Searching Arrays

Problem:

Find a particular element in an array. If the value is found return its index, otherwise return -1

This is the search array idiom

Go through every element in the array until you come to an element equal to **x**.

```
for each element
    if x equals that element
       remember the index
```

If **x** isn't in the array, set the index to minus one.

Code

Here is the code to search an array for *x*.

```
int index = -1;
for(i = 0; i < a.length; i++)
   if(a[i] == x)
   {
      index = i;
      break;
   }

// index has the correct value
```

What if

- Would the code perform differently if the break statement wasn't there?
- What would happen if the braces weren't there?
- What would happen if the length of the array was zero?

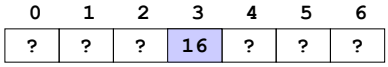
Improvements

- Could this algorithm be improved?
- What do we mean by improved? (Faster, simpler).
- How long does it take?
  - if the element isn't in the array
  - if the element is in the first position
  - if in the last position
  - in the average case

Improvements

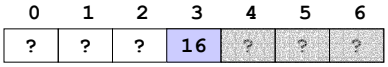
- Could you speed up the algorithm if the array was sorted?
- Could we use a different algorithm if the array was sorted?
- How do you look up a person's number in a phone book?
- If you had only a number how would you find the corresponding name?

Searching a sorted array



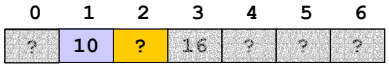
- Imagine you're searching a sorted array for the value 11.
- If you look at the middle element and see that it's 16 what can you say about where the element 11 is?

Searching a sorted array



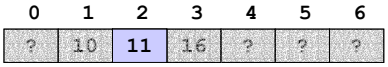
- The element 11 must be to the left of 16 (because you know the array is sorted and 11 is less than 16).
- This means you can immediately eliminate all the elements to the right of 16 (a[4] to a[6]).

Searching a sorted array



- You can repeat this process on the left sub-array.
- The middle element is 10, 11 is greater than 10 and so we can discard the left side of this sub-array.

Searching a sorted array



- Now the sub-array has only one element, a[2], which is 11 and we have found our number.

Binary Search

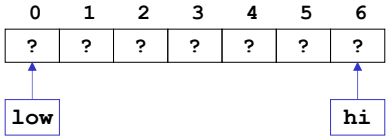
- This technique is called binary search.
- It is much more effective for larger arrays.
- With a million elements in the array, binary search would find the required element using no more than 20 comparisons.
- How many searches would the original idiom have required?

Binary Search

- Binary search can be used on a sorted array to dramatically speed up the search.
- The technique is
  1. Examine the middle element of the array. If it is equal to x, we are done.
  2. If *x* is less than the middle element then search the left part of the array otherwise search the right part.
  3. if there is more array left, goto step 1.

Binary Search - the code

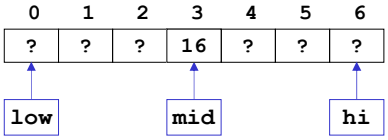
The main trick in the code is to keep two indices that point to the start and end of the sub-array under question.



The sub-array is initially equal to the complete array

Binary Search - managing the sub-array

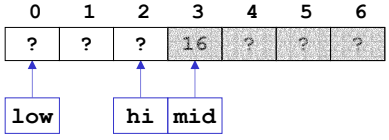
Then you check the middle element



The middle element is `(low+hi)/2`

Binary Search - managing the sub-array

If the element you're searching for is less than the middle element, then ignore all the elements to the right of `mid`.

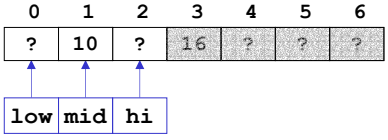


Do this by setting `hi = mid-1`

Binary Search - managing the sub-array

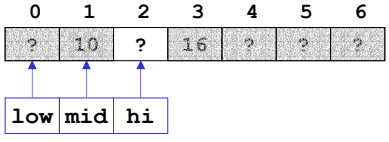
Find the middle element of the new sub-array.

```
mid = (low + hi) / 2
```



Binary Search - managing the sub-array

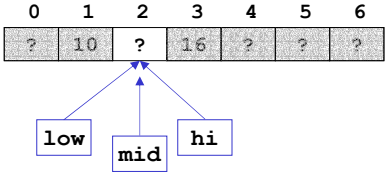
The middle element (10) is less than 11, so discard all elements to the right



Do this by setting `low=mid+1`

Binary Search - managing the sub-array

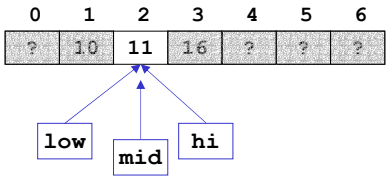
The sub-array contains just one element; both hi and low point to this element (as does mid)



low and hi point to the same element

Binary Search - managing the sub-array

- If 11 is in the array, it will be at this point. So after at most three comparisons we can find any element in the array.



low and hi point to the same element

Binary Search - the code

The pseudocode looks like

```
// find the middle index
mid = (low + hi)/2;
if(x < a[mid])
   hi = mid - 1; // to the left
else if(x > a[mid])
   low = mid + 1; // to the right
else // must be equal ...
   // ... found it!
```

Binary Search - the code

The whole lot would be contained within a while loop like:

```
low = 0;
hi = a.length - 1;
while(low <= hi)
{
      :
      :
}
```

```
int low = 0;
int hi = a.length - 1;
int index = -1;
while(low <= hi)
{
   int mid = (low + hi) / 2;
   if(x < a[mid])
      hi = mid - 1;
   else if(x > a[mid])
      low = mid + 1;
   else
   {  // must be equal, found it
      index = mid;
      break;
   }
}
```

Questions

- How many comparisons would be required if N was 1000000?
- How many for arbitary N?

How many for a million

1,000,000

500,000

250,000

125,000

62,500

:

:

1

4

20

logs

$2^{20} \approx$ million

$\log_2$million ≈ 20

=> divide million by 2 twenty times gives you 1.