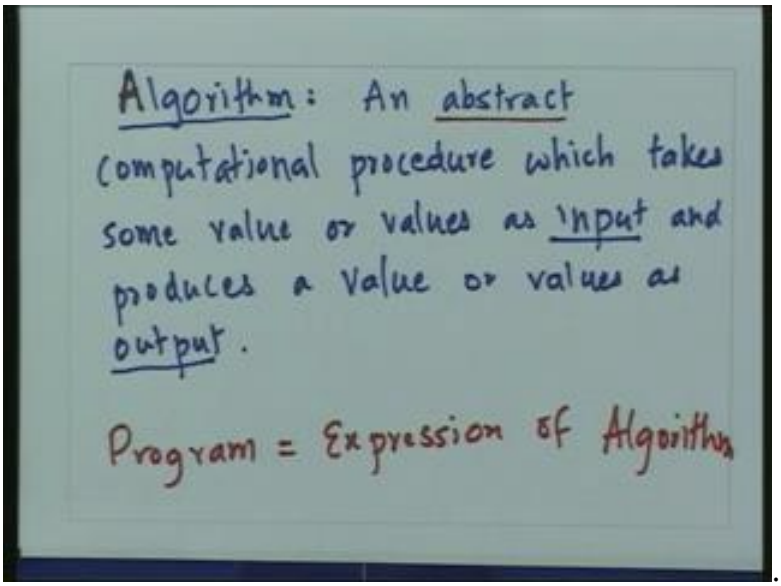
So, let us go back to the other problems and maybe we can think of what the notion of size is going to be over there. So, going back to our shortest path problem, a notion of size could be the size of the map, so the number of roads in the map or the number of roads and the number of cities together. So, clearly finding a shortest path in a map which only involves one road is going to be really easy.

And therefore, we should expect we should an algorithm which takes us takes us short time on such a small instance should not really be thought of as a great algorithm. On the other hand if an algorithm takes a small amount of time on a map which consists of 1000 cities and 2000 roads. Then, that algorithm we should certainly say is a god algorithm. So, essentially that is the idea, we want to when we evaluate algorithms. We want to evaluate that in comparison evaluate the time taken in comparison to the size of the input instance.

For the dictionary problem the size of the dictionary the number of words in the dictionary that is would be a good indication of the size. And for the x ray problem, we will somehow have to take that x ray and convert it into bits of some kind. So, we could say for example, that the size of the x ray say the number of the if the x ray is has a resolution 1000 by 1000. Then, they could say that the size is a million or something like that.

We often use the phrase problem size also to denote the size of the instance. So, if you say if you hear the phrase problem size it is really talking about the instance of the problem rather than the problem directly. But, that is a term which is very commonly used on the literature.

(Refer Slide Time: 24:37)




They next important term that we need to discuss is algorithm. When I say algorithm, I mean an abstract computational procedure which takes some value or values as input and produces a value or values as output. I use the term abstract in order to denote that an algorithm can be expressed in many ways. So, a program is an expression of an algorithm.

So, the same algorithm might give rise to different programs say in different languages. A program has been concrete and algorithms has been abstract. Of course even for eve for discussing algorithms, we will need to have an ocean of a language. So; however, this notion is not going to be as rigid or as strict as the notion that we have when we discuss programs. When we discuss programs, we have a very well defined very, very strict language which has very, very strict rules for syntax.

We will not be worrying about all of that when we discuss algorithms. We would like to think of algorithms as the idea behind the program. And so long as we are able to convey that idea in as in very clear terms you will happy. So, the basic our goal in this course is going to be description of algorithms. So that human beings can understand, what is being said and we will not worry so much about the precise syntax that is used.

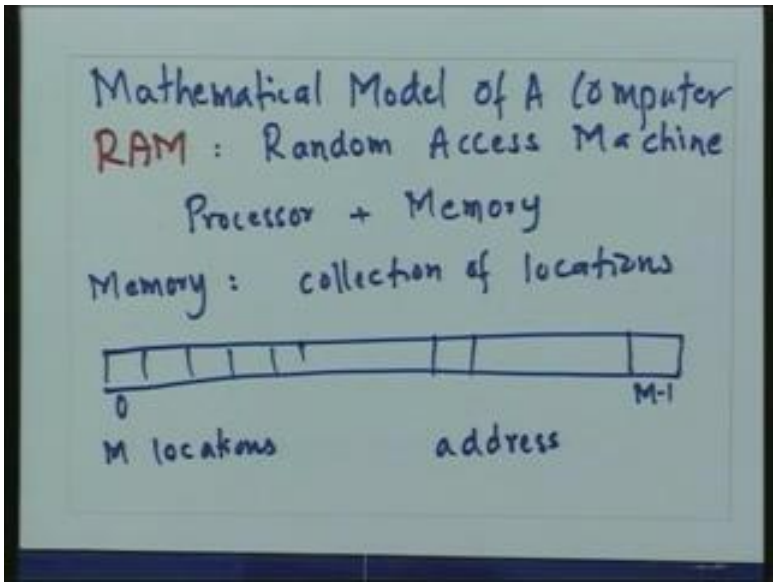
Initially, we will describe algorithms at a fairly great level of detail. As the course progresses, we will abbreviate our descriptions. And it will become clear to somebody who has gone through the course exactly what is being met. The reason for describing

algorithms is of course, one reason is to convey what is the idea. And the other reason why we will be discussing algorithms in this course is of course to evaluate their time.

So, I tell you what an algorithm is. It should be clear to you, what exactly are the operations that I have in mind. And you should be able to write the program, but not only that. It should also be clear to you how that program will execute on a machine. And especially, on the model machine that we are going to talk about and that is going to be another important purpose another important point that we want to keep in mind, when we discuss algorithms.

So, we have to describe algorithms at such a level of detail. That it is fairly easy to analyze how long they will take on our mathematical model. All these issues will become clear, when we describe our mathematical model which we will do right now.

(Refer Slide Time: 28:48)



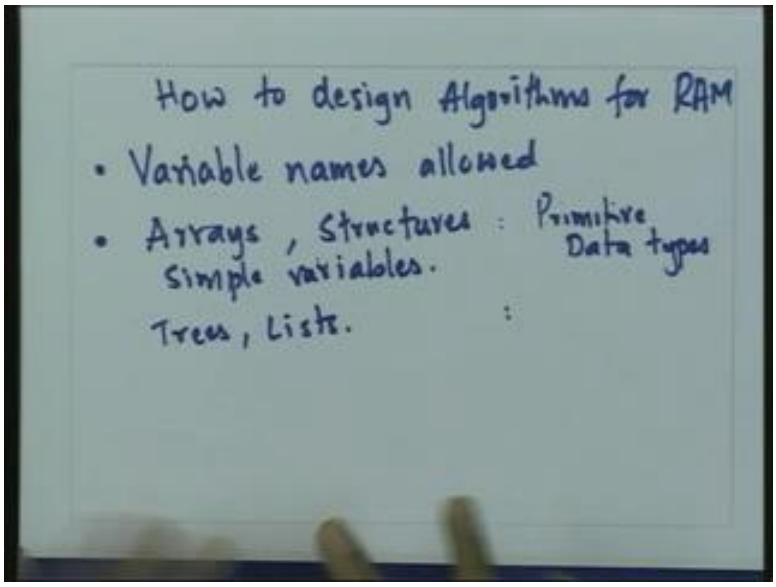

The mathematical model of a computer that we are going to use in this course is called RAM. And RAM stands for random access machine. This is a very simplified computer model. And it only consists of basically consists of two parts. So, there is a processor which will be executing programs. And then, there is going to a memory.

The memory is going to be a correction of locations. And in fact, it is convenient to think of the memory as an array with numbers on it. So, the locations start with a certain say 0

and there might be say m minus 1. The last number could be m minus 1, if there are m locations overall. So, each location has a number which is also called its address.

So, we can refer to locations by assigning by describing the number. Of course that is going to extremely inconvenient in general. And so while writing algorithms, we will want to do something which is more pleasant. And let me start describing, how we I will describe, what exactly, how we are going to refer to the locations.

(Refer Slide Time: 30:58)

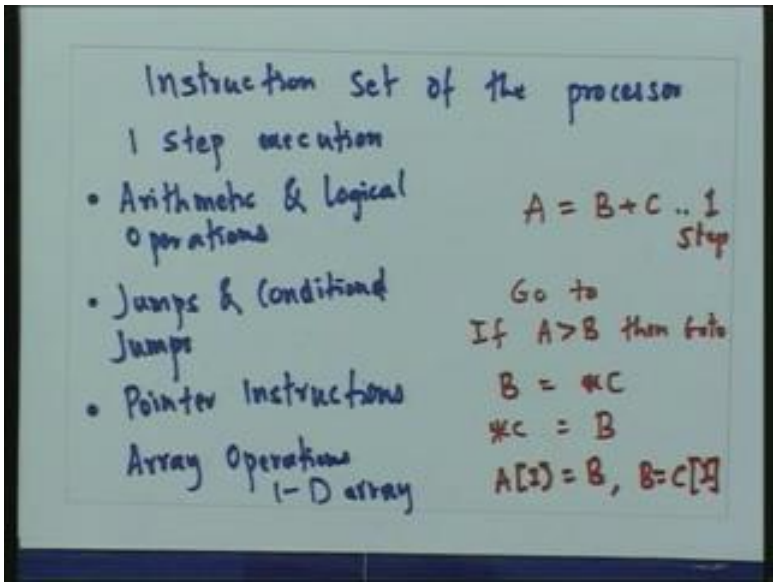



And in fact, as we describe the RAM model I will also be describing how we program the RAM model or how we design algorithms for the RAM model. So, the first thing to notice that although the RAM model contains locations which are addressed by numbers we will in fact allow variable names. So when we describe algorithms, we can say that say the value is contained in this variable a. A certain value is contained in variable rather than a certain value. We stored in the location fifty three or something like that.

In fact, we will allow a variety of data types. Say, we will allow plain, simple plain simple variables. But, will also allow say arrays and will also allow structures. I would like to think of these two as sort of the primitive data types. And of course, let me write down simple variables along with them. In addition of course,, we will allow other things like trees lists and so on as well.

You will be able to build your own data structures as well, but somehow or the other they will have to be built out of these data structures. So, this is as far as the memory is concerned. There will be a memory which will store the program as well. But, we will think of it as being quite separate. So, the program and data do not mix. So, here is again our picture (Refer Slide Time: 28:48) of the RAM model. So, there is a memory and then there is a processor.

(Refer Slide Time: 33:10)




Now, I have to tell you what the processor can do in each step. So, basically this is going to be a description of the instruction set of the processor. So, the processor is going to have a number of instructions and will assume for simplicity that all instructions execute in one step.

There are basically three, four groups of instructions that we will have. So, one group is arithmetic and logical operations. So in this, you will be allowed in your program to say take two locations from memory. Add their contents and deposit them in a third location. Let me, write down how you will actually express this, when you write programs. And do not worry; it is going to quite in a quite friendly. This is going this can be represented in a very friendly pleasant manner.

So, for example, you could say A equals B plus C as a part of your algorithm. And this is going to be one instruction. As we said an instruction is going to be taking two operands

B and C which are stored in two locations, add them up and put them back. So, this will happen in one step.

Then, you will be allowed to have conditions jumps and conditional jumps. And this will also execute in one step. So, correspondingly as a part of your program you will be allowed to write something like go to. This will happen in one step or you will be allowed to write say something like if A greater than B then go to. This will all happen in one step.

Defining our model, we want to keep this definition reasonably simple. You may be wondering at a stage, real computers probably do not look like this. And you are right and we will take that question a little bit later. I would like to make another comment over here. Although, the very second group of instructions that I am talking about concerns go to is this does not suggest. This should not suggest to you in any way that when we design algorithms, we recommend use of go to is far from that.

Algorithms as I said are intended to be read mostly by human beings. And therefore, structured programming presenting the algorithms in a nice readable manner is extremely important. However, when we talk about machines go to is are a very convenient mechanism. And that is the reason, why we have go to is in our instruction set. We will soon come to instructions which are more structured which, but that will be built out of those will be built out of our basic instruction set.

So, they will take several instructions and several cycles of execution. We will come to that very soon. There is a third group of instructions which is important and which I will call as pointer instructions. So, these are simply operations of the form say B equals star C, where I am using C style pointer notation. So, I am going to think of C as a pointer or C itself contains the address.

And I am going to fetch that location whose value is contained that location whose address is contained in C and B will get that value. I can also have a store based on pointers. So say for example, I could write star C is equal to B. And all B's and both of these will also be executed in a single step. All of these algorithmic actions will take just one step.

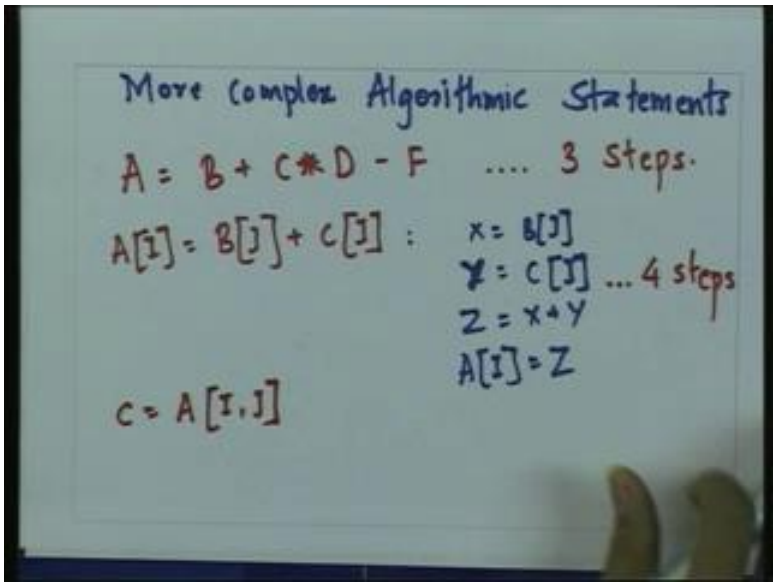
Pointers and arrays are very related and the C language in particular mixes pointers and arrays a lot. And in fact, our machine our random access machine is also going to treat pointers and arrays in a very similar consistent manner. So in fact, in this group itself I will put down array operations and here I mean one dimensional array. So for example, you are allowed to say A of I equals B or B equals C of I. I do not mean this C to be the same as this C. Just C is just some array which you have declared.

Going back to arrays, let me just make one comment about that. So, we said that arrays that our machine will contain arrays and structures will assume the usual C like representation of arrays. So, if an array has size 100, then we will assume that the array is stored in 100 contiguous locations in memory. Similarly, if an array if we have a structure which consists of three components then the structure will be stored in three contiguous locations in the memory.

So coming back, we have a processor and a memory a processor whose basic instruction set I have just described, a memory which consists of a location. I have not told you what a location is really. A location simply is a collection of bits. It has to be a fixed number of bits.

Say does not really matter what number it is, but it has to be fixed once. And for all say it could be a number like 64 which is the number which is used in most modern computers. So, there is a notion of a word and a notion of a word really goes along with this notion of a location.

(Refer Slide Time: 40:38)




So, we looked at the basic instruction set and the basic algorithmic actions that are possible, what I am going to do next is think about more complex algorithmic actions more complex algorithmic statements. So, we said that for example, we allow these instructions, but naturally these should suggest to you some more complex instructions or statements that you would like to have.

For example, we would like to write down a statement which looks like this. So, say A equals B plus C times D minus F, how long does this take. Well our rule is very simple. We will have to break this down into out elementary instructions. So here, we have three operations and therefore, this will take three elementary steps. The three elementary instructions and therefore, this will take three steps.

So, we will allow use of such statements in our algorithm. But, when we count the time we will have to count 3. We will also want to use arrays in such expressions. So for example, we might want to say A of I equals B of I plus C of I. Again, we have to see, how this statement is going to be represented in our basic instruction set. So, here is how this could be represented.

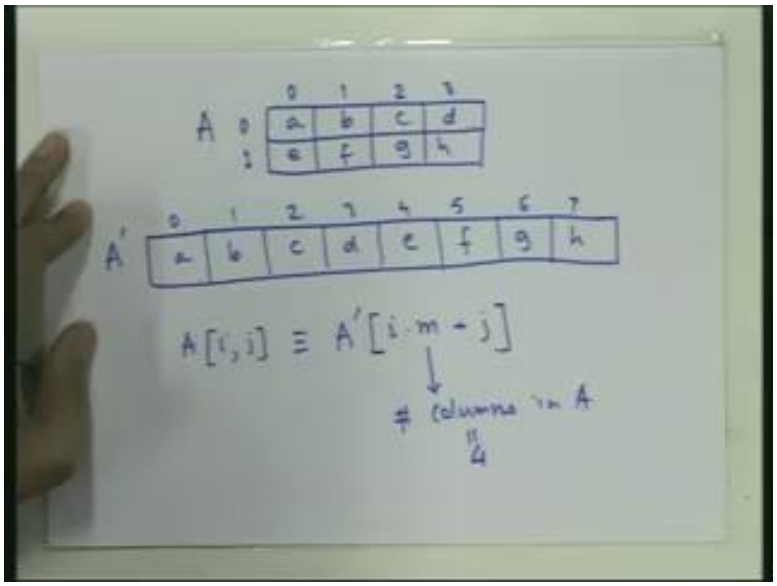
So, first we need to fetch the value of B, the element of B. So, this could get translated to say something like X equals B of I. And this, itself is a primitive statement that we allowed and we said this takes one step. Then, we can similarly fetch C equals sorry say

Y equals C of Y. And this can also again be done in a single step, because this is our basic instruction itself. Then, we could compute Z equals X plus Y.

So now, we have ahead of the two values one of B of I and C of J and the second of C of J. And we have computed their sum and all that remains is that this sum needs to be stored into A of I. So now, we can write A of I equals Z. So, this simple statement well simple which we wrote down very simply in that sense. Really has to be translated into four machine instructions so to say. And therefore, this entire statement will take four steps during execution. We could also have multi dimensional arrays.

So for example, we could have an instruction which looks like C equals A of I comma J. So here, we will have to decide, how two dimensional arrays are stored. But, we will assume that two dimensional arrays are built on top of one dimensional arrays.

(Refer Slide Time: 44:19)



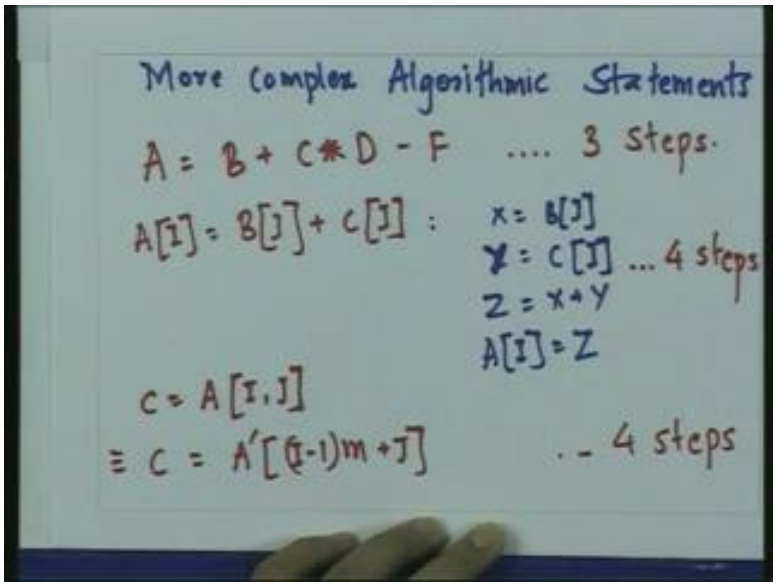

So for example, if we have a two dimensional array which looks like this. So, say this is an array A which has two rows and four columns. Say here at two rows and four columns. Let us say it stores elements a b c d. In the first row, e f g h in the second row. Let us say that the array is indexed c style. So, let us say this is row index 0, row index 1. This is column index 0 column index 1, 2 and 3. Now, on our machine on our RAM, we are going to store this using one dimensional array. We are essentially going to simulate it using the one dimensional array. But of course, the array must have the same number

of elements. And so that is 8 2 3 4 5 6 7 8 and these elements will have to appear in this one dimensional array somehow.

Let us say they are stored row wise. Many possibilities are there, but we are just picking one. So, a b c d e f g h, so essentially every time you want to access this two dimensional array, we will be accessing some element of this one dimensional array. And by the way remember that the index set of this is 0 1 2 3 4 5 6, how do we know, which element of this array we access. In order to access a particular element of array of this array well there is a very simple correspondence.

So, A of i j if you want to access the i, jth element. So, row i column j, then that corresponds to an element of A prime which element well it is simply i times m plus j where m is the number of columns in a which in this case is 4. So, wherever we see a i j we really should be reading it as this as far as the problem of accounting how much time it takes.

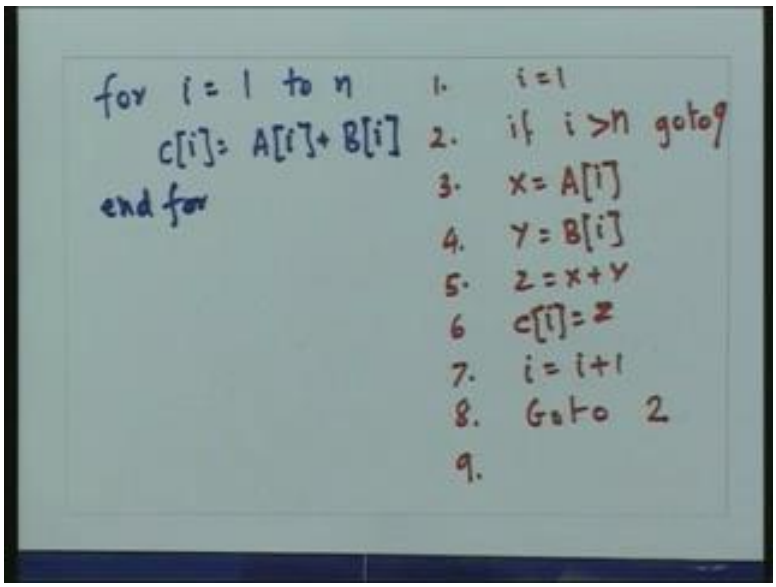
(Refer Slide Time: 46:54)




So, let us do that. So, C of A I J well we should really be thinking of as C equals A prime of I minus 1 times m plus J. But once, we think of this statement in this manner. Then, estimating the time taken for it is fairly straight forward because now we know that we have to do one subtraction. We have to subtract 1 from I.

Then, we have to do one multiplication in the multiplication with m. Then, we have to do one addition and then we have to do a plain indirect axis. So, this whole thing will take 4 steps. Let us, now turn to some structured computing statements.

(Refer Slide Time: 47:54)



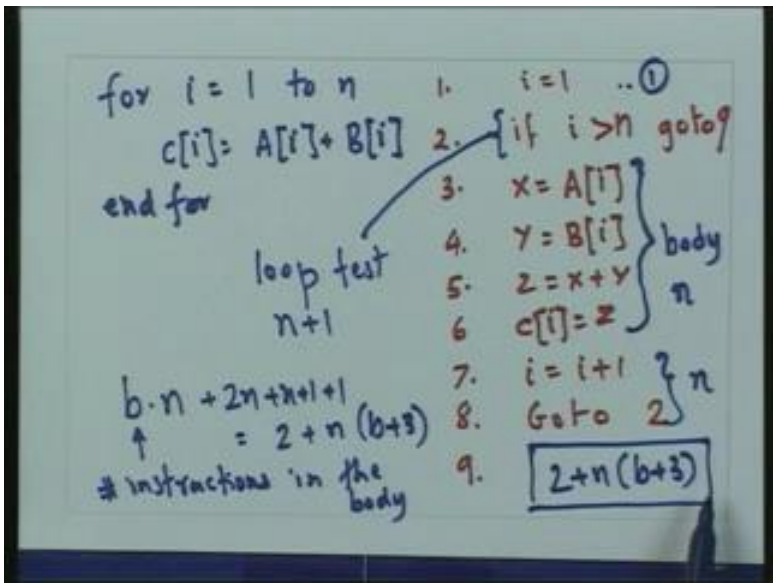
Let us take a loop. So, let us say we have for loop. For i equals 1 to n, let us say we do something like C of i equals A of i plus B of i. So as I said, we have to translate these instructions into our basic constructions and here is a possible translation. So, we are going to have to start off by initializing. So, we will write i equal 1. That is how the initialization step will be.

Then, we have to have the loop test. So, we will write this as if i greater than 1 greater than n, then go to end of loop. Then, we will write something like fetching the ith element of A. So, we write something like X equals A of i. Then, we will write Y equals B of i and then we will write say Z equals X plus Y in the manner that we just discussed and now we have to store it back.

So, we will write C of i equals Z. At this point we have to step through the loop. So, we will write i equal i plus 1. And now we will just go back. So, we will go to let me number these statements 1 2 3 4 5 6 7 8. And so we are going to go back to 2 and this should jump out of loop. So, this has to go to 9. So, this is going to be a translation of this.

So, although in our algorithms, we will write this for statement and just for completeness may be we will have an end for as well. But, as far as the purpose of accounting is concerned, this is the time, this is the code that we should be considering. So, when we want to analyze the time taken, this is the code that is going to be of interest.

(Refer Slide Time: 50:44)

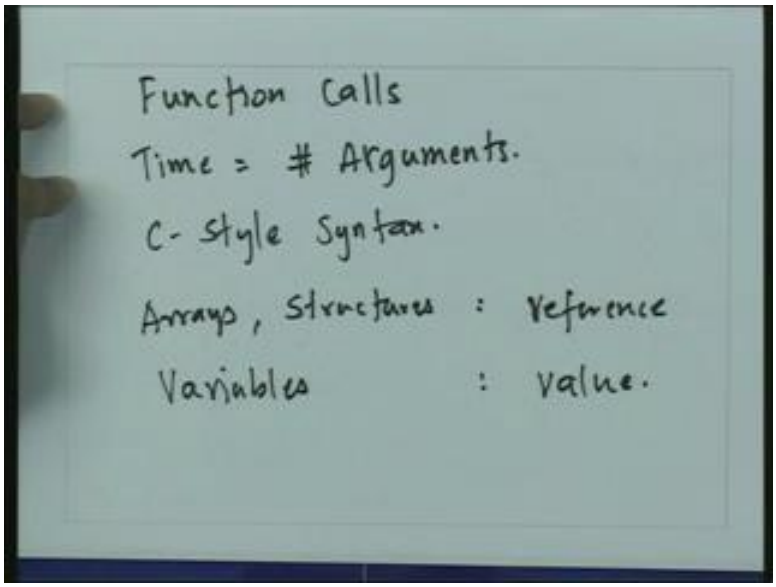


So, let us try and analyze this code. So, the analysis is going to be reasonably straight forward. We are just going to go over each statement and we have to see how many times it is going to be executed. So, let us try statement number, how many times will statement number one we executed. Well this will just be executed 1 time.

This part A of i to this part is what might be thought of as the body of the loop. This will be executed n times. This loop counting and this jump back will also be executed n times. And this loop test let me write that down here. This will be executed once for each iteration. But, it will also be executed one more time. Because, that is when the machine is going to determine that we need to exit. So in fact, this statement will be executed n plus 1 time.

So, the total number of steps taken for all of this is going to be b times n, where b is the number of instructions in the body. Plus these 2 steps plus these 2 n steps plus 2 n plus these n plus 1 plus this extra 1. So, this is going to be nothing but 2 plus n times b plus 3. So, this is our final answer. Let me write that down in big letters over here 2 plus n times b plus 3. This is going to the number of steps needed to execute this loop. Of course for this loop b has the value 4 and therefore, that is going to be the time.

(Refer Slide Time: 52:48)

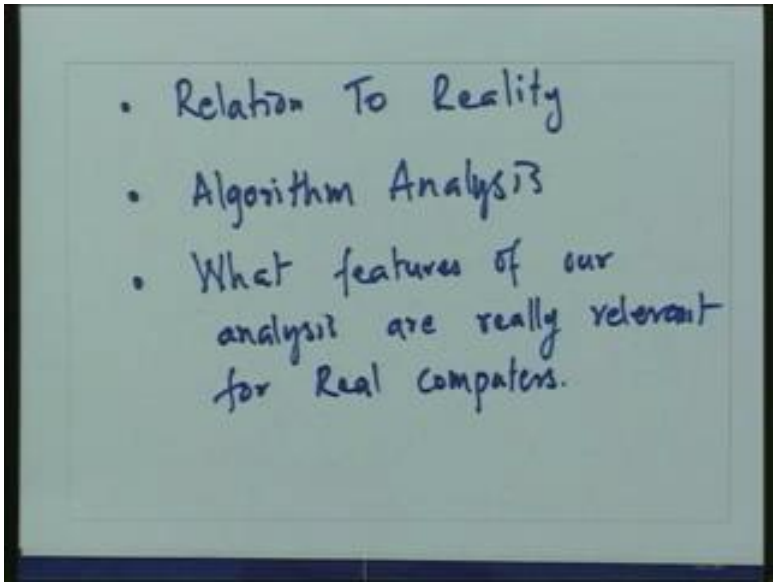



We will also allow functions calls in our programs. And when we will assume for simplicity that the number of steps is needed for the function calls are going to be the number of arguments passed. So, time equal to number of arguments. Of course I should

say something about the syntax of the function calls. And the syntax is going to be pretty much like the C language, what I mean by this, is that arrays and structures will be passed by reference which means that you can modify them in the called procedure.

And the modifications will be seen by the calling procedure, whereas variables will be passed by value. So, this basically concludes the description of our random access machine. There are a number of issues that we need to consider and I will mention some of them.

(Refer Slide Time: 54:17)




Now and these have to do with how the how this machine relates to reality are real computers like this or are they different. Then, we have to ask questions about we will want to take algorithms and see their complete analysis. And finally, we will want to say what part of our analysis is really interesting, when we consider real computers, what features of our analysis are really relevant for real computers. So this, we will do in the next lectures.