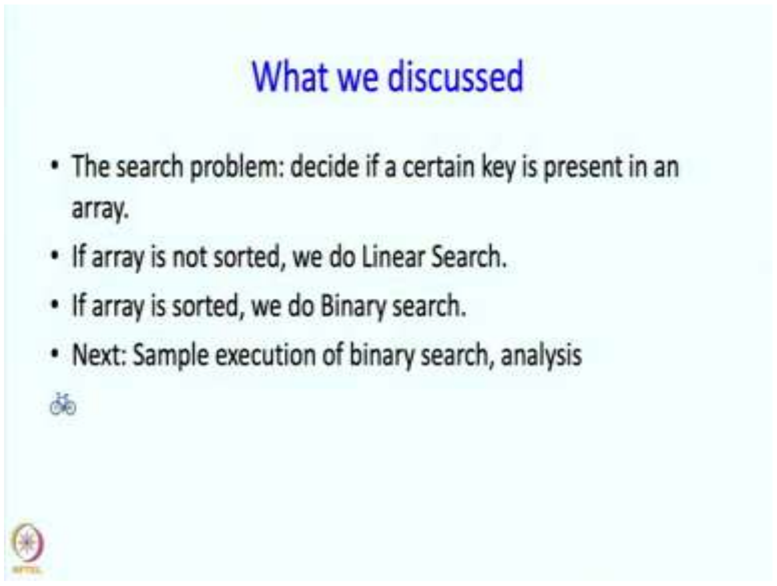
An Introduction to Programming through C++

Professor Abhiram G. Ranade
Department of computer science and Engineering,
Indian Information Technology Bombay

Lecture 18 (Part - 2)
Arrays and recursion
Binary search analysis

Welcome back!

(Refer Slide Time: 0:22)

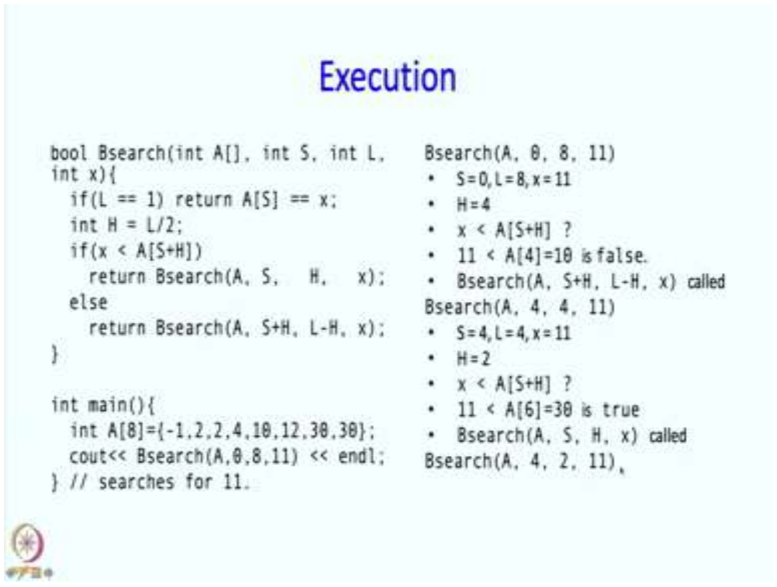


What we discussed

- The search problem: decide if a certain key is present in an array.
- If array is not sorted, we do Linear Search.
- If array is sorted, we do Binary search.
- Next: Sample execution of binary search, analysis

In the previous segment, we defined a search problem and we also defined Binary Search. Now we are going to do sample execution through our code.

(Refer Slide Time: 0:29)



Execution

```
bool Bsearch(int A[], int S, int L,
int x){
  if(L == 1) return A[S] == x;
  int H = L/2;
  if(x < A[S+H])
    return Bsearch(A, S,   H,   x);
  else
    return Bsearch(A, S+H, L-H, x);
}

int main(){
  int A[8]={-1,2,2,4,10,12,30,30};
  cout<< Bsearch(A,0,8,11) << endl;
} // searches for 11.
```

Bsearch(A, 0, 8, 11)
- S=0, L=8, x=11
- H=4
- x < A[S+H] ?
- 11 < A[4]=10 is false.
- Bsearch(A, S+H, L-H, x) called

Bsearch(A, 4, 4, 11)
- S=4, L=4, x=11
- H=2
- x < A[S+H] ?
- 11 < A[6]=30 is true
- Bsearch(A, S, H, x) called

Bsearch(A, 4, 2, 11),

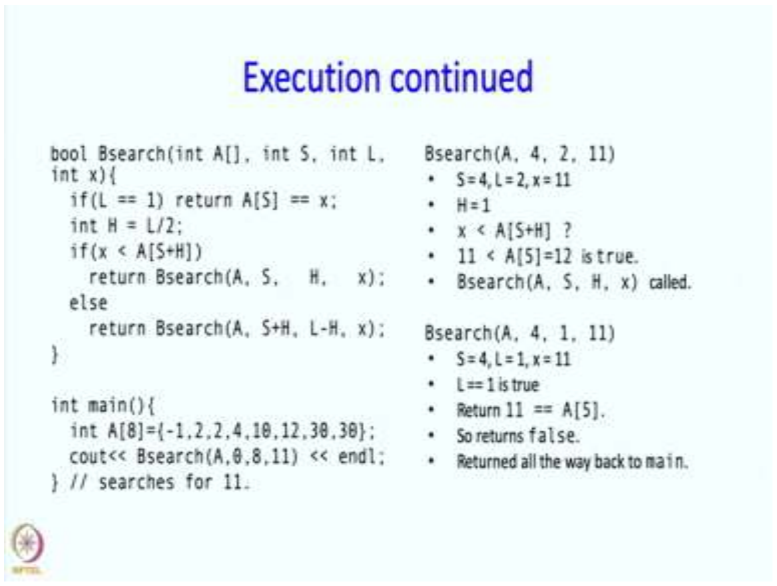
Okay, so this is our code and I have just removed the comment, so that I can just fit my code into this, into the slides, so it is really the same code. So, we are searching this array over here, okay. So starting index is 0, length is 8 and we are searching for 11. So, the first call is this call is this call. So Bsearch(A, 0, 8) is our first call. So, how does this call execute. So, when this call start executing, S has the value 0 which comes from this argument itself and L has the value 8, again from the argument and x has the value 11 from the argument, okay.

So, we are at this point then we do a check, is L equal to 1. Okay. Well L is 8 then certainly not equal 1, so this statement does not execute. So, we come to this statement. So at this point we are defining a new variable H whose value is going to be L/2, so H becomes 4. Next, we execute we execute this statement. So, we check whether the, what is the relationship between x and A[S+H]. Well what is x, x is 11, what is S+H, so S is 0, H is 4,so S+H is 4. So, what is A[S+H]? Well this is A, so 0, 1, 2, 3, 4, so A[S+H] is 10. So, the question that we are asking is this, is 11 smaller than 10, well that is not true, 11 is not smaller than 10. So this comparison comes out to be a false, so therefore we are going to make this call.

So, this is the call that we make next, Bsearch(A, S+H, L-H, x), but what are the values, the values are S+H is 4, L-H what is L, L is 8, H is 4, so it is also 4 and x is 11. So, a new recursive call starts. So, what happens in this recursive call? Well we are going to start from here. So, S in this recursive call is 4, which is we got from our arguments, so S is 4, L is also 4 and x is 11. So, in this new recursive call these are the values, so we are going to do the same thing again. So, what happens over here, so we first check if L is 1, well L is not 1, L is 4 now, so in any case the statement is not going to get executed. So, we now create H equals L by 2, so H so L is 4, so H becomes 2. Next, we are going to ask, how does x compare to A[S+H]? So, what is the value of x, again it is 11, what is S+H this time, so S is 4, H is 2, so S+H is 8. So, A[S+H] is 0, 1, 2, 3, 4, 5, 6, so this is 30.

So, we are comparing 11 with A[6] which is 30, so 11 is smaller than 30, so this is true and therefore, we are going, so this is true so we are going to make this recursive call. So, Bsearch(A, S, H, x) gets called, so what are these values? So, these value are 4, S is 4 over here, H is 2 and x is 11. So, the values that are going to get called, that call you are going to make is Bsearch(A, 4, 2, 11). Alright

(Refer Slide Time: 4:32)



Execution continued

```
bool Bsearch(int A[], int S, int L,
int x){
  if(L == 1) return A[S] == x;
  int H = L/2;
  if(x < A[S+H])
    return Bsearch(A, S,   H,   x);
  else
    return Bsearch(A, S+H, L-H, x);
}

int main(){
  int A[8]={-1,2,2,4,10,12,30,30};
  cout<< Bsearch(A,0,8,11) << endl;
} // searches for 11.
```

Bsearch(A, 4, 2, 11)

- S=4, L=2, x=11
- H=1
- x < A[S+H] ?
- 11 < A[5]=12 is true.
- Bsearch(A, S, H, x) called.

Bsearch(A, 4, 1, 11)

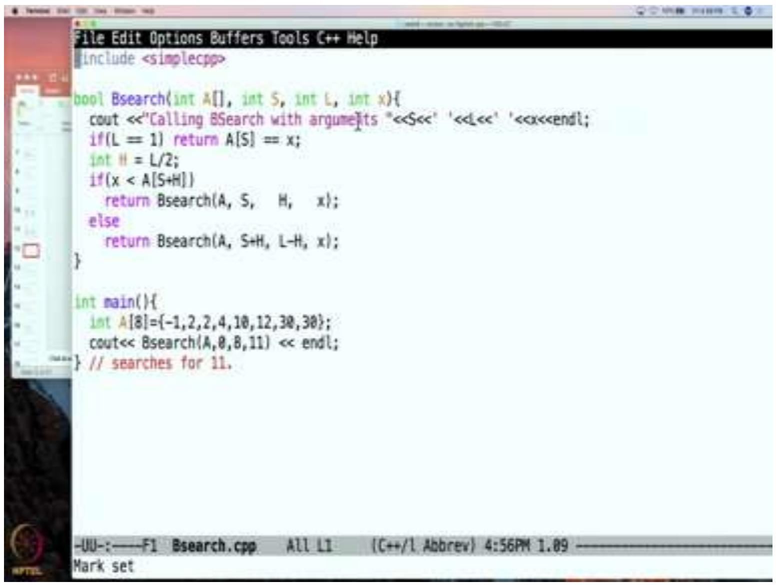
- S=4, L=1, x=11
- L==1 is true
- Return 11 == A[5].
- So returns false.
- Returned all the way back to main.

So let us examine this call Bsearch(A, 4, 2, 11), so what's happen in this. So, again the variables in the activation frame take values, S will take value 4, L will take value 2, these come directly from the arguments, x also will take value 11, then we are going to execute, we are going to execute this statement, but L is not 1, it is 2, so nothing will happen. Then we move on to this, what will happen to H, well H will become 1, half of L and then we are going to execute this.

So, we are going to ask is x less than A[S+H]. So, what is x again? It has stayed 11 all the time and S+H is now 5, so we are going to ask A[S+H] is 0, 1, 2, 3, 4, 5, so we are going to ask is x less than A[5] or 12 and this is also true. So, if it is true, then we are going to make this call, so in this call S is 4, h is 1 and x is 11. So, we are now going to start another recursive call, okay. So, again we start from this point and because of the argument list, S is going to get 4 this value, L is going to get 1 this value, and x is going to be 11.

So, next what is going to happen, we are going to check if L is equal to 1, but, yes, this time L is indeed 1 and therefore, we are going to return whether A[S] is equal to x, okay. So, what is A of S, so S is 4, so 0, 1, 2, 3, 4. So, A[S] is x. So, L is equal to 1 is true and therefore, we are going to return whether 11 is equal to A[5]. So, is 11 equal to A of 5? Well, so A of 5 is 12, so this is not equal and therefore, we are going to be returning false and all the previous recursive calls will just return the same thing, forward it back, send it backwards, and so this is the value that will go onto main as we expected, alright.

(Refer Slide Time: 7:21)

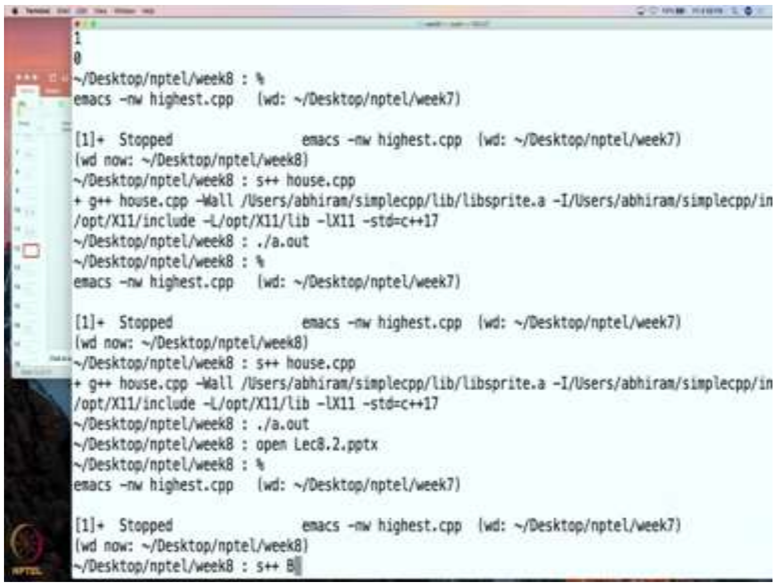


```
#include <simplecpp>

bool Bsearch(int A[], int S, int L, int x){
  cout <<"Calling BSearch with arguments "<<S<<' '<<L<<' '<<x<<endl;
  if(L == 1) return A[S] == x;
  int H = L/2;
  if(x < A[S+H])
    return Bsearch(A, S,   H,   x);
  else
    return Bsearch(A, S+H, L-H, x);
}

int main(){
  int A[8]={-1,2,2,4,10,12,30,30};
  cout<< Bsearch(A,0,8,11) << endl;
} // searches for 11.
```

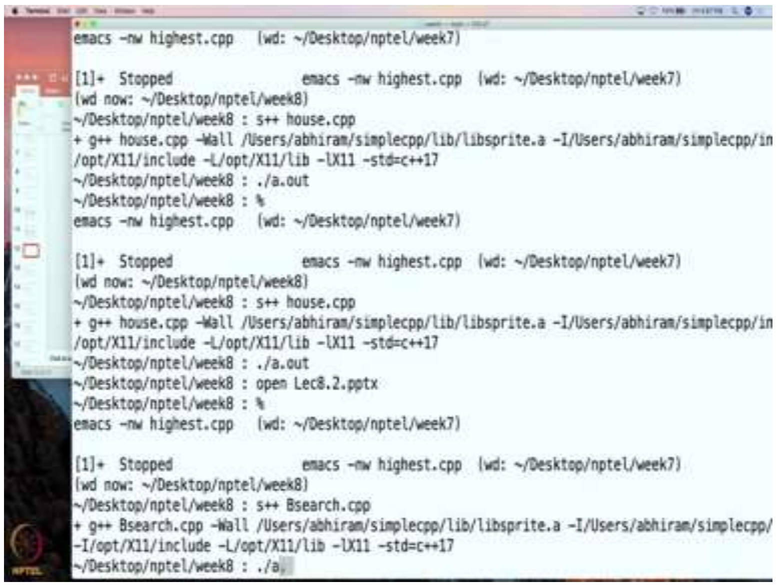
```
-UU-:----F1  Bsearch.cpp    All L1     (C++/l Abbrev) 4:56PM 1.09 -----------------------
Mark set
```

Okay, so let us do a quick demo of this. So, the function is Bsearch, so let me show it to you. So, this is Bsearch and Bsearch is the same as what I have shown you, but the only thing is I am writing over here what the arguments are. So, just you see how the function executes.

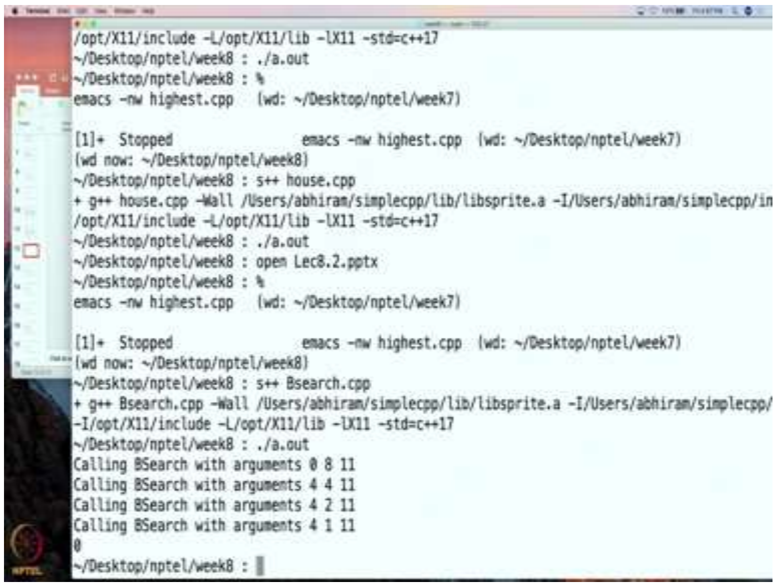
(Refer Slide Time: 7:48)



```
1
0
~/Desktop/nptel/week8 : %
emacs -nw highest.cpp   (wd: ~/Desktop/nptel/week7)

[1]+  Stopped                 emacs -nw highest.cpp  (wd: ~/Desktop/nptel/week7)
(wd now: ~/Desktop/nptel/week8)
~/Desktop/nptel/week8 : s++ house.cpp
+ g++ house.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecpp/in
/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week8 : ./a.out
~/Desktop/nptel/week8 : %
emacs -nw highest.cpp   (wd: ~/Desktop/nptel/week7)

[1]+  Stopped                 emacs -nw highest.cpp  (wd: ~/Desktop/nptel/week7)
(wd now: ~/Desktop/nptel/week8)
~/Desktop/nptel/week8 : s++ house.cpp
+ g++ house.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecpp/in
/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week8 : ./a.out
~/Desktop/nptel/week8 : open Lec8.2.pptx
~/Desktop/nptel/week8 : %
emacs -nw highest.cpp   (wd: ~/Desktop/nptel/week7)

[1]+  Stopped                 emacs -nw highest.cpp  (wd: ~/Desktop/nptel/week7)
(wd now: ~/Desktop/nptel/week8)
~/Desktop/nptel/week8 : s++ B
```



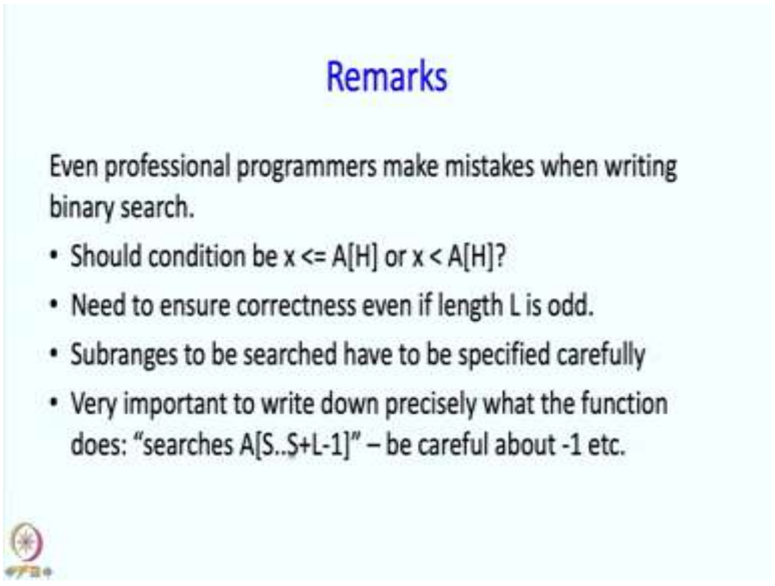
```
emacs -nw highest.cpp   (wd: ~/Desktop/nptel/week7)

[1]+  Stopped                 emacs -nw highest.cpp  (wd: ~/Desktop/nptel/week7)
(wd now: ~/Desktop/nptel/week8)
~/Desktop/nptel/week8 : s++ house.cpp
+ g++ house.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecpp/in
/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week8 : ./a.out
~/Desktop/nptel/week8 : %
emacs -nw highest.cpp   (wd: ~/Desktop/nptel/week7)

[1]+  Stopped                 emacs -nw highest.cpp  (wd: ~/Desktop/nptel/week7)
(wd now: ~/Desktop/nptel/week8)
~/Desktop/nptel/week8 : s++ house.cpp
+ g++ house.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecpp/in
/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week8 : ./a.out
~/Desktop/nptel/week8 : open Lec8.2.pptx
~/Desktop/nptel/week8 : %
emacs -nw highest.cpp   (wd: ~/Desktop/nptel/week7)

[1]+  Stopped                 emacs -nw highest.cpp  (wd: ~/Desktop/nptel/week7)
(wd now: ~/Desktop/nptel/week8)
~/Desktop/nptel/week8 : s++ Bsearch.cpp
+ g++ Bsearch.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecpp/
-I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week8 : ./a
```



```
/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week8 : ./a.out
~/Desktop/nptel/week8 : %
emacs -nw highest.cpp   (wd: ~/Desktop/nptel/week7)

[1]+  Stopped                 emacs -nw highest.cpp  (wd: ~/Desktop/nptel/week7)
(wd now: ~/Desktop/nptel/week8)
~/Desktop/nptel/week8 : s++ house.cpp
+ g++ house.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecpp/in
/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week8 : ./a.out
~/Desktop/nptel/week8 : open Lec8.2.pptx
~/Desktop/nptel/week8 : %
emacs -nw highest.cpp   (wd: ~/Desktop/nptel/week7)

[1]+  Stopped                 emacs -nw highest.cpp  (wd: ~/Desktop/nptel/week7)
(wd now: ~/Desktop/nptel/week8)
~/Desktop/nptel/week8 : s++ Bsearch.cpp
+ g++ Bsearch.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecpp/
-I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week8 : ./a.out
Calling BSearch with arguments 0 8 11
Calling BSearch with arguments 4 4 11
Calling BSearch with arguments 4 2 11
Calling BSearch with arguments 4 1 11
0
~/Desktop/nptel/week8 :
```

So, let us compile it and run it. So, these were this, this is how the search ran, so first it was called with arguments 0, 8, 11 and, of course, the argument A is there, but I am I am leaving that out. Then (4,4,11), (4,2,11), (4,1,11) and if you see what we executed, this was exactly the sequence and even here 0 was written as happen in our execution sequence, the execution that we did by hand as well. So, let us move on to the presentation. So a few remarks, binary search is based on a simple but powerful idea but even though the idea is simple, the details are somewhat tricky and experience shows that even professional programmers may make mistakes when writing binary search.

(Refer Slide Time: 8:57)



Remarks

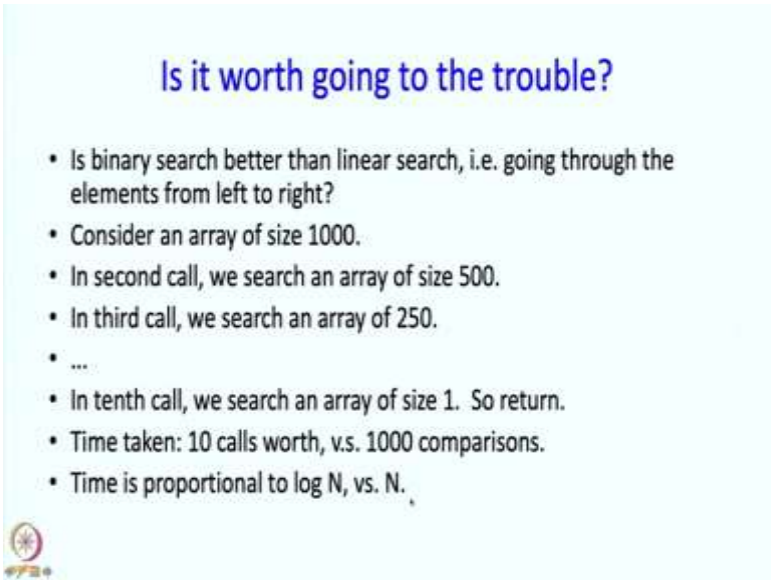
Even professional programmers make mistakes when writing binary search.

- Should condition be x <= A[H] or x < A[H]?
- Need to ensure correctness even if length L is odd.
- Subranges to be searched have to be specified carefully
- Very important to write down precisely what the function does: "searches A[S..S+L-1]" – be careful about -1 etc.

So the mistakes are of this kind, so when we are comparing, it should be asking less than or equal to or less than. So, then there is this question that if the region that we are searching is odd length, then integer division behaves in a slightly different way, then we might expect, so we have to be careful about that. And then, the subranges to be searched have to be specified carefully, so should not be S through H+L-1 or S through H+L, whatever it is, so the way to get it right is to write down precisely what the function is expected to do, so at least once you write down what you want to happen, then you can check whether that is happening.

So, at least you are, you have to be sure as to what you want, so please write the specification very carefully and make sure you are getting these minus ones, minus ones, correct.

(Refer Slide Time: 9:59)



Is it worth going to the trouble?

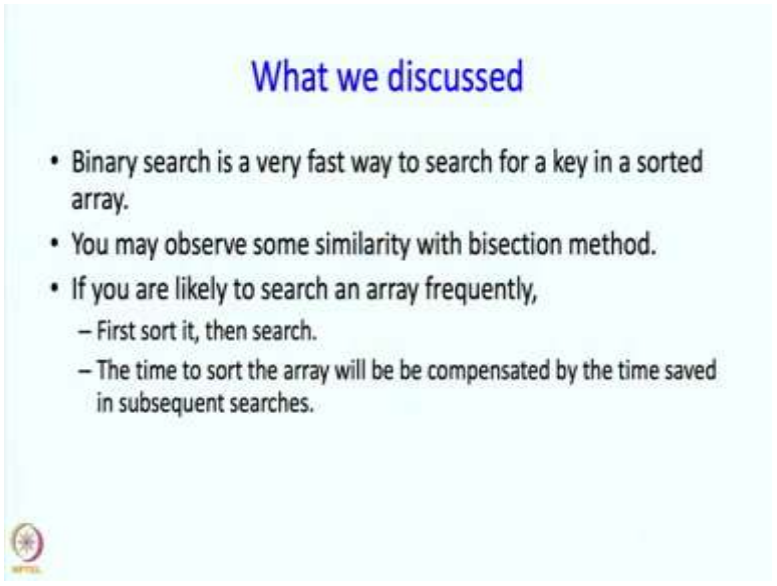
- Is binary search better than linear search, i.e. going through the elements from left to right?
- Consider an array of size 1000.
- In second call, we search an array of size 500.
- In third call, we search an array of 250.
- ...
- In tenth call, we search an array of size 1. So return.
- Time taken: 10 calls worth, v.s. 1000 comparisons.
- Time is proportional to log N, vs. N.

Alright, so now let us analyze this and check look if this is so carefully, have to be written so carefully, is it worth going to the trouble. So, in other words, is binary search better than linear search, just going through the elements left to right. So, if we consider an array of size 1000, in the second call, we search an array of size 500. In third call, we search an array of size 250 and so on. So, every time the size of the array we are searching is roughly halving. So, that means in the tenth call, if you do this 10 times you will get an array of size 1. So, at that point we know, we do not have, we do not require further and we can just return. And whatever we return at that point is return all the way back to the main program.

So, what does this mean, so time taken in this case is about 10 calls worth of work and each call is fairly simple, as oppose to that in linear search we would do 1000 comparisons. Now. let us see what happens if we were searching through million elements. Then here we would have million comparisons and how many calls would we have over here? Interesting as it may you may find, the number of calls over here is only going to be 20.

So, binary search is much-much more efficient, especially if the size of the array being searched is larger, is large. So, put it another way, this is log n, so the time is proportional to log n and this is n, so rather than n, so definitely a big difference.

(Refer Slide Time: 11:51)



What we discussed

- Binary search is a very fast way to search for a key in a sorted array.
- You may observe some similarity with bisection method.
- If you are likely to search an array frequently,
  - First sort it, then search.
  - The time to sort the array will be be compensated by the time saved in subsequent searches.

So, what have we discussed in this segment. We said that binary search is a very fast way to search for a key in a sorted array. And, in fact, this notion of dividing by 2 is something that you have encountered, right, so we did in the bisection method, so even there we were probing in the middle of the, in the middle, in the middle of the range.

So, range here is sort of the size of the array that we are searching, searching over. Then if you are likely to search an array frequently, it seems that you should first sort it and then search it, because the time to sort array will be compensated for by the time saved in the subsequent searches. So, this raises the question, how do you sort an array in the first place? So, this is what we are going to discuss in the next segment. So, let us take a break.