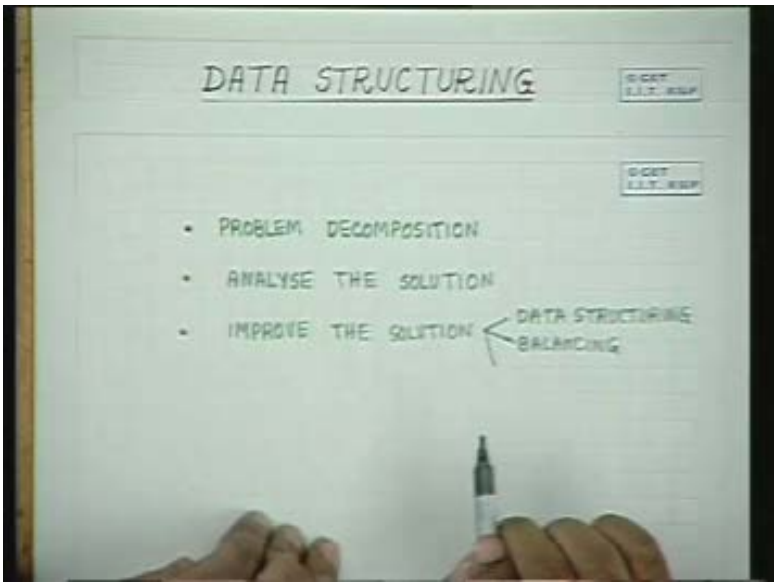
Programming and Data Structure
Dr.P.P.Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 21
DATA Structuring

The topic of today's lecture is data structuring. Till now we have concentrated more on the efforts, our efforts have more been concentrated towards problem solving techniques and as we said before that solving a problem in a computer consists of two main parts, one is called the problem solving or what we call the problem decomposition aspect and once we know how to decompose the problem and try to generate an initial solution and then we will analyze the problem.

The next aspect is to analyze the solution to the problem and we have seen very quickly what is meant by asymptotic analysis which is one thing which we will do to analyze that. Second is how to improve the solution and obtain good solutions. For example in our analysis of the merge sort algorithm we saw that if we split it in the middle, we will get a better algorithm than whether we split it on other ways. There are various other aspects how to improve the solution, one of which is organization of data.

(Refer Slide Time 03:10)



So before studying the techniques of various ways in which we can improve the solution, we will also have to study. One of them is data structuring, one of them we saw is let us say let us call it balancing the split. It is called balancing, where will you split if you decompose into two parts or into n parts then where will you split the problem. The balance is very important that is called the problem of balancing. There are various other issues which we shall come to later on and then we shall see how to combine all of them in more complex situations. So we have just done a bit of problem decomposition, we

have seen how to analyze the solutions asymptotically either by doing a loop analysis in terms of analyzing how many loops will be there or in terms of solving a recurrence equation.

The other aspect is the organization, structuring, storage and manipulation of the data elements that we have in our program and then there are other techniques of improving a solution which we shall come to later on but data structuring is also a very important consideration of algorithm design and this is something which we shall concentrate independently by choosing problems which are typically data structuring problems themselves, so that the algorithm design and the problem decomposition aspect does not come immediately. We shall just choose problems which are just pure data structuring problems and see how to organize and how to define such data structures and in order to have such data structures what are the techniques we may need to employ and what are the standard techniques which are known. And it is there that all dynamic allocation and the structure of all linked lists and the way we have manipulated them will come very handy.

(Refer Slide Time 05:25)

So we shall concentrate for the next few classes on the problem of data structuring and then we shall go back using our knowledge of data structuring, balancing and one or two more techniques to see how we can completely solve given problems so that will be the approach that we shall take till the rest of the course. So what is data structuring which is today's topic. The data structuring consists of two things, one is the organization of the data elements and two the efficient procedures for data manipulation. So how we will organize the data elements, how we will store the data elements and how we will efficiently write by manipulation routines on the data elements and both are interlinked and we shall first see some examples of problems which are pure data structuring problems alright.

And every programming language will provide you some data structures or data types which are in built with certain operations and facility is to create other data structures. So first we will see several examples of data structuring problems themselves from simple ones to fairly complex ones. The first one is something which we have seen before. We have to represent complex numbers and in complex numbers we have to add, subtract, read and write them and we have seen how we will represent the complex number as a structure with two parts conceptually it is one single structure with two parts, the real part and the imaginary part and we define operations read, write, add and subtract on them.

Now when we provide a data structure we will not allow external user who will use this data structure to manipulate the internal data elements of that data structure directly. The only operations or view of the data structure available to anybody who is using the data structure is the procedures or functions which operate on that data structure. Now programming language will provide you automatically several such data structures in the name of data types. For example even an array is a structure provided by the program in which you can declare and define it and the operation provided is direct access.

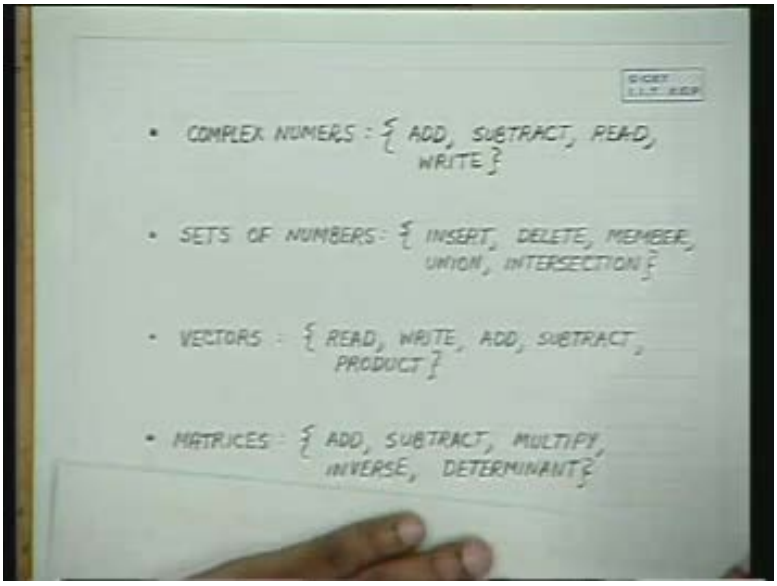
Suppose you define a two dimensional array then a i j when you write with instantiated values of i and j will give you is actually an operation on that data structure. We are not allowed to change the address of the array, we are not allowed to move that array in the memory and store it in another location of memory, so some things we are not allowed to do, the system automatically will store it in some place and provide you certain operations. So a programming language like C will provide you certain mechanisms to define data structures and will give you certain predefined data types like a string data type and it has given you a whole set of libraries to operate on that string compare, this that everything. Similarly for example you could have created a data structure in a language and allowed the user that this is the data structure that I am giving you and these are the functions that you can use on that data structure, alright.

So if you provide that, you have given a data structure as a data type in a programming language. So a data structure is of two types, is of two categories, one is its declaration. That is this is a complex, this is a data structure for the complex number with these operations. The other is its instance that is I may declare two complex numbers of the same type and then I may operate on them alright. So one is like a type declaration which is called the data structure definition, the other is like a variable declaration which is called the data structure instantiation.

So for different instance, the operations will be the same but the data element inside the data structure will be different. So you can define a data structure for complex numbers and define some 20 complex numbers and then add and subtract and read them but you cannot manipulate when you declare this data structure, you will not allow manipulation of this in terms of, for example the real part cannot be suddenly changed or the imaginary part cannot be suddenly changed because these are not provided. Second, sets of numbers and the operations that you define, insert an element into a set, delete an element from a set, find out whether an element is a member of a set, take two sets and perform union of them and return a set, take two sets and perform an intersection of them and return a set.

You can implement this by some means given. So this, the data inside and these operations, the organization of the data in these operations will find what is your data structure. Now for the same data structure you can have 3 or 4 types of implementation of that data structure. And what we will be looking for is an efficient implementation of a data structure.

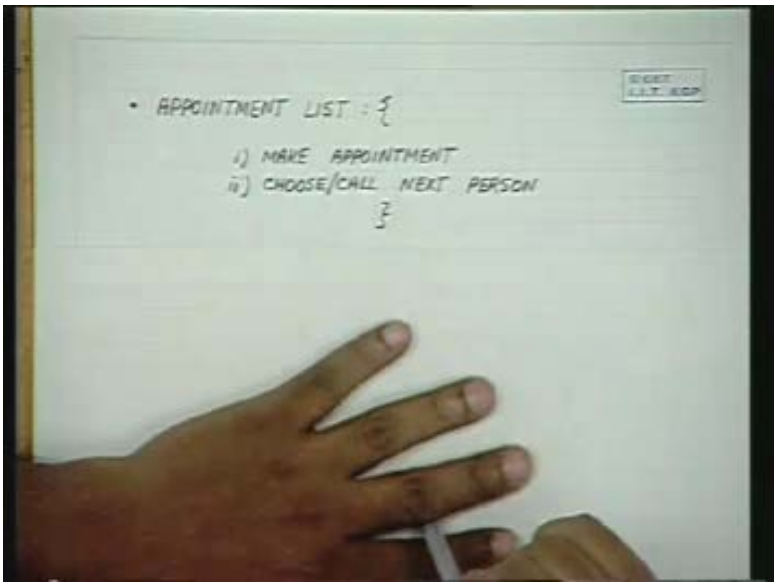
(Refer Slide Time 10:51)



Let's see another one, vectors. You can define vectors and each vector may be parameterized by the size of that vector then read a vector, write a vector, add two vectors, subtract two vectors, take a dot product of two vectors. So similarly you can define a data structure for this and you can imagine how you will implement the software. Matrices, you can define a data structure called a matrix which is parameterized by two parameters depending on the rows and the columns of the matrix and then you can define operations for adding two matrices, subtracting two matrices, multiplication of two matrices, inversion of a matrix, finding the determinant of a matrix and so many other. So these matrices will be operated by the operators which are defined.

So these are, suppose I give you such problems, declare and use complex numbers, define a library for complex numbers that is you are actually left to defining the data in a particular way and writing out these manipulation procedures in that data in an efficient manner. Let us see some examples which are slightly different but they are also in the same mode as data structure. Let us look at doctors, a simple doctor's appointment list. And in this appointment list you will be asked, say somebody will say request for an appointment and secretary of the doctor will make that appointment and by normal law of making an appointment you will now be the last person that is you will be the last person who is in the normal point. And what you will, that operator will do as when the person, the doctor has seen already one person? They will choose or call the next person, alright.

(Refer Slide Time 11:57)



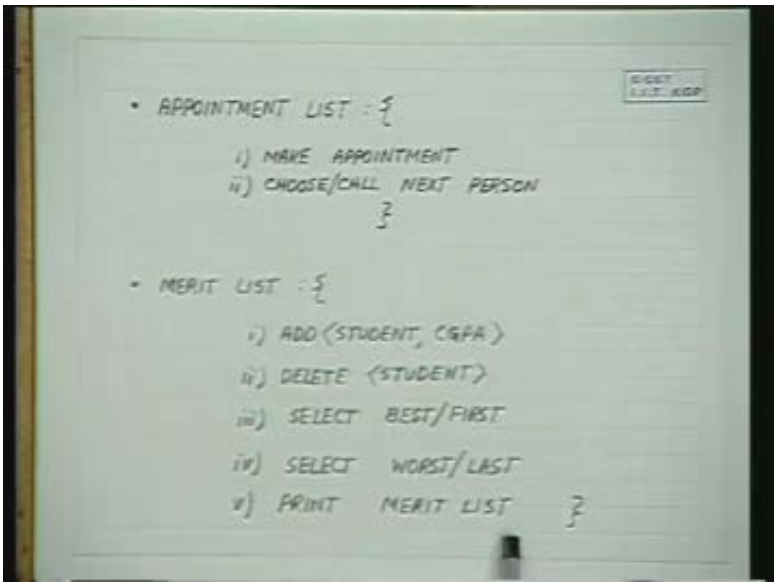
So these are the two operations, over and above this you can define cancellation of appointments. Somebody may make an appointment and then ask for cancellation, you may make it more complex and say he will make an appointment and ask for the time slot that I am can you give me this slot. So they will look and see whether that slot is available for that person. Here we have made a simple appointment; there may be various ways in which you can look at this appointment problem. The simplest is like a queue, you stand in a queue and you just go to the doctor and put your chit there, so your number 9. And when number 6 goes away, number 7 is called and then you can finally say somebody may cancel the appointment, so number 12 may get cancelled.

So after 11, 13 should be called that's the way you can make an cancel and appointment or in a more complex situation you can ask appointment for a particular slot and you will be given the time slot to come for. That is you come between 10 and 11 and between this time your appointment will be given. And it is in somebody may ask for time slot between 10 and 12 and then choose or call the next person will be more complex because many people will be overlapping between 10 and 11 and you may have to form a queue out there or you may have to make a certain prioritized operation. So it may become more and more complex depending on the situation but this is also a problem of organization of the data elements. Here the data elements will be the people who will request for the appointment and you have to organize the data in such a way so that you can make an appointment appropriately and select the next person appropriately. So this is the job of the computer as if in a menu these two operations are there and these are the two operations which have to be done or they may be called from another function like this.

Now let us look at it. You can define this as a data structure and in another program where you require an appointment list and you will be making a program which will be doing an appointment list. You can say I will declare an appointment list L_1 and

appointment list L_2 and appointment list L_3 for three doctors in a poly clinic all right and you will declare three such appointment lists and you will manipulate them by making an appropriate call to this appointment list. So, there may be a main menu at the secretary of the poly clinic where 3 or fo4ur appointment lists for three doctors, so the program will use this data structure to define a higher level menu which will make an appointment to a given doctor at any particular point of time. So this is how you will grow up and you will develop the structures. So look at it as if in a very very high level programming this is the data type, it is a predefined data type. In a very high language you can provide to a user who is using a menu or who is writing a program this is his data type.

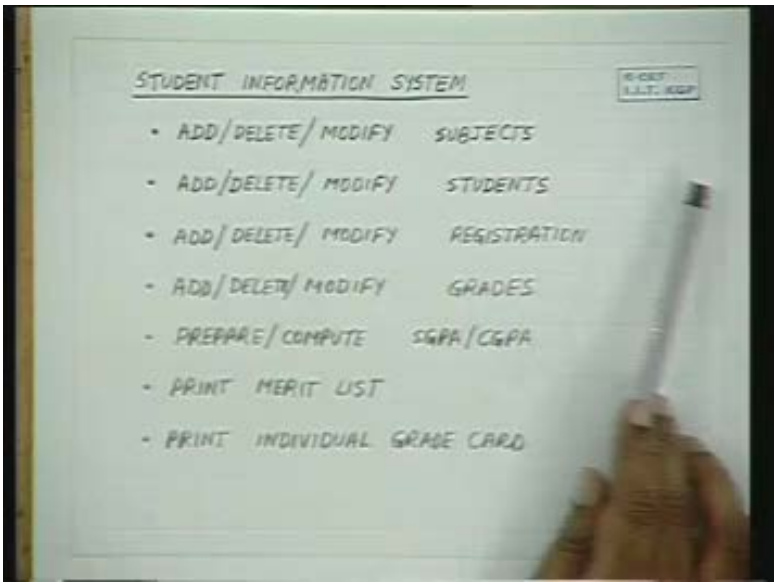
(Refer Slide Time 16:18)



You can define appointment list app list x, x_1 x_2 x_3 just like you declare int i_1 i_2 i_3. So this is how the problem of data structuring scales up slowly. Let's come to a slightly more interesting problem of a merit list. Here in our student information system we are to prepare a merit list and here you will add a student in terms of the student means the student roll number and the cgpa, may be you can delete a student, may be you can select the best of a student and you can select the worst or last student that is you will print, find out the best at any and you can print the complete merit list at a particular point of time. So this is how, these are the operations available on this problem.

We are now discussing pure problems where the problem is only of data structuring all right. And each one of them will have to be implemented by an algorithm and we will have to come to an algorithm which will implement this. But these algorithms will be related to the data structure defined, so we are more or less on a pure data structuring mode rather than an absolute problem decomposition mode. And finally maybe we can have a complete academic section information which will have add, delete or modify all the subjects, these are the subjects, subject name, subject number offered by this department etc etc.

(Refer Slide Time 17:05)

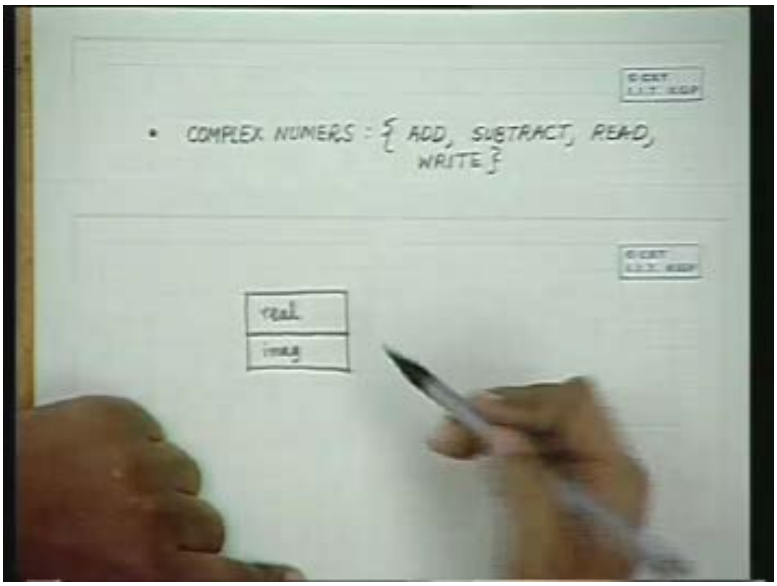


Student information, these are the students, this department, etc, add, delete, modify. Registration information, grade information, computation information of cgpa, sgpa, merit list preparation, failure list preparation, individual grade card preparation, printing out individual grade cards and then when you register for a semester you may have to check whether he has not registered for the same subject twice that he has cleared, all the rules of the institute must be maintained. So this is another problem which leads to a pure data structuring type of a problem and when this is in massive amount of data, there are two aspects of data structuring. We are discussing data structuring which we will write ordinary programs for and in which the data will be stored in the memory all right. But if the data is to be stored in the files because when large amount of data is there, you cannot store all the data in the memory and you will have to store it in a file and you have to do file operations. So the complexity issues will be slightly different from the memory operations, depending on the type of file, what file you are using, how you are accessing the file. So we will come slowly to all these issues but before we come to them, we should start from beginning and see what are the issues in data structure manipulation.

So let us come back to our first problem which we have done before and let us see what we have here, complex numbers with operations add, subtract, read and write and we saw that converting to a programming language is the last thing. It is the conceptual design of the data structure which is important and we will see how we can develop a language in diagrams or what ever to develop this complex structure. Now here we have these operations. So what we decided was in the previous case, in that we have one structure that is the data structure. That is as we said before there will be two parts, the data organization and its manipulation will consist of two, the real part and the imaginary part. And this is what we will define for any instance of the data. For different instances these values will be different, different actual allocation will be done but the structure or the type will be the same and the functions add locate two numbers and add them, subtract this is what we have written before. And let us see, could we have done anything better? I

don’t think so, because we have done just two addition operations for these two things and for read and write, so there is the complexity of both of them are all in constant time, all operations add is done in constant time, subtraction can be done in constant time, read is done in a constant time for a fixed size.

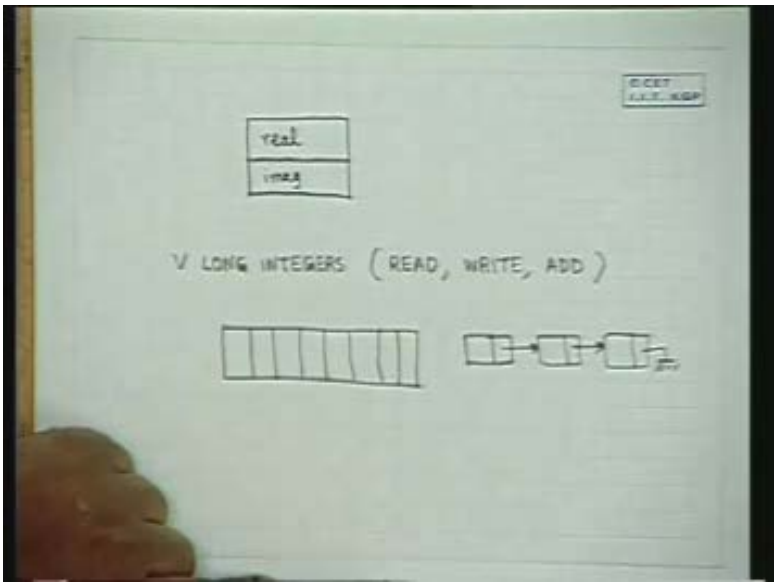
(Refer Slide Time 19:55)



Obviously if you are asked to read complex numbers of length some 20,000 then you have to think of it in a different way. For example the problem of adding two very large numbers which you cannot declare in your programming language is itself a data structuring problem. Given two numbers, you will have to add two numbers of arbitrary sizes. Then how will you read, them how will you store them? Here we have fixed size, so all of them are constant time and therefore this is a reasonably good way to implement it.

On the other hand if we gave you that you have to work on integer long, very long integers and you will have to read them, write them and say add them. Then how would you implement such very long integers? For example very long means, you don’t know the size, you will be just getting it and it will be ending with a dollar, you will be given a number, huge number it will be ending with a dollar. So how would you organize it, how would you store it? One way could be, you could break it up into fixed size numbers. That is you know that a number can be at most say 10 digits in your representation and you would break it up into fixed size numbers. But you still do not know how long it will be. So what would you do? Either you would, when you read this number you would read it into character string, you would read it into a character string or somebody must have told you that is you will read character by character, count how many are there and after you count how many are there, you know what is the size all right. And after you know what is the size, you know how many integers you will require to store that size. So you can malloc that many integers, you can simply malloc that many integers.

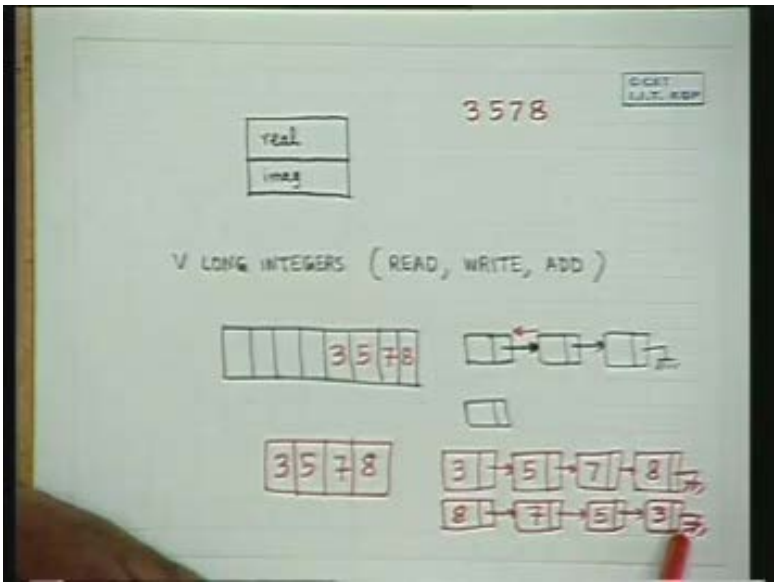
(Refer Slide Time 22:58)



So after reading the number of characters, you can malloc or you can dynamically allocate that many fixed size integers to store your very long integer. The other option could be, you could have done it as a linked list. That is after you will read in some numbers and you will just form a linked list of these numbers. Now given two such very long integers, writing them is no problem, both of them will require one will require to access the array and the other will require to access the linked list and the time of reading and writing will be proportional to the length of the number of digits in that integer which is quite fine because you cannot read it in less than that time.

Addition, how would you do addition? In a linked list given two linked lists, suppose this is the least significant, how would you, would you make the point as this way or would you make the point as this way or would you make it both. Can you tell me? Suppose the number is 3 5 7 8 and suppose your digit allows you to store only one integer at a time, then in the array representation it would be 8 7 5 and 3 and this is you would have declared an array of size 4 and stored it like this. In a linked list, you would have declared 4 elements and how you would have stored it? Suppose you define the linked list like this, would you store it like this, 3 5 7 8? The other alternative 8 7 5 3, these are the two alternatives which we can use to store it or we can put in a linked list which has got both front and back point as which makes it optional. So could you tell me which one would we choose here, the first one or the second one? The second one will be chosen.

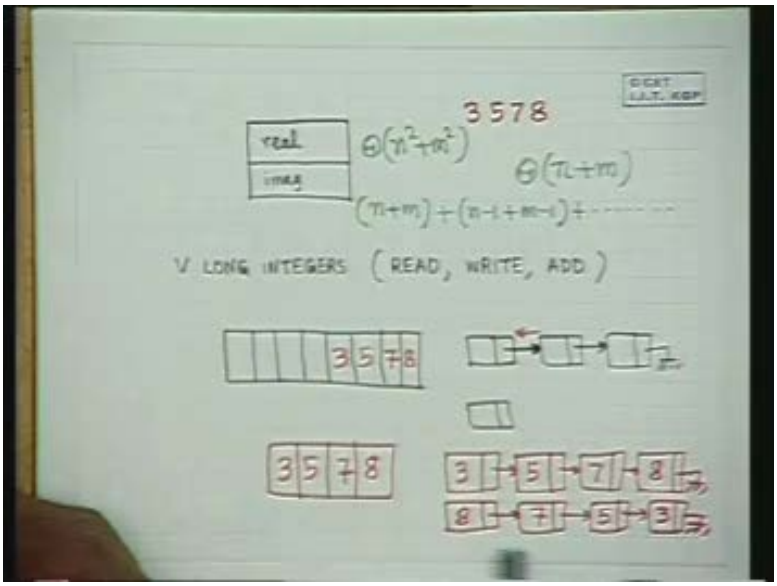
(Refer Slide Time 25:00)



Now why, why will the second one be chosen when compared to the first one, why the first one will not be chosen, why? What is the intuitive argument that you are having to choose the second one and not the first one? That is because of the, see the read and the write operation are unaffected whether you store it this way or this way because you can, as we have seen in linked list operations you read an element and you can prepare the list this way or prepare the list that way, we have seen programs to do that. But to do the add operation, if two numbers are stored with the most significant bit and pointing this way, addition inherently you will have to add it this way. So every time you may, if you have pointers in this direction, when you have added this and the next one then you will not be able to go back here. We will again have to start from the beginning and move. So moving at every point of time will be of complexity of order n where n is the number of integers. Moving to the least significant bit will be n then it will be n minus 1 then it will be n minus 2 then it will be n minus 3.

And your addition operation which normally in this case you would have added these two and moved here, added these two and moved here, added these two and moved here. So very quickly if you analyze, if you store it in the second form, your addition operation will be order n plus m where n is the size of one number and m is the size of other number. On the other hand if you store it with the pointers as pointed out here then what will be your complexity for each operation? For the first operation you will require n plus m movements, for the second one n minus 1 plus m minus 1, for the third one we added like this and you will end up with order of n square plus m square. This is actually you can say it's this order itself.

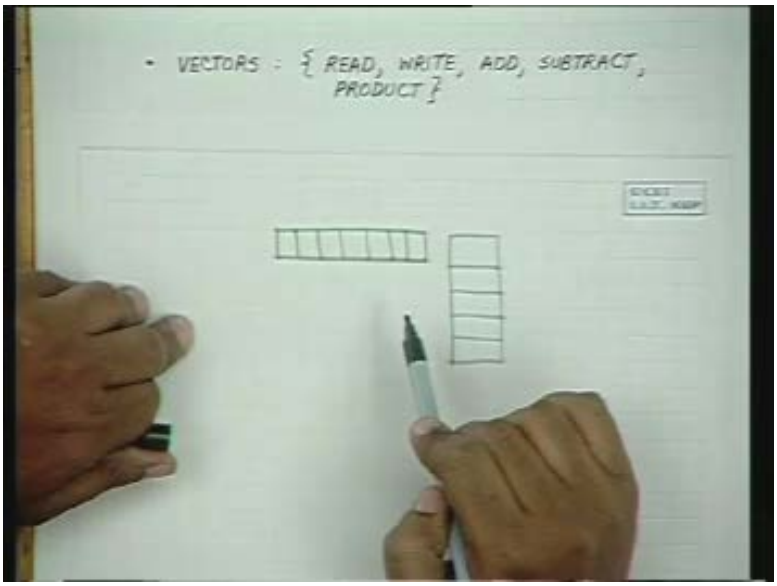
(Refer Slide Time 27:22)



So we have seen an example where the manipulation where the storage of data itself, even simple things like which point, how the order in which it will be done has to depend on the efficiency of the algorithm which will implement. That is why I said, the efficient manipulation algorithm and the storage of the data will go hand in hand. These are very simple examples. Examples will be very complex but still is it better than this one? If you store it as two such blocks of arrays then you will have to know the size of the array obviously and then you can add them this way. So this one will also take order n time to add, isn't it? This one will also take order in time because you will add two numbers like this then take the carry, add two numbers like this, take the carry and instead of being a single digit it may be a big number itself.

It may not be a single digit; it may be a big number itself. You will just use your maximum size available to you in your current program. Now how do you compare? So this one is out, how do you compare this with this, can you tell me which is better? In terms of addition operation functionality, so in terms of time both are order n but can you tell me which one is better? In terms of storage space this will be better than this because this uses an additional pointer and this will just allocate and block of array as memory. So when time is the same as we said before, space becomes an important consideration. So this is how we will decide on the data structure and once we decide on the data structure, we will decide on the, our final algorithms to implement it. So here we will get order n algorithms to read, to write and to add two numbers, two very very long numbers. And obviously you cannot, you can always show that you cannot do in less than order n time, otherwise you will be missing out some integer in the reading or the writing or in the addition itself.

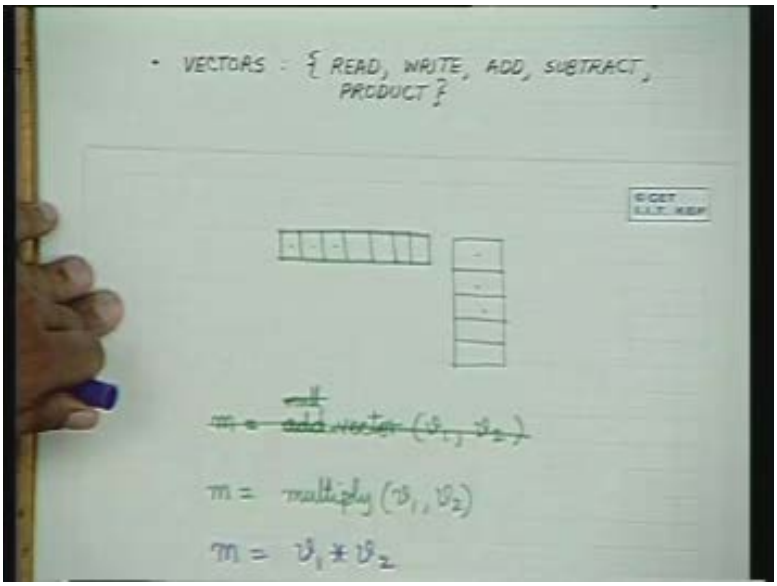
(Refer Slide Time 31:29)



So let's come to the vector problem. Can you see? So, how will you represent vectors? How will you represent a vector? Let us see the alternatives. The size of the vector will be fixed for every vector. Now this vector is say a vector of integers, it's a vector of numbers say it can be a vector of complex numbers as well where each element of that vector will be implemented as a complex number and the addition of the elements will be implemented by complex number addition. So that way we can develop it hierarchically but let us see what are the alternatives. Here again if you know the size, you can define a vector as a data, as a block of an array by dynamically allocating that size.

Once somebody asks you to declare a vector of a particular size, you can dynamically allocate that size as an array and then you can do the read and store it, the write print it, add two vectors in linear time, subtract two vectors, take the product. That is also how much time will take the product of two vectors be, how much time, what will be the complexity of taking the dot product of two vectors? This with this plus this with this plus this with this order n times, isn't it? It will be order n time. On the other hand if you have the cross product then it would return, what? A cross product of two vectors would return a matrix. So if you define cross product of two vectors, you would also have to define a matrix and you would have to write m is equal to add vector v_1 v_2 where v_1 and v_2 are declared as vectors, add vector operates on these two and m is of type matrix. And this is how you would just simply write it, sorry multiply, I am sorry this will be, cut it out, m is equal to multiply v_1 v_2. Now lets the good fun of what is called something called polymorphism, you would simply write m is equal to v_1 star v_2.

(Refer Slide Time 32:57)



Now the system would find out that this is of type vector, this is of type vector, it will realize that this multiplication must be vector multiplication. When you generate the code, the translator will itself know what to do. When you write v_3 is equal to v_1 plus v_2, it will do vector addition because it knows vectors have been declared. So automatically the function, the appropriate function will be called knowing what data you have defined. So this will make your programming, your style of programming, your technique of programming much more simpler but you will have to implement all these data structures in an efficient way. So this is the one approach which data structure and programming takes to take programming out at the highest level and for that you must know what is data structuring because data structuring will be required even at the highest level.

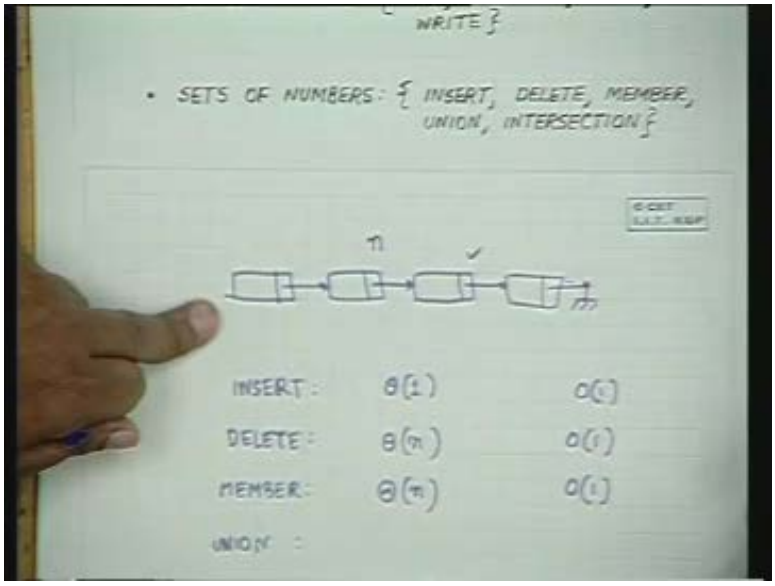
Once you are provided with matrix multiplication operation, you would like to do something more complex and define it as a data structure. On the other hand if these vectors were bits 1 0 1 0 1 0 then may be you wouldn't represent it as an array, you would represent it may be as one integer in a bit format and if a language allows you to do bitwise ANDing and ORing, it would be useful for you to do several other operations may be AND OR because it is a bit vector, bit vectors usually represent some other types of information. So bit vectors can be used to implement such things.

Coming to sets, sets can be implemented by bit vectors. Sets of elements can be, if you know that there are fixed size of elements, there is a maximum limit on the number of elements and you know the universal sets. So let's see how sets will be represented. Suppose you know the universal set, so if you know the universal set then you can represent any defined set. Suppose a universal set is between say 1 to 50 then you can represent it by a vector of size 15, insert will be making that point one. Insert an element i will make the ith point one, delete an element will meet the ith point zero, member will check if that ith point is one, union will do ORing, bitwise ORing of the two. You know the OR operation, bitwise OR operation, intersection will do the bitwise AND operation.

On the other hand if you do not know the universal set then all you have left with this is to make a linked list and store them up. That is you will store these numbers as a linked list because now you have got an insert and delete which dynamically changes the size of your set. In any set the size of the set will be changed dynamically. Now once the size of the set changes dynamically, you cannot malloc a fixed size. What we could do in the case of long integers where we knew the size of the integer, here because the new element may be inserted at any point of time and the element may be deleted. What we can do is we can implement it by a linked list.

We cannot malloc a fixed size and implement it. Are you getting the point? So what will the insert operation, suppose we just make a linked list, insert operation how much time will it take? Constant time, you can insert in the beginning, you can insert in the end wherever you want. Delete operation will take, so insert operation will take order one time. Delete operation will take, if you give a number and asked to delete that number, how much will it take? You have to find out the number in the list and then delete it. You will have to find out the number in the list and then delete it. So in the worst case, if there are n elements it will take order n time. Member will take, member will take how much time? Order of n. On the other hand, if you knew the universal set and you represented it as an array of binary numbers that is as bit vectors, insert would take order one time, delete would take order one time, member would take order one time, just accessing that element in that array. Isn't it?

(Refer Slide Time 38:48)

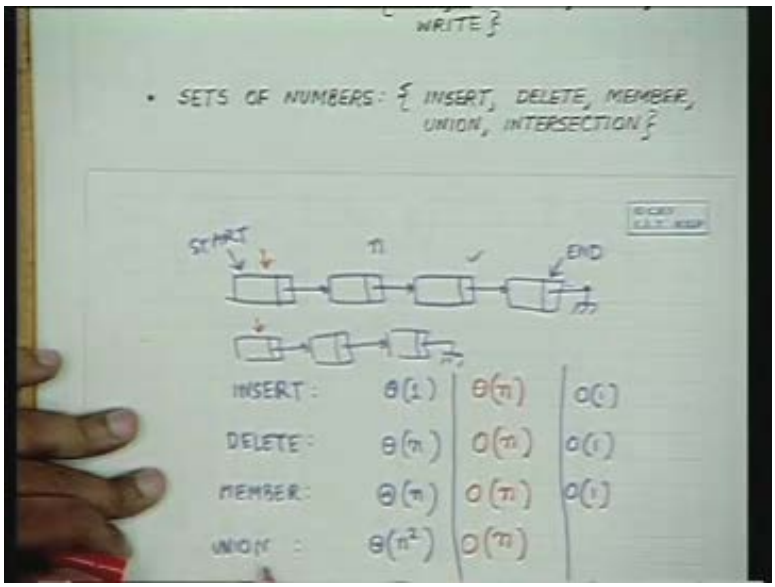


How much time will union take in this case and intersection, how will you find union? Remember if they were disjoined sets, union could be done in constant time. Can you tell me, just make the tail of the list just join the end of this list to the beginning of that list. If they were disjoined sets that is you were told that they were disjoined sets then you would have to maintain in every list a start and the end. So your data structure would be a linked list like this because a start and an end pointer. And somebody tells you to do

union of disjoined elements then you would make the end of one point to the start of the other, it does not depend which one you are doing to which one. But on the other hand if they do not contain disjoined elements then you are in trouble then you will have to scan one element, see whether it is present in the other list. If it is not present, only then you can put it on the end of the list or you can create a third list and put it all right. So this is what you can do but how much time will this take? Order n into n which is n square. It will take n into n where n is the size of one element size of the other, so it will take order n square. So is there anything that you can do about it, is there any improvement that you can make about it, is there any way in which you can organize the data better? Can you come up with a suggestion?

I will give you a hint. This was an arbitrary list, if I give you 5 3 4 7, you would have written 5 3 4 7. But suppose during insertion you sorted it, during insertion you kept it in sorted order like you do in insertion sort then insert would be order n. Let's see how it changes? Delete would be order n, member would be order n, largest smallest is order one but member is order n, largest smallest is order one. If you wanted to find largest smallest here both would be n but here largest smallest is order one but this is order n. What about union? N, you remember merge sort. The merge routine of merge sort, you remember the merge routine of merge sort? You start with one pointer here and one pointer here and if this is less than this you put this. If it is equal to this only put one of them then that is the only change that you have to make. Are you getting my point? So then it will become order n plus m which is the same as order 2 m which is order n. So by sorting the list, you have increased your complexity here but you have brought one order n square to order n.

(Refer Slide Time 42:02)



So now you will think about how many times you have to do this compared to how many times you have to do this. See if the number of times you have to do this is more or less proportional to the number of times you have to do this then you would choose this one

because the total complexity would be sigma$_1$ into order n plus that is the number of times you have to do this plus sigma$_2$ into this plus sigma$_3$, a weighted sum. And now if this one to this one, the relation is not square that is only if this one is to be done more than square of the number of times this one is to be done then this is better in the worst case otherwise this is better. So now I hope you are getting the idea what is meant by data structuring, how to organize the data elements, in exactly what form the data will be stored whether it will be sorted or unsorted this that everything and each and every routine and what is the complexity of each and every operation. If you can do union, you can do intersection is nearly the same.

So, in this particular case if all of them take equal amount, they can come with equal probability then for the time being we would choose this one. May be we can see later on whether we can improve on this one or not but for the time being this is what we have. So this is the essence of data structure and we shall come to other examples and see slowly slowly what we can do about different problems and whether we can define other than simple linked lists or ordered linked lists, whether we can define other types of conceptual structures which will help us to perform these operations better. Those are known as the well known data structures like linked lists, we will see other well known data structures like trees, graphs and other things.