# Lecture 13: binary search

# Searching an array for an item

How do we tell if an unsorted array of n items contains a particular item? (like int_array's contains)

*When can we stop early?*

*When would we have to look at all the items?*

*If there are n items, this takes O(n) time*

Would this become faster if the array were sorted smallest to largest?

*When can we stop early?*

*When would we have to look at all the items?*

*If there are n items, this takes O(n) time*

Can we do any better?

# Binary search

If the array's sorted, we can search it by saying:

Is the middle item the item we want?, (If so we found the item.)
Else, which side of the array should we look in?

Example code:
Given the array

1, 6, 17, 22, 24, 26, 27, 35, 51

Tell me what the middle element is.
Given the array

1, 6, 17, 22, 24, 26, 27, 35, 51, 98

Tell me what the middle element is.

# Binary search

Example code: Given the array

1, 6, 17, 22, 24, 26, 27, 35, 51

If we're looking for the 35 after checking the first middle, what's the sub-array to search next?

Let's start…

```
bool binary_contains1(const int* arr, unsigned int first,
                      unsigned int last, int target)
```

# Time bounds

Each time through the recursion, we throw out half the array

"Divide and conquer" at work…

So making the array twice as big will require checking one more middle element…

Making it 4 times as big will require checking 2 more middle elements…

# Binary search

Right now we are passing the whole array in each time

We could pass subarrays using pointer arithmetic…

bool binary_contains2(const int* arr, unsigned int size, int target)

How to find middle element?

How to specify new subarray?  Size is important.