Problem Solving through Programming In C

Prof. Anupam Basu
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

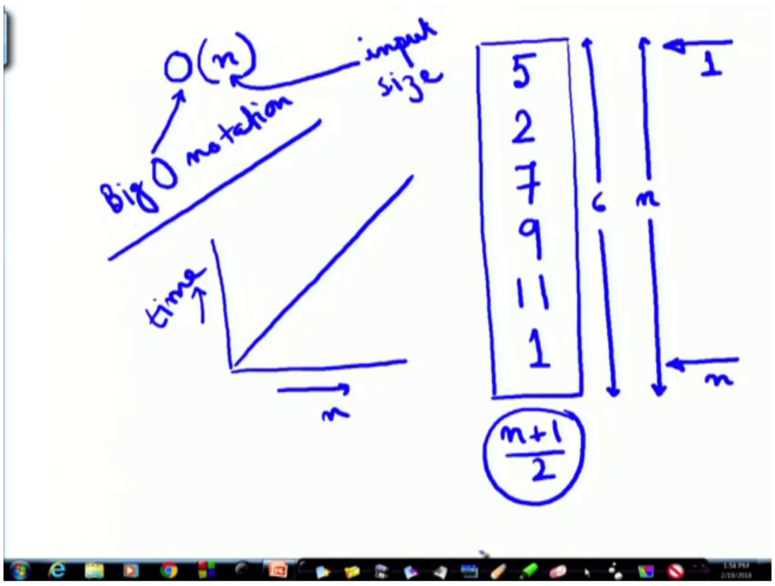
Lecture – 43
Binary Search

In the earlier lecture, we have seen, how we can search in a list of items, may be list of integers or may be list of names or list of real numbers in a linear way; that means, we start from one end of the list and we have got a key and we check for every element whether the element in the list matches the key or not. In that way, we go on from top to bottom. And ultimately, if we get the match of the key to an element of the list, we declare it is found. And we also say where it is found.

Otherwise, if it, if we exhaust the list and still do not find the element, then we say that the list is not found. And the way in which we have find searching from the top to bottom, that process is known as linear search. We have also seen that, the time that time that is required to search for an element in the best case, it would be of order 1; that means, at the very beginning, we can find the element. Otherwise, we have to show many comparisons, we have to do? The minimum is 1 if we get the match at the beginning of the top of the list. Otherwise, we have to we may have to exhaust all the elements and whether it is found or not found. We can say only after we have compared all the elements.

So, if there is a list of n elements, the maximum number of comparisons that I may have to do will be n. Therefore, on an average, it will be n plus 1 by 2 that we say in complexity of algorithms parlance. In that parlance, in that terminology, we call it order of n ok. That is not the main issue with us, but with we can say therefore, that in a list of elements, if I have got n elements and the list is not sorted, so, may be 5 2 7 9 11 1, say, this is the list and I want to search for a key.

(Refer Slide Time: 02:44)

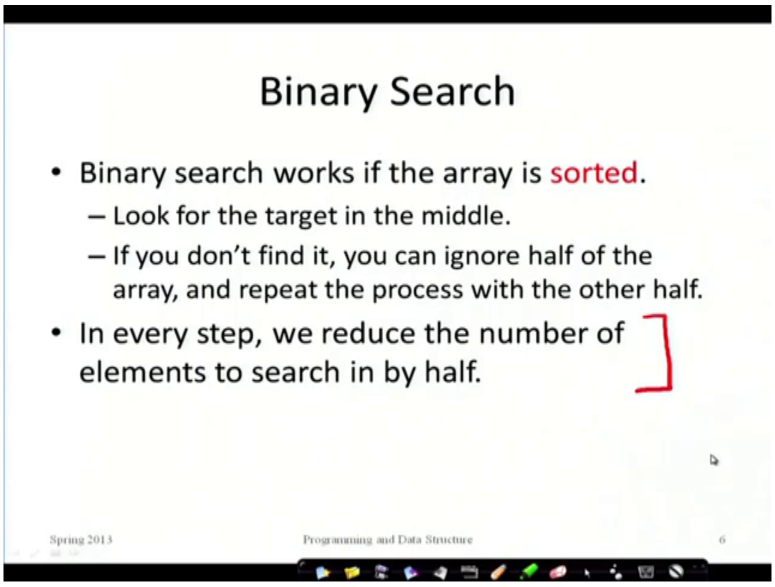


So, this in this case, it is 6. But, in general, I can say that there are n elements right. In general, I can say number of elements is n. So, the best case I can get a match here, the number of comparisons I require is 1 and the worst case I have to come up to this and compare all these n elements.

So, the number of comparison on an average will be n plus 1 by 2, all right. That is a average number of comparisons that we denote in computer parlance as of the order of n. This o is has got name called big o. I am sorry let me write it in this way. This called big o notation big o notation. So, where it just denotes the how much time a computer can a program will take to run, in terms of the input size, what is this n? This is the size of the input data, we call it input size.

So, obviously, you can see that, as these list increases, if it becomes 100, then the time average time that will be taken average. This is average complexity all right, will be more than this. If it is with 1000 even more and as increases, the time will increase in a linear fashion. As n increases, the time will increase in a linear fashion. However, the way we are searching is known as linear search. One thing to note is that, in this case, we are not making any assumption. This is time. We are not making any assumption about the way in which the data items are organized.

(Refer Slide Time: 05:37)



Binary Search

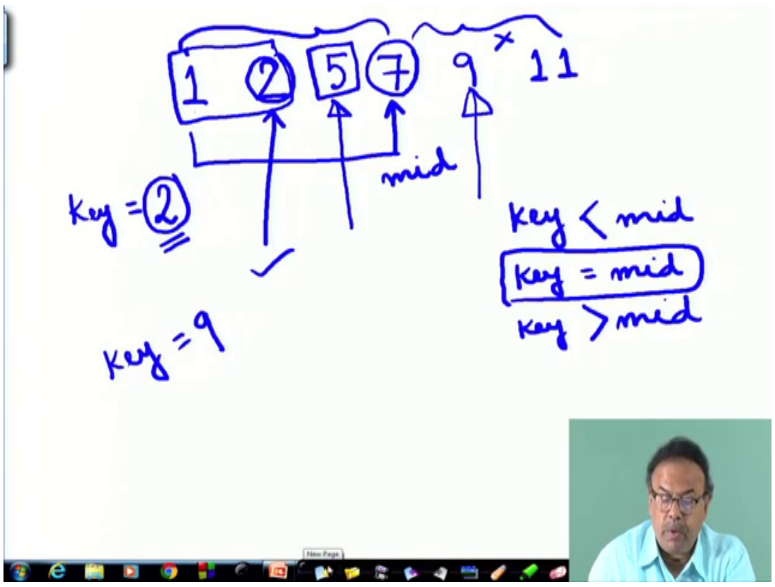
- Binary search works if the array is sorted.
  - Look for the target in the middle.
  - If you don't find it, you can ignore half of the array, and repeat the process with the other half.
- In every step, we reduce the number of elements to search in by half.

Spring 2013 Programming and Data Structure 6

Now, in an endeavor, to see, if we can make it better, we today discuss another very important type of search algorithm, which is known as binary search. Why the name binary is coming very clear in a moment, but the important thing to note is that, in this case, the list or the array must be sorted; that means, it is organized in some particular way; either in an ascending order or in the descending order ok.

So, that is why, it is called binary. Let us let me just try to explain it with an example. Say, I have got the array, but in a sorted way.

(Refer Slide Time: 06:31)



So, I have got 1 2 the elements where there in the earlier case one 2 five seven nine eleven right 1 2 5 7 9 11. So, 1 2 5 7 9 11. Now suppose my key is well, let the key be 2. Now, we will start at middle of this array first we will look at the middle. Now, since is this is an even number, even sized array, 6 elements middle can be somewhere here. Say, let me take this, this is the midpoint.

Now, I compare the key with the middle element 2 and 7 are being compared. 2 things can happen; either 2 the key I I will rather say, either the key is less than the mid element or the key is equal to the mid element or the key is greater than the mid element. 3 things can happen. Now, if the key is equal to the mid element, then my search immediately stops. Yes, I found it and where did I find it? The index is mid.

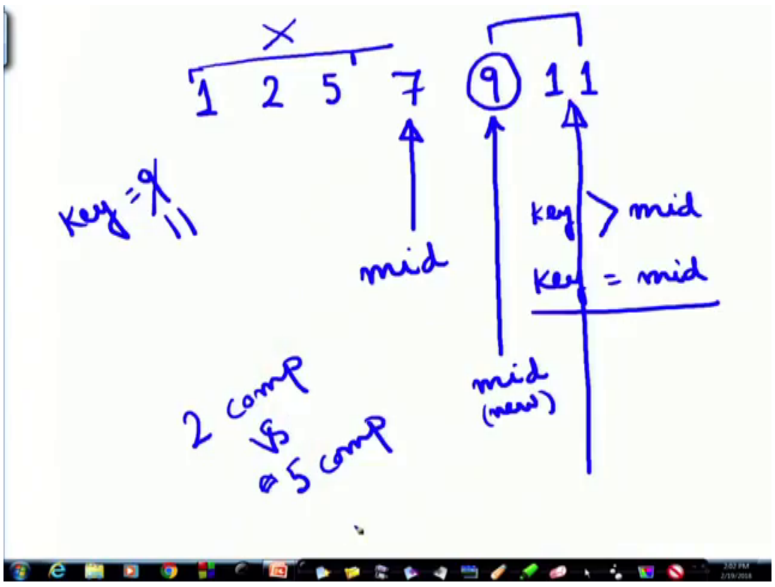
If the key is less than the mid element, as is the case here, then what can I say? I can immediately say that, the I still do not know whether the key is there in this array or not, whether the key is here or not I do not know as here. But the thing that I say is, that the key, since it is less than the mid, it cannot be on this side, it cannot be on this side. Not possible. It must be if at all on this side, because, the key is less than the mid.

Therefore, I can restrict my search within this period. Suppose, within this zone, now what I will do? I will again take the middle point of this array, right? So, between here it is becoming very simple. So, the middle point of this part is suppose 5. I could have had it from 1 1 to 5, but I am just taking from 1 to 7.

I come at this point and again now I compare this element and still it is less therefore, am sure that it is not in this area it must be in this zone. Now I again, only for this part , I apply again, I find mid element and this is here and I find that the key and the element are matching. So, it is found. So, how many comparisons I needed here? In this case, I need it 1 2 3 comparisons. In the case of linear search of course, you could have got it luckily here in 2 comparisons all right.

But, what would have happened if my key was 9? In the case of linear search, suppose, the key was 9 in that case, in the case of linear case what would what would have happened I would have started from here 1 comparison, 2 comparison, 3 comparison, 4 comparison and the 5th comparison I would have got it right, on the 5th comparison I would have got it. But, let us see what would have happened in the case of the new search technique that we are looking at.

(Refer Slide Time: 11:04)



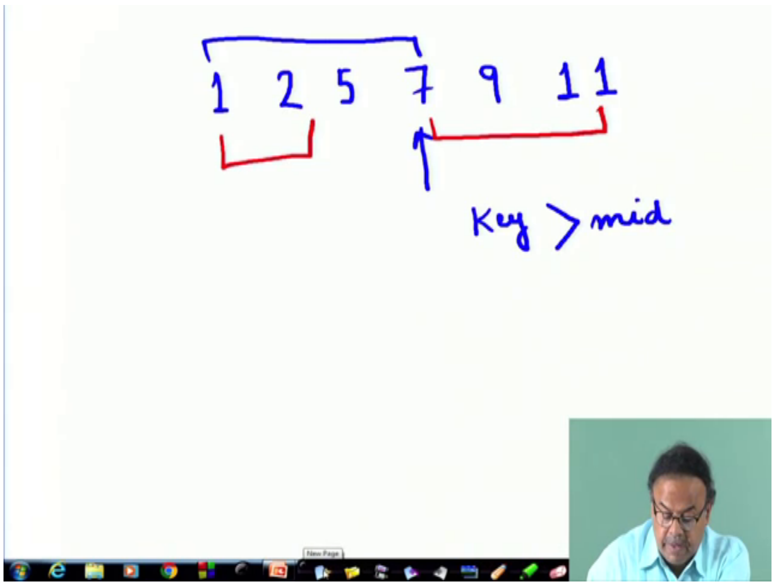
So, again I have 1 2 5 7 9 11 and my key is 9. I know that in case of linear search, I needed 5 comparisons. What will happen here? I will come to the middle point here. I find that the key is greater than mid. This is mid. The key is greater than mid. Therefore, I know that the key if at all cannot remain in this zone. This part is ruled out this part is ruled out. It must be in this zone. So, again within this I find the new mid. So, the new mid comes here. And I find this new mid mid new this and I check this, now I find that the key is equal to mid.

Therefore, I find my search completes here with this index as output. Now how many comparisons I needed here? 1 2 only 2 comparisons.

So, 2 comparisons verses 1 2 3 4 5 comparisons in the case of linear search. Now why was it reduced? The same thing would have happened with if my key was 11. Let us have a look if the a key was 11. In that case, suppose, the key was 11, in the case of linear search, I would have required 6 comparisons here to reach at 11.

However, in this case, I would I would be sure if key is greater than is, then my next iteration will be between these 2 and I would have found mid new mid and I would have got it with 3 comparisons. Now why is it becoming so? Why is the number of comparison being reduced? The reason is we have got this array and at every stage, what I am doing is, am looking at the key and depending on whether the key is greater than the mid.

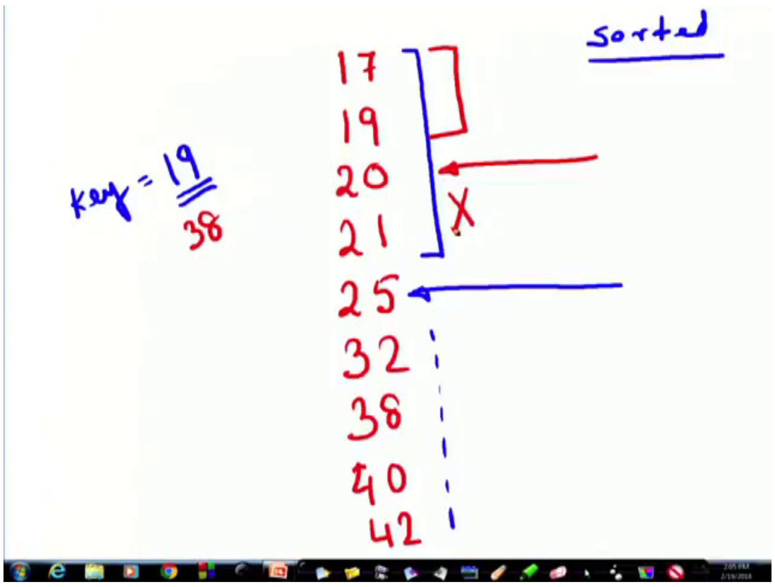
(Refer Slide Time: 13:44)



I am concentrating on only one half of the array either this array or this array. And then, depending on the key value, I take suppose it is greater, then I will concentrate on these zone. If it is less, I will concentrate on this zone. And iteratively, I will be reducing my search to a smaller array. Let us have another example.

Let us have another example. Let us have it a little bigger all right. Let us have 17 19 20 21 25 32 38 40.

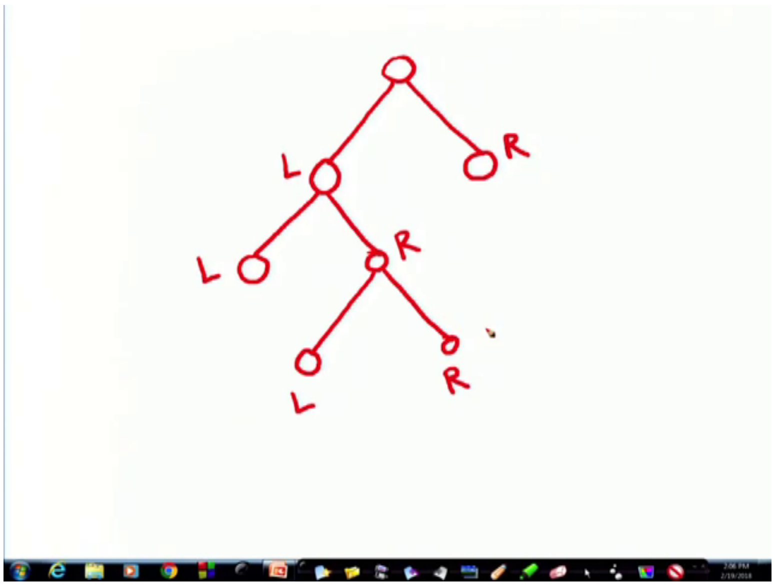
(Refer Slide Time: 14:45)



So, now I have got again even number let me have 42. Say odd number arrays 1 2 3 4. Now I have got 9 elements right. Now I will start with the mid element. What is the mid element? Here, between this, the mid element is this. And suppose my key is 19 , then I know that, my key cannot lie, since it is less than 19, it cannot lie in this part of the array. If at all, it will lie on this part of the array. Why? Why do I say that? I can say that because, this array is sorted.

Otherwise, I could not have said. Since it is an increasing order, I can say that, since the key is less than the mid element, therefore, it must be in this zone. And so, I come to this zone and find out the mid element again may be in this zone or up to this. Say I come here and find out the mid element this again. So, am looking, I am immediately reduced my list to half, right? This half, it could have been if my key was 38, then I would have restricted to this half not this half. But for 19, am restricted to this half. Now again, 19 is less than 20, immediately I will restrict myself to this half and I will not consider this part, gradually am going to the lower half or the higher half at every stage.

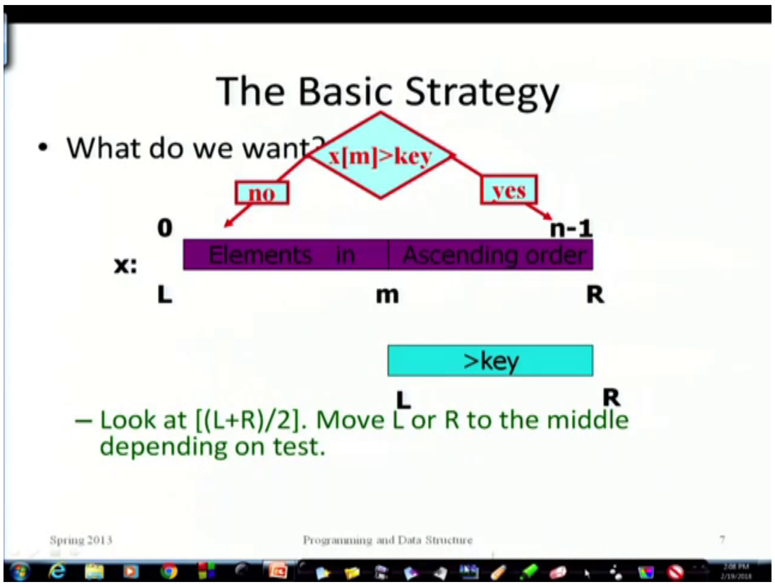
(Refer Slide Time: 17:10)



So, what is happening is something like this. Am starting with the whole array, then am concentrating either in the left half or on the right half. Then again, depending on that, either on the left half or on the right half if it is in the right half, then among them, this am coming to left half and right half. So, at every stage, am dividing the array to half. That is why this is called binary search. Either it is here or it is not here not in this zone.

So, in that way, I carry on. So, con consequently, the number of elements that I restrict my search to gets reduced at every iteration. Let us look at this in a little more detail. So, in every state, we reduce the number of elements by half. I think this is clear now, that this statement. Now, if you do not find it, you can ignore the half of the array and repeat the process.

(Refer Slide Time: 18:24)



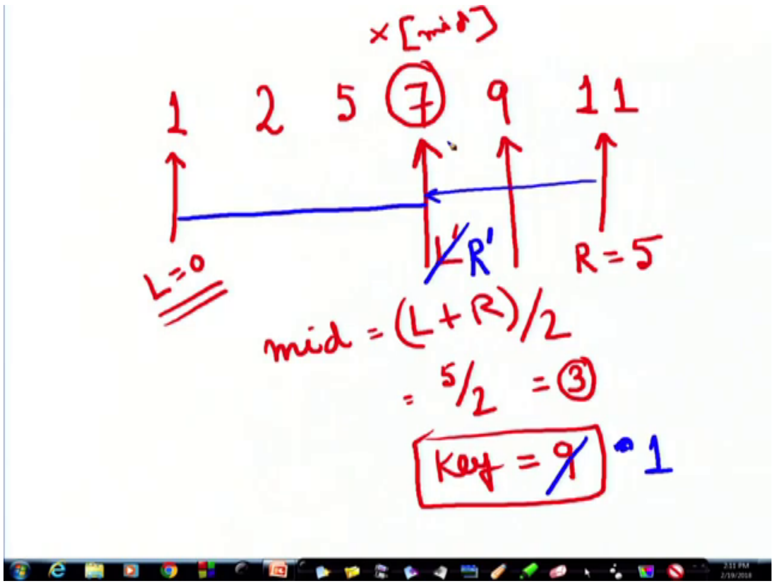
The Basic Strategy

- What do we want?
  - Look at [(L+R)/2]. Move L or R to the middle depending on test.

So, now, let us look at this basics strategy. What do you want? The array is from 0 to n minus 1, n elements. Here, I was showing mid zone only. So, this is the mid m and this is the left end. This is the right end. So, there are 2 indexes, indices all right? Now, am taking the key and I am checking this part m. The element m this element; that is, xm. And depending on whether if it is greater than the key, if xm, if the key is greater than xm, key is greater than xm is less, then I will be concentrating here. If xm is greater than the key, then I will be concentrating on this half on this half or on this half. That is what I was explaining till now ok.

So, here, if it is no, then I will. If it xm is less than the key , then the I will come to this half. Otherwise, if it is less I will come to this half greater than that all right. So, will first given this L and r, what is my mid element? How will I compute my mid element or m? This L plus R divided by 2. And then, depending on whether it is less or greater, we will we will move the left or L or R till the middle depending on the test. So, again let me show it with a example that we are showing.

(Refer Slide Time: 20:21)

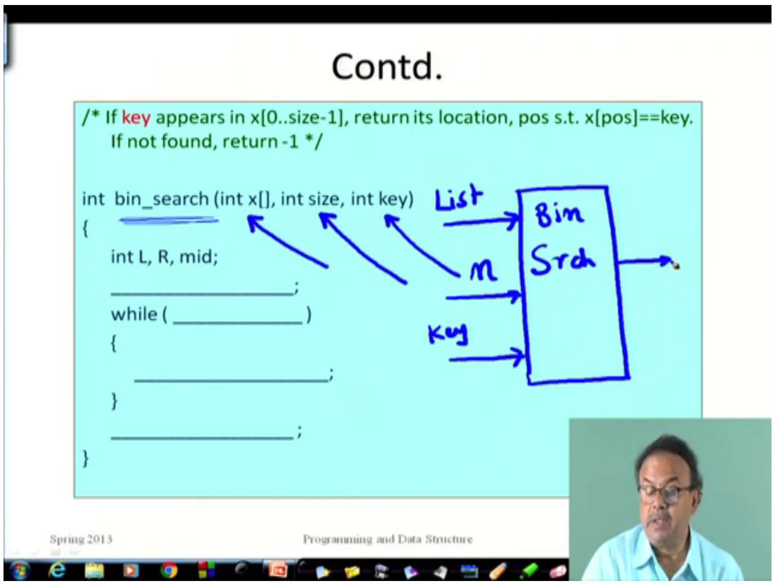


So, something like this 1 2 5 7 9 11 what is being said is, 9 11 ah. So, L is L equals 0 and R is equal to 1 2 3 4 5 6 0 1 2 3 4 5. So, R is equal to 5. So, mid is L plus R divided by 2. So, that is 5 divided by 2. So, we can take 2 or we can take 3 depending on. So, if I come to if I take 2, if I take 3, then am coming to mid. So, what I will do is, if 3 is my midpoint, if I take 5 by 2 is 3, then, that means, it will be in case of odd I will add 1 to that. So, I can move. Now if the key is suppose, key is 9 suppose the key is 9. So, my mid was the 3rd.

It is 0 1 2 3 or let me 0 1 2 3. There is my mid. Now, since the key is key is greater than the element x m x mid, since it is x mid is less. I will be restrict my search in this area. Therefore, what I will do? I will move this L to mid. So, this will be my new L. And I will find the mid with. Now, next mid will be this new L plus R by 2. So, it will be this element if my search was on the other side; that means, if the key was not 9, but the key was 2 or 1, let us make it 1.

Then, at this point, I find that the key is less than x mid, then I will be restricting my search in this area. In that case, I will move my R, I will shift this R over here, and this the mid will be R, R will be the mid. So, next, I will be restricting my search in this half and forget about this half. So, that is how we at every iteration, we break down the entire array into halves. So, if you have understood this, let us proceed. Repeat the search operation in the reduced interval. So, what we are doing is, we are looking at this binary search algorithm and we are trying to design a function.

(Refer Slide Time: 23:35)



Contd.

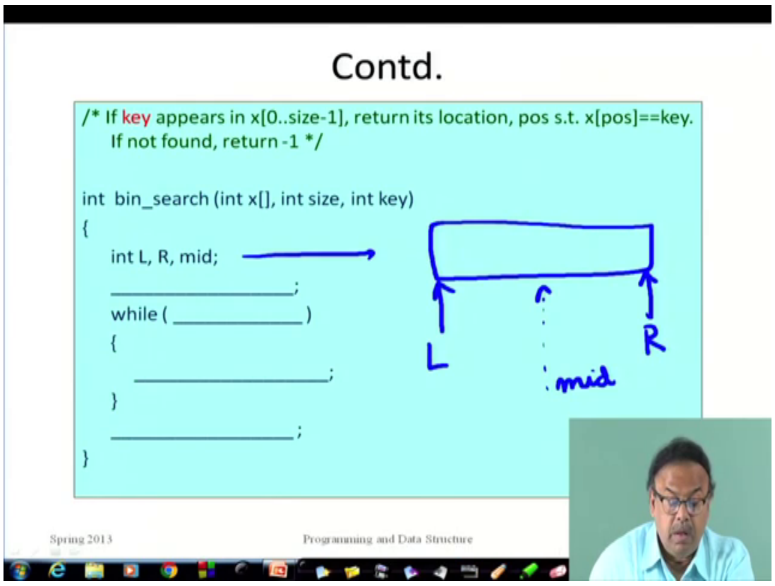
```
/* If key appears in x[0..size-1], return its location, pos s.t. x[pos]==key.
   If not found, return -1 */

int  bin_search (int x[], int size, int key)
{
    int L, R, mid;
    _______________;
    while ( __________ )
    {
        _______________;
    }
    _______________;
}
```

And finery binary search is a function, whose name am just keeping as bin search all right. This is the name of the algorithm. And what are the parameters? Let us see. The parameters are one is the array that is being passed the list that have to search. I also need the size of the array and I need the key.

So, these are 3 things. So, binary search is a function. What are it is inputs? The list or the array which is x, the size or what we are calling about right now n and the key and what will the output be, bin search we will tell us whether it has been found or not ok. But, so, it will be a 0 or 1. Found or not. 0 or 1 or it can written as an index also ok. So, now, if we proceed with this idea, then let us develop the algorithm step by step inside this function. I have got the search. I have got the list the list is given to me.

(Refer Slide Time: 25:39)



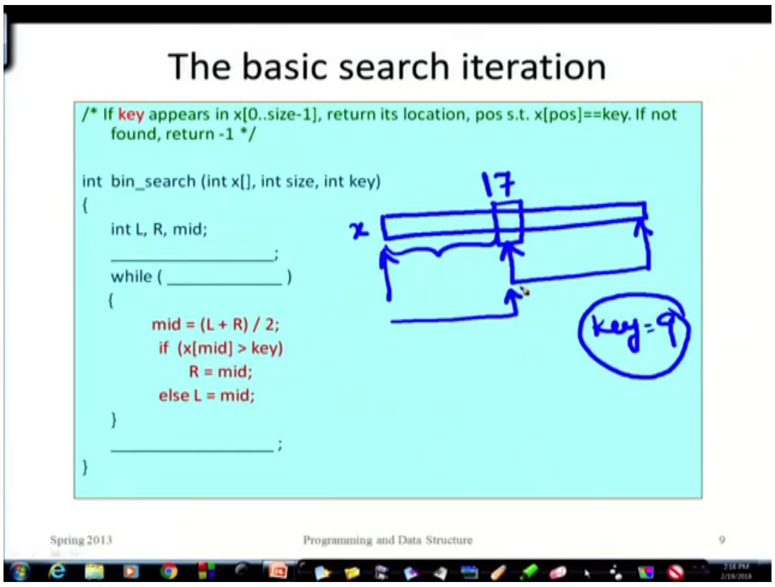
Contd.

```
/* If key appears in x[0..size-1], return its location, pos s.t. x[pos]==key.
   If not found, return -1 */

int bin_search (int x[], int size, int key)
{
    int L, R, mid;
    ________________;
    while ( ____________ )
    {
        ________________;
    }
    ________________;
}
```

This array is known to me. I declare internally L R and I have to use a another index called mid. Now, this has got no meaning outside. This function next step what should I do?

(Refer Slide Time: 26:04)



The basic search iteration

```
/* If key appears in x[0..size-1], return its location, pos s.t. x[pos]==key. If not
   found, return -1 */

int bin_search (int x[], int size, int key)
{
    int L, R, mid;
    ________________;
    while ( ____________ )
    {
        mid = (L + R) / 2;
        if (x[mid] > key)
            R = mid;
        else L = mid;
    }
    ________________;
}
```

Next step would be while some condition mid, I will have to find out L plus R by 2. As you know, if this is the array x, if the mid element is greater than the keys So, the key, say this mid element is 7 and my key is 9. So, this element is greater than I know that this element is greater. So, this is 17. This is greater than the key. Then I will have to keep my search within this zone. So, this R will be updated and the R will come here. Otherwise, L will be moved here. This much is clear. You think over this and build upon this algorithm we will take it up in next step again, we have to decide on how to build this things up.