we begin with discussion of some basic design techniques for algorithms
um we will start with fairly simple problems that many of which you may have seen but we will um hopefully see solutions to non trivial problems which you may not have seen
the first example i would like to start with is finding the minimum element in an array okay
so the input problem is find min
the input is an array a and the output is a minimum element in a
element with whose value is the minimum
okay this is a problem which most of you would have see earlier um in your um perhaps the first programming course may be later even in your course on data structures right
let’s anyway go over the solution
okay so the standard solution is you have you take a temporary variable
there is a temporary variable which contains the current minimum
so this contains the current minimum and it’s updated at each step
so you start with the first element in the array okay and then you scan the array element by element
first element then the second element third element and so on up to the last element in the array
each time you compare the array element with temp and update temp if necessary
okay this simple technique actually is used is very powerful and used often which is if your input is in the form of an array or a list then if you can solve the problem for the first n minus one elements of the array
can you solve it for the n th element okay
this is the step which is put in a loop
so let me just write this down
so the important step is this
if \( a[i] \) is less than temp then um temp is set to \( a[i] \)
this is the crucial step and this is usually put in a loop
okay i varies from one to \( n \) where let’s say \( n \) is a size of the array and at the end of this
temp will have the minimum element in the array
so i will be when i discuss algorithms um often i may not um write the full code
[04:06] in fact i will not write the full code
the idea is to give you the main ideas behind the design the design of this algorithm um
i may not take care of some of the um stray cases okay
i may not initialize variables properly etcetera etcetera
so all these programming details i will not get into okay
the main idea is to get the main um the techniques and the ideas behind the algorithm
which solves the problem and you should be able to sort of take these ideas together and
write the program which actually works okay
yes so if you look at this this is the um level at which we will describe algorithms may be
even less
okay i could just say scan the array element by element and update the current minimum
as required okay
(Refer slide time [05:00])
okay there are so there are two things um couple of things that i would like to point out
so this thing um this design technique is often called well induction both recursion and iteration actually go with go with this
i will explain this a bit more
the second technique the second sort of thing we will keep in mind is ordering okay and the third thing is to store um store value already computed
[06:01] i must add store necessary this value okay
so let’s go each of these one by one
so induction what you mean by induction is this
you want to solve a problem okay
(Refer slide time [06:18])
your input has some size
okay the array has say size n
now the one way to solve it is supposing you can solve this problem for smaller values of input okay
if this problem can be solved for smaller values of input now can i extend these solutions to a solution for the bigger input okay
if you can solve the problem for the smaller values of input can you extend the solution to the to a bigger value of the input
for instance if i can solve a problem for all arrays of size n minus one can i solve this problem for an array of size n
[07:06] now this step which “ “ from n minus one to n is often put um is the crucial step and once you come up with this step we just put this step in a loop or you use recursion okay you first recurse on an array of size n minus one and then extend it to an array of size n or you scan the array one by one element by element and you update the solution as you go along
so this this is what i mean by induction and it’s at the base of every algorithm design technique
okay you can call it the mother of all algorithm design techniques
you will learn some more fancier things later but this is the very crux of design of every algorithms
second point that i would like to make is ordering the input okay um looking at the input in the right order often helps [08:02] in this case it’s simple you know you look at the array elements in increasing order of the array index okay um in fact you could look at the array in in any order it really doesn’t matter um but there are cases we will see cases where ordering plays a very crucial role okay in solving the problem um the third thing which is also fairly simple here is to store some of the values that you already computed okay that’s the third point want to make now even in this case it’s simple you just store the previous minimum um the variable temp the temporary variable which we add so these are values that you would like to use in future almost every algorithm design technique that we will study are a combination of these three some of them will use just induction and storing whole values um [09:04] some of them will use all three some of them will just order the input and use induction okay so these three are things that you must keep at the back of your mind when you design when you design any algorithm let us analyze this algorithm so here is the here is an analysis every algorithm that we design we will analyze okay analysis is as important a part of this subject as design and we would like algorithms to be as fast as possible um in the worst case all our analysis pertain to worst case and we would like to design algorithms which are as fast as possible
okay so in this case we look at every array element one okay and we make one comparison for array element
[10:04] um this leads to um we make n minus one comparisons
um we don’t really compare anything with the first element but with each of these subsequent we make one comparison
n is the size of the array
so we make n minus one comparisons
along with comparison okay we need to um we also store this new value in temp right but the number of times we do this exactly equals the number of mean it’s um let’s say one more than the number of comparisons we make
so if you are not really worried about constants then you can just focus on the number of comparisons
as long as i bound the number of comparisons the other small operations that i do like incrementing the index variable of the array or storing the value in temp
[11:06] they all are of the same order as um as the number of comparisons
so the time taken here we would say is order n
okay the time taken is order n but our focus is just on the number of comparisons
okay um i think the question that one can ask and one should ask is is this the best can i find the minimum element in an array in using less than n minus one comparisons
well we just think about it um or even if you don’t think about it i guess most of you will jump up and say of course you need n minus one comparisons
right that’s the sort of first reaction that most people will have
without making n minus one comparisons how can you find the minimum um
[12:00] why is this so
um can you logically argue that you actually need n minus one and you know you can’t do with less than n minus one
this argument is easy but not absolutely trivial okay um
let’s see so why should we make n minus one can we do it with less
well um every element you have to look at least right
every element in the array must be compared to something or the other
if it’s not compared okay then you may actually make it the minimum right
it could be the minimum
if you had um output something else as a minimum then this could have been made the
minimum or you can easily make some other element of the array um the minimum
so without comparing if you do not compare an element then you cannot um tell the
minimum element in an array
[13:01] now this gives you um this doesn’t give you n minus one
it gives you n by two okay assuming n is even
so at least n by two comparisons are needed okay
the reason every element must be compared must be in some comparison okay
um what i mean is every element must be compared with some other element
okay this can be done with n by two by the way okay
so um the first element is compared with the second element
the third element is compared with the fourth element okay and so on
the fifth element is compared with the sixth element and so on
[14:02] so with n by two comparisons if n is even or n by two plus one if n is odd um
well ceiling of n by two comparisons are all that’s necessary to um to sort of satisfy this
condition that every element must be in some comparison
now clearly n by two is not the right answer
n minus one is the right answer and why is it that one is n minus one
so here here is an argument
um see initially when you have when you have not made any comparison there are n
candidates for the minimum right
each element in the array can actually be a minimum right
you don’t know which of these n elements are minimum
now when you make a comparison you can get rid of only one of these candidates right
when you compare let’s say at some stage you have candidates x one x two up to x k
okay some k elements are candidates for the minimum
[15:07] each of them um based on the comparisons that you have made
previously each of them is equally likely
i mean each of them can be a minimum
now if you compare two of these candidates you can get rid of one right
whichever one is smaller that still remains a candidate
whichever one was larger that no longer remains the candidate but you can only get rid of one candidate
so with each comparison i can only get rid of one candidate right
i have n candidates to start with
so um i need n minus one comparisons okay
this is okay this is actually a complete proof though a bit hang wavy
you can make it more um more rigorous also
um here is one more way of looking at it
ok supposing you draw the following graph
so initially we have i have all these nodes
[16:02] let’s say the elements x one x two up to x n
these are elements and the array and also vertices in our graph
okay let me just put a circle around these two indicate that these are also vertices
now when you compare x i and x j here is x i and here is x j i draw an edge between these two okay
now when you compare let’s say x one and x two and x n i draw this edge
now x j and x n are compared i draw this edge and so on
as you make comparisons i keep drawing these edges okay yea
(Refer slide time [16:40])
so once your program ends terminates you have done all these comparisons now i look at this graph and i look at each connected component in this graph if there are more than one connected components in this graph then i will not be able to tell the minimum okay
[17:05] now in each connected component i know which is a minimum there will be one which is a minimum but i can surely give values to these so that i can pick the global minimum from any one of these connected components okay here this is an argument that after the comparisons are over once you finished all comparisons i better have one connected component this means um from a discrete discrete structure’s class you know that you need n minus one edges good so this is um this is just the same argument um both of them have the same same idea behind both for instance here you start with n connected components and each time you add an edge you can decrease the number of connected components by one utmost one okay [18:00] so it’s the same argument okay so you need n minus one and this sort of simple scan of the array does it with n minus one comparisons okay let’s look at a slight variation of this problem
now i want to find not just the minimum in the array but also the maximum
okay so this problem is called max min okay
so the input is in array a okay and the output is the maximum and the minimum elements
in a
so i want both the maximum and the minimum element
now if i just wanted to find the maximum the the procedure is clearly the same as the one
for the minimum element okay
[19:05] what if i want to find both both the maximum and-
well i could first find the minimum and then i can find the maximum
how many comparisons does this take
well n minus one for the maximum
n minus one for the minimum and that makes it two n minus two right
so the number of comparisons that a naïve algorithm
so this is the number of comparisons that the naïve algorithm makes
we can ask the same question
is this the best
you can try the previous argument that we had of connected components and so you can
show that you need m minus one
okay that’s fine but when you see more than that is very difficult to prove
[20:02] um it’s not absolutely impossible but it’s difficult and if you try to certainly
increase it to two n minus two it’s impossible
okay you will not be able to prove it
okay let’s look at some small values
supposing i have four elements
okay i have let’s say x one x two x three and x four
the naïve algorithm took x one compared with all of them okay found the minimum and
then we were done
then we took again took x one we compared it with all of them and while maintaining the
maximum okay the temporary maximum
now many of these comparisons are repeated right
so you see that um many of some of these comparisons that you make when you “the whole thing out are repeated
so our aim is to sort of get rid of these unnecessary comparisons
now in four elements here is what you can do
[21:00] i first compare x one and x two
okay so supposing x one is less than x two
so this is the first comparison okay
x one is less than x two
now i compare x three and x four
supposing x three is greater than x four
this is my second comparison okay
now where do i find the minimum
i mean how do i find the minimum
the minimum clearly is either x one or x four
okay it’s the smaller of x one and x four
so i compare these two and find the minimum right
similarly i compare these two and find the maximum
good so how many comparisons i have made
one two three and four
so four comparisons
what does our old old algorithm say
(Refer slide time [21:59])
[22:00] it say two n minus two
n is four
so this is six when n is four right
so we seem to have done certainly better than two n minus two
when there are four elements we have certainly done better than um than two n minus two
and well the trick was once we found the minimum the minimum and maximum
between x one x two and x three x four then for the minimums right i only need to look at
x one and x four
okay the smaller on the left hand side and the smaller on the right hand side
i don’t have to bother about the bigger one and similarly for the maximum
so this trick can be applied recursively
well it’s certainly worth trying and let’s do it
okay so what we do is this
here is the let’s say i have you have an array of size n right
[23:00] so let’s divided this into two parts okay
recursively find the maximum in this part
let’s say max l
this is the left part and that’s the right part and min l
recursively find the maximum minimum on the right hand side
so that’s max r and min r okay
now how do i find the maximum and minimum
well i need to compare these two to find the minimum
i need to compare those two find the maximum
okay so here’s an algorithm that seems natural
i divide this into two parts okay
divide this into two parts
let’s say two equal parts
two equal halves
halves are always equal
okay um i find the maximum and the minimum on the left hand side
the left half i find the maximum and minimum in the right hand side
[24:03] now i compare the two minimums to output the minimum of the array
i compare the two maximums to find the maximum of the array
how many comparisons does this algorithm take
okay um
let me write down the algorithm but in future with this explanation you should be able to
write the algorithm right
so so divide into halves
let’s say left and right okay
um then recurse on both parts okay on the left and on the right um and the answers are
max l and min l and max r and min r
then put these things together to get the minimum and maximum okay and then compute
final solution from the solution of the two parts
[25:25] well i have written the um essence of the algorithm without really writing details
i hope you can fill in the details right
(Refer slide time [25:27])
okay um define procedures and write down recursive calls and you know do these two comparisons and output the minimum maximum
okay do how many comparisons does this take
that’s the question we need to answer
so let’s say \( t_n \) is the time taken by \( \max \min \) on arrays of size \( n \) okay
[26:13] it’s the time taken by \( \max \min \) on arrays of size \( n \)
then \( t_n \) is there are two problems of half the size
you solve two problems of half the size
there is two \( n \) by two okay plus two more comparisons
one between the two maxs to get the new max one between the two mins to get the new minimum okay
okay this these are this is for the recursive call
so you call the left hand side
that’s \( n \) by two right hand side \( n \) by two and then two
well if \( n \) is odd i would have a ceiling and floor somewhere but let’s let’s not worry about it for the time being
okay let’s assume that \( n \) is even
you can assume \( n \) is the power of two okay um
we also know that $t_2$ is one
okay for two elements i can find it in one comparison
so what’s the solution to this recurrence
um let’s see
so the easiest way to solve all this is to check how this recurrence behaves
so $t_n$ is nothing but two and now i open this out
this is $t_n$ by four plus two plus two okay which is two square $t_n$ by two square okay
plus two square plus two
right you can right this down once more and essentially we want to see how what pattern
this follows
(Refer slide time [27:49])

![Image showing the recurrence relation: $T(n) = 2T(n/2) + 2$ with $T(2) = 1$ and recursive expansion]

well it’s not too difficult to guess what the pattern is
the pattern is this
so $t_n$ is two to the i right $t_n$ by two to the i plus two to the i plus two to the i minus one
and so on all the way up to two okay
[28:14] so now um we set we set n by two to the i to be two because we know that t of
um t of two is one okay
then we have t of n is two to the i right
this t this becomes two
so t of two is one
okay plus um two to the i plus two to the i minus one and so on up to two okay
so this is nothing but two to the i plus one plus well two to the i minus one and so on up
to two okay
[29:06] um you can check that this is nothing but we also know that two to the i plus one
is n and i hope you can solve this okay
i will leave it for you to solve this
this is nothing but n and this sum you will get as n by two minus two right
if you sum this up using the usual geometric series and use this fact that n is two to the i
plus one you will get that this sum is nothing but n by two minus two
well put this together
you get t of n is three n by two minus two
okay can check that this when n is two this is one which is what we want and this also
satisfies the recurrence that we had okay
(Refer slide time [29:56])
the recurrence was let me refresh your memory
the recurrence was um t n is twice t n by two plus two
[30:03] so if i put t n equals three n by two minus two this satisfies the recurrence
you can prove that t n is this “ “
well so the number of comparisons that we seem to make using this method is three n by
two minus two which is certainly better than two n minus two okay
we seem to have done something fairly mechanically and we seem to have improved um
the number of comparisons made quite drastically okay
so this technique is is called divide and conquer is used by the british um in the last
century i mean last century and even before that okay
we will put it to good use in designing algorithms
so let me write down the main steps of this technique
the first step is to divide the problem into okay i will say two parts okay
[31:14] often we will want these parts to have equal sizes okay
often of equal sizes
okay the next step is recurse on each part
okay so you recurse on each part and solve each of them both the parts
if there are two okay and the final step is put these solutions together
put these solutions together to get a solution for the original problem
okay often you can just do this blindly
in fact the for the max min we could have done it blindly
take this array of size m divide that divide this array into two arrays to size n by two okay
find the maximum and minimum on the left array the maximum and minimum on the
right array and once you have these two solutions together now you find the maximum of
the whole array comparing the two maximums
find the minimum of the um array by comparing two minimums okay
again the essence um is this is this um is this induction okay in the sense that if you could
solve problems of smaller size which is what you are doing in this recursion you are
somehow putting these together to get a solution for the big problem okay
how you will find these small problems varies from problem to problem
[32:08] okay let’s look at um another problem where we will apply this this method
method blindly and we will see what we get okay
this problem is to find both the minimum and second minimum
okay the minimum is the smallest element in the array
the second minimum is the is the next one
okay the second smallest element in the array
so your input is an array and you are going to find both the minimum and the second
minimum
the usual way you would do it is you first scan the array and find the minimum right
now you again scan the array and find the second minimum
um the other way to do it is to have two temporary variables okay
temp one and temp two
in temporary one i store the current minimum
in temporary two i store the current second minimum
now when i get to an array element i let’s say the i the element i compare this first with
the with the minimum
[34:08] then with the second minimum that i have and based on the result of these two
comparisons i update minimum and second minimum okay
now this will take we have seen how how many comparisons this will take um
it’s two n minus two okay as it was in the max min case and let’s just apply our divide and conquer paradigm blindly to this problem and see what you get okay
so how would we do this
so here is the array
the array is of size n
i divide this equally into two paths okay um
i find a minimum in second minimum here
so let’s say min left and the second min left min right and s min right okay
[35:01] so i have found these four values and now i want to find the minimum and second minimum for the entire array
the minimum is not a problem
okay so i just compare min l and these two values right and i can output the minimum
the smaller of these two is the minimum
now what do i do about the second minimum
now supposing min l was smaller than min r
okay so without loss of generality this min l so assume min l was less than min r which means at this point min l has been output okay
now what are the candidates for the second minimum
clearly min r is still a candidate for the second minimum okay
second minimum of the left hand side s min l is also a candidate for the second minimum but one one of these elements we have we have sort of um thrown out which is s min r
[36:10] this does not figure in the picture at all
so we need exactly one more comparison to get the second minimum which is supposing this is true then compare the second minimum on the left and the minimum from the right okay
so you need to compare just these two elements and you can see that um the minimum of these two will give me the second minimum right
the minimum of the entire array i get by comparing these two minimums okay the minimum the left hand side minimum the right hand side minimum and the second minimum i can get by comparing the minimum element which lost the first comparison which was larger and the second minimum of the element that one okay
(Refer slide time [36:57])

[37:02] so that will give me the second minimum
good so how many comparisons have we does this take okay
well if you write down the recurrence this seems to be very very similar to the previous one
so what is-, so if \( t_n \) is the time taken by the algorithm we have two sub problems each of
size \( n \) by two
okay that takes time \( t_n \) of \( n \) by two and then we have two more comparisons one with the
two minimums and one to find the second minimum
okay we also know that \( t_2 \) of two is one
if i have two elements one comparison suffices to find both the minimum and second
minimum and these set of equations are exactly the same as the set of equations we had before
so the solution is \( t_n \) is three \( n \) by two minus two
so the number of comparisons we make is three half \( n \) minus two and it’s not um it’s not
two \( n \) minus two okay
(Refer slide time [38:01])
it’s much less
one can ask is this the best
can we better than three half three half n minus two
this question can be asked both for max min okay and also for min and second min
well it turns out that these two problems behave differently
for max min three half n minus two is the best we can do okay
so three half n comparisons is the best um we need three half n comparisons
while in this case when minimum and second minimum you can do with actually less okay
the divide and conquer sort of paradigm gave us three half n but that’s not the best
so why so let me give you a reason why you can do better here
okay to to better this you need to understand a bit more as to how this algorithm works okay
[39:07] um let’s unfold the recurrent the recursion out and see what this looks like
okay initially i have array elements x one x two x three x four and so on all the way up to x n
okay what we do is we just divide it into two parts right
we divide it down the middle and then you recurse on these two okay
on the left half we divide it again into two halves recurse divide recurse divide recurse divide recurse
now we come down all the way down to when they avail as size two
this is when the comparisons start happening okay
the recurrence sort of bottoms down till you reach arrays of size two
now i will find the minimum of x one and x two that goes up right
the minimum of x three and x four is pushed up right
[40:03] also the second minimum is pushed up but let’s not worry about that for the minute
um the maximum case also it works very similar
you put the max min max mins are pushed up at each level
okay we will jus focus on the minimum element okay
um so the minimum element is pushed up
from here the minimum element is pushed up from there
at the next level i will compare the minimum of these two
okay this contains a minimum of x one x two
this contains a minimum of x three x four
this contains a minimum of these two which is actually the minimum of these four and so on
this would be a bigger tree and so on okay
all the way up to the root okay and this root you can the minimum is known right
so for instance here x n minus one and x n are compared
this is the minimum
[41:01] at the next instance you compare it with the with the um the other two okay
this would be x n minus two and x n minus three and so on okay all the way up to the root
where the minimum is known
now this looks like if n was say power of two this looks like um very familiar complete binary tree okay
so there are log n levels right
there are are n leaves
there are log n levels
um and in each level we sort of have some sort of minimum and some portions of the array and these are pushed up okay
now where was the minimum element
the minimum element sits somewhere in this in this in this array and at each stage it’s pushed up
it sort of wins its comparison um each time at each level of the of this of this tree and it finds its way to the to the top
okay somewhere with perhaps came from the left
[42:02] may be it came from the right came from the left came from the and so on okay so it does traverse some root all the way from the root node to a leaf and this is where the minimum element resided
now what can you say of the second minimum element
now well the crucial sort of observation that you need to make to um speed up the algorithm is that at some stage the second minimum must have been compared with the minimum element okay
this is this is absolutely crucial
if the second minimum element were never compared with the minimum element then you really don’t know which of these two is the minimum because in each comparison the second minimum one it was smaller than every other element right
so was the minimum element which of these two is minimum
to know that you must have compared the second minimum element with the minimum element okay
[43:05] now let’s look at this picture
okay here is the picture
how many elements did um the minimum element you know win against
how many elements were compared with the minimum element
okay if you look at this picture and you follow this at each stage in this in this in this path down from the loop to the leaf the minimum element was compared with exactly one one element right
the length of this path is log n
so the minimum element was compared with utmost log n elements in this tree
(Refer slide time [43:36])

right which means log n elements in this array were compared with a minimum element
and one of these log n elements remember must be the second minimum
okay so to find the second minimum all we do is this
find the minimum using this using this tree
okay you can do it recursively if you want
once you find the minimum element collect all elements that the minimum element won
against was compared against okay
[44:07] if you have this tree in front of you you can certainly go down the tree and figure
out which were these elements
among these elements find out which is the minimum and that will give you a second one
there are log n elements
so initially you made n comparisons
okay may be it’s n minus one
n minus one comparisons to find a minimum and then you need about log n minus one
comparisons more to find the second minimum okay
this then is actually optimum um though we will not do it in this course um there’s an argument which shows that you need n plus log n but um it’s surprising that you can actually do this in n plus log n and this problem um in this way it differs from the previous problem

a straight forward application of divide and conquer doesn’t work

[45:09] you need to use some more intuition

you need to understand a problem a bit more um come up with new ideas okay and that’s what algorithm design is all about um

often there are problems which are which are hard

you really don’t know what to do and when you come up with a smart answer to an algorithm you feel really um you feel nice

(Refer slide time [44:49])