Lecture 6

Linear Searches

Linear Searches

- Suppose you have an array int f[20] which already contains values.
- you also have a target value stored in the variable x int
- you know that this value is also stored somewhere in f
- you want to find the **smallest** index i, where f[i] == x

When we are finished we have a picture like this.

We haven't found x in any place to the left of the line, but we have found it in position i

At the start, before we begin, the line is at the left of the array.

We can say that the value x doesn't appear on the left of the line, because there are no values there

Our job is to keep moving that line to the right until we find the value x. We know that we will find it because we know it is in there somewhere.

```
// f[20] and x already have values

    int i ;

    i = 0 ;
    while ( f[i] != x )
        {
        i = i + 1 ;
        }

// x is not in the part of f before index i
// and now f[i] == x
```

Linear Searches

- The integer square root of a natural number n, is the largest number i where i * i <= n
- e.g. the integer square root of 126 is 11 because 11*11 <= 126, but 126 < 12*12
- Given a natural number n find the integer square root of n

All of the values to the left of the line have the property that when you square them you get a value <= n.

When we are finished, we have found the smallest value i, where n < i*i.

So,this will mean that the integer square root which we are looking for must be the value i-1

0, 1, 2, 3, 4, 5, 6, 7 ...., (i-1), i, (i+1)

At the start all of the values to the left of the line have the property that when you square them you get a value <= n.

Our job is to move that line to the right.

0, 1, 2, 3, 4, 5, 6, 7 ...., (i-1), i, (i+1)

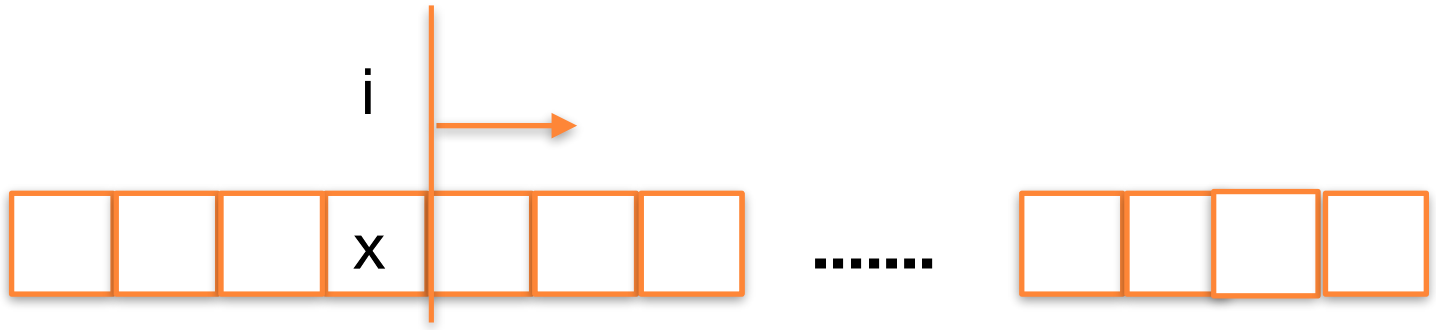
```
int i ;
i = 0 ;

while ( i*i <= n )
    {
    i = i + 1 ;
    }

// all of the numbers from 0 to i-1 have the property
// that when they are squared they are <= n.
// but n < i*i, so (i-1) is the integer square root of n.
```

Linear Searches

- Suppose you have an array int f[20] which already contains values.
- you also have a target value stored in the variable x int
- you know that this value is also stored somewhere in f
- you want to find the **largest** index i, where f[i] == x

When we are finished we have a
picture like this.

We haven't found x in any place to the **right** of the line, but we have found it in position i



At the start, before we begin, the line is at the **right** of the array.

We can say that the value x doesn't appear on the **right** of the line, because there are no values there

Our job is to keep moving that line to the left until we find the value x. We know that we will find it because we know it is in there somewhere.

```
// f[20] and x already have values

        int i ;

        i = 19 ;
        while ( f[i] != x )
                {
                i = i - 1 ;
                }

// x is not in the part of f after index i
// and now f[i] == x
```

Linear Searches

- In the examples which we have seen so far we knew that the value we were searching for existed.
- In the examples using arrays this prevented us from falling off the end of the array. In the number example it prevented us from going on forever.
- What would we need to change if we were not sure that the value we were searching for could be found?

Linear Searches

- Suppose you have an array int f[20] which already contains values.

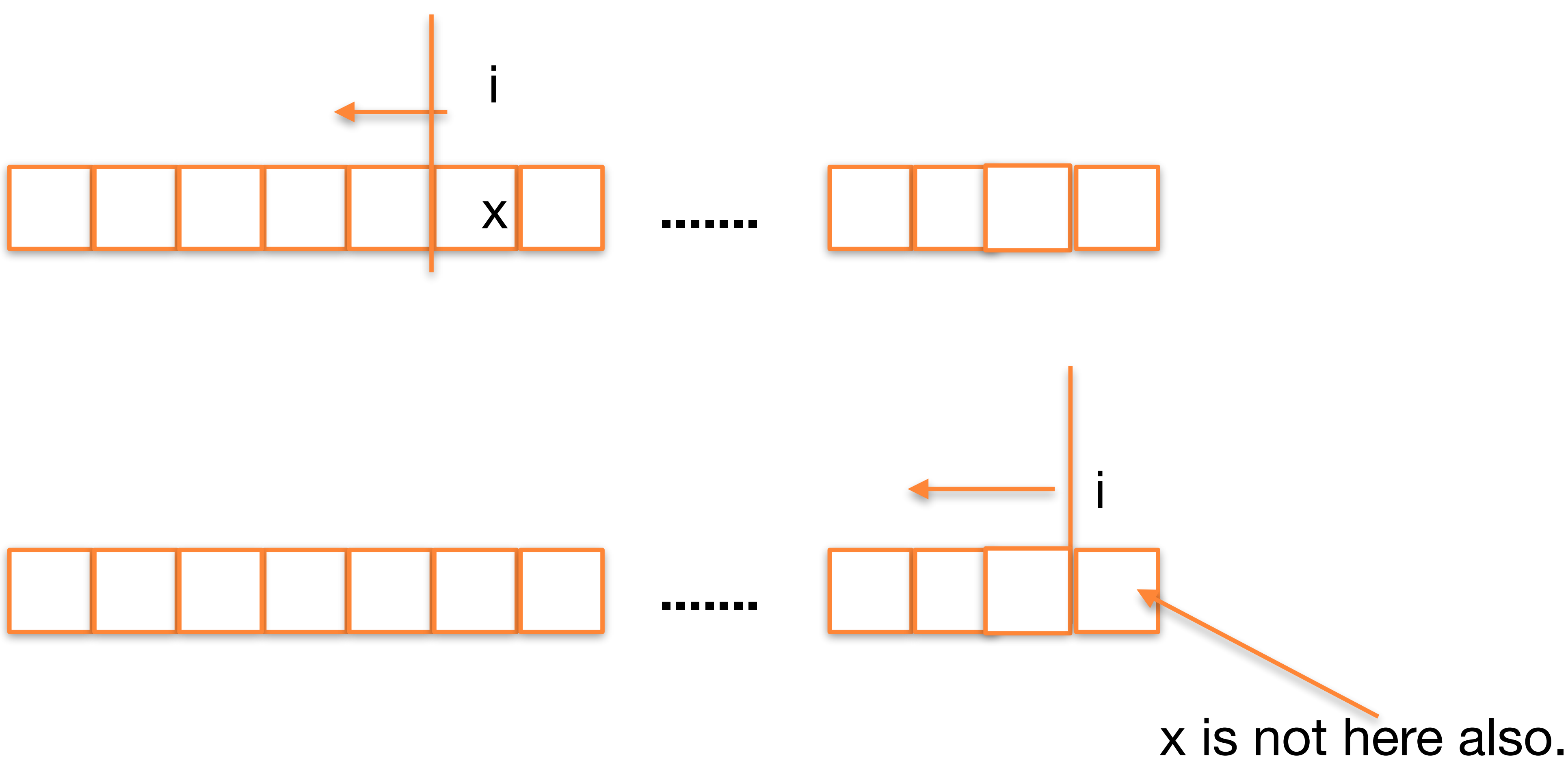
- you also have a target value stored in the variable x int

- you want to find the smallest index i, where f[i] == x, if such an index actually exists.

Now, when we are finished we have a
picture like this.

We haven't found x in any place to the left of the
line, but we have found it in position i.

Or, we haven't found x in any place to the left of the line
and we are at the last place we could look and we don't find it there either.



Case 1.
“From index 0 to index i-1, we didn’t find x“ and “f[i] == x”

Case 2.
“From index 0 to index i-1, we didn’t find x” and “i = 19 and f[i] != x”

We can combine these to get

“From index 0 to index i-1, we didn’t find x”

&&

(f[i] == x) || (i == 19 && f[i] != x)

This describes what will be true when we finish

$$(f[i] == x) \,||\, (i == 19 \;\&\&\; f[i] \;!= x)$$

It can be simplified to

$$(f[i] == x) \,||\, (i ==19)$$

Now if this describes when we are finished

(f[i] == x) || (i == 19)

Then the opposite of this describes when we are not finished

(f[i] != x) && (i != 19)

So this will be the guard on our loop.

```
// f[20] and x already have values

        int i ;

        i = 0 ;
        while (( f[i] != x) && ( i != 19 ))
                {
                i = i + 1;
                }

// none of the locations before index i contain the value x.
// at this stage f[i] == x or i = 19.
// Now we check to see if we found x

        if ( f[i] == x)
                { printf (“ the value was found at location %d “, i ) ; }
        else
                { printf(“The value was not found in the array”) ; }
```

We made use of some Logical Laws in solving that last problem.

[(P || (Not.P && Q)) == (P || Q)]

[Not.(P && Q) == Not.P || Not.Q]

[Not. (P || Q) == Not.P && Not.Q]

Linear Searches

- Suppose we have an array int f[100] which already contains values.

- We want to write a program to determine if the 2nd half of the array is an exact copy of the first half.

Linear Searches

- If the 2nd half is an exact copy of the 1st half then for each value f[j] in the 1st half it will be true that
f[j] == f[j+50]

- If the 2nd half is not an exact copy of the 1st half then there must be some value f[i] in the 1st half where we find that f[i] != f[i+50]

- I am going to try to find a value that doesn’t appear 50 places further on, if I find one I conclude they aren’t exact copies, if I can’t find one and I get to the end I conclude they are exact copies.

Our picture at the end tells us that

“ each of the values in f to the left of index i appear again 50 places to the right”

and either

“ at the current index i, f[i] != f[i+50] ”
or
“ we are at the last position in the 1st half of f, i == 49, and here too f[i] == f[50+i]”

We can write this more precisely as

((f[i] != f[i+50]) || ((i == 49) && (f[i] == f[50+i])))

This can be simplified to

(f[i] != f[i+50]) || (i == 49))

The **opposite** of this then becomes the **loop guard**

(f[i] == f[i+50]) && (i != 49))

```
// f[100] already has values

      int i ;

      i = 0 ;
      while (( f[i] == f[i+50) && ( i != 49))
            {
            i = i + 1 ;
            }

// Each of the values from the start up to position i
// are repeated 50 places further on.
// Now either f[i] != f[i+50]  or i == 49. Let us find check.

      if ( f[i] != f[i+50])
            { printf(" 2nd half is not a copy of first half") ; }
      else
            { printf(" 2nd half is an exact copy of 1st half"); }
```

Linear Searches

- In the last 2 examples we were searching for something without knowing whether we could find it.
- When we don’t have a guarantee that we will find the value we call our search a Bounded Linear Search.

Further problems

- Given int f[100] which already contains values, construct programs to do the following
- Find out if all of the values in f are positive
- Find out if any of the values in f are even
- Find out if f is a palindrome.
- Find out if f is sorted in ascending order

A Challenge

Suppose we have f[5] of int which contains single digit values.
We can read it like a single number.

f

1	2	5	4	3

12543

Suppose we decide to rearrange it so it reads like the next highest number that we could make using these digits.

1	3	2	4	5

13245

How would we do this?