Searching

Tiziana Ligorio

Today's Plan

Searching algorithms and their analysis

Searching

Looking for something!

In this discussion we will assume searching for an element in an array

Linear search

Most intuitive

Start at first position and keep looking until you find it

```
int linearSearch(int a[], int size, int value)
{

    for (int i = 0; i < size; i++)
    {
        if (a[i] == value) {
            return i;
        }
    }
    return-1;
}
```

How long does linear search take?

If you assume value is in the array and probability of finding it at any location is uniform, on **average n/2**

If value is not in the array (worst case) **n**

Either way it's $O(n)$

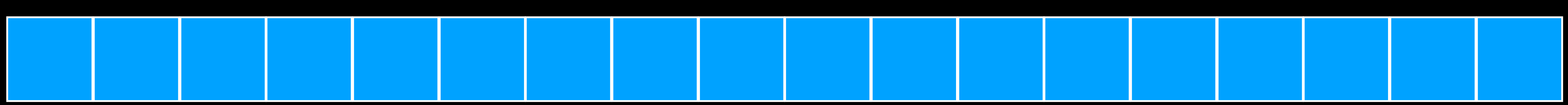
What if you know **array is sorted**?
Can you do better than linear search?

Lecture Activity

You are given a sorted array of integers.

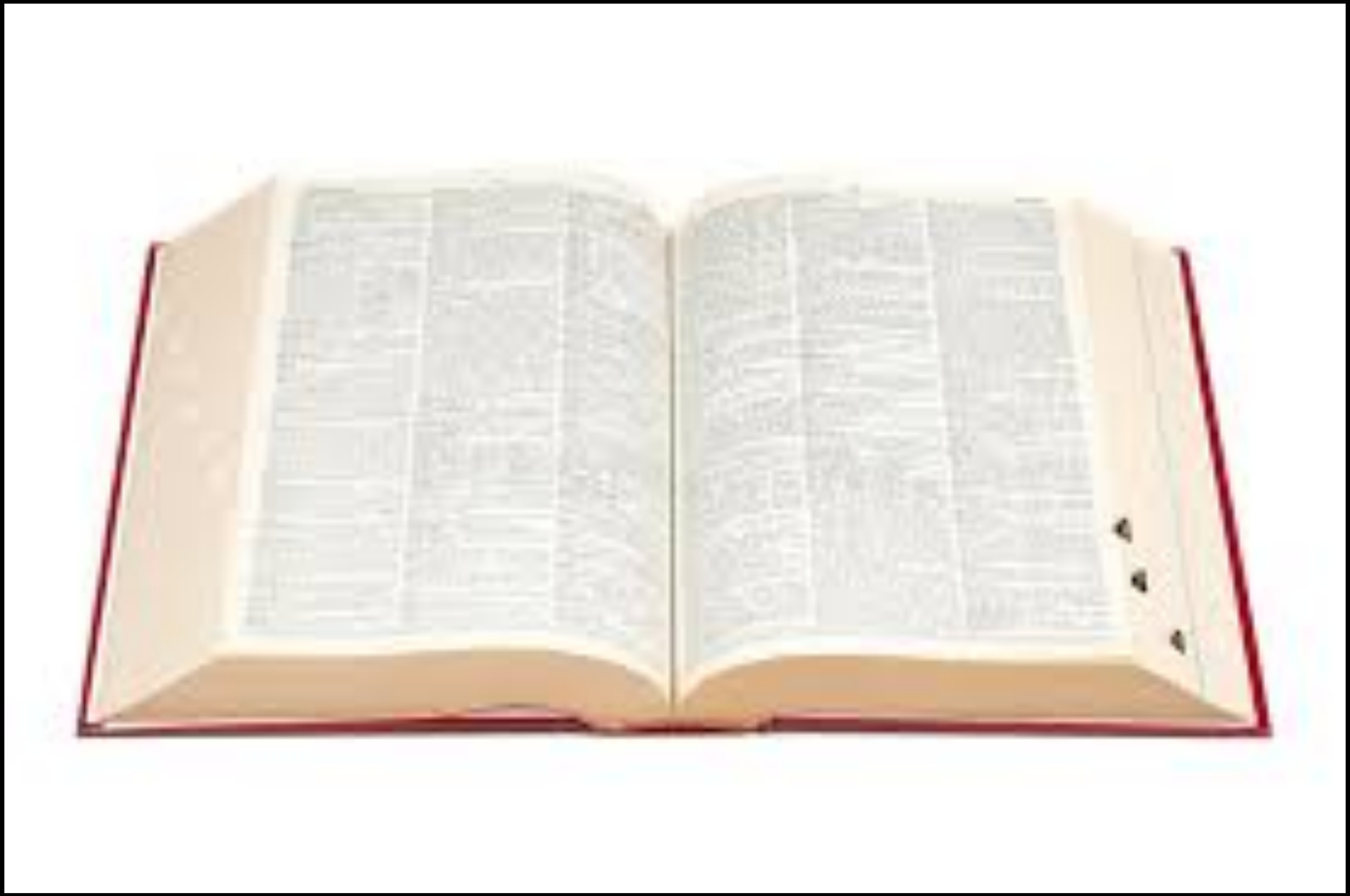
How would you search for 115? (try to do it in fewer than n steps: don't search sequentially)

You can write pseudocode or succinctly explain your algorithm



We have done this before!

When?

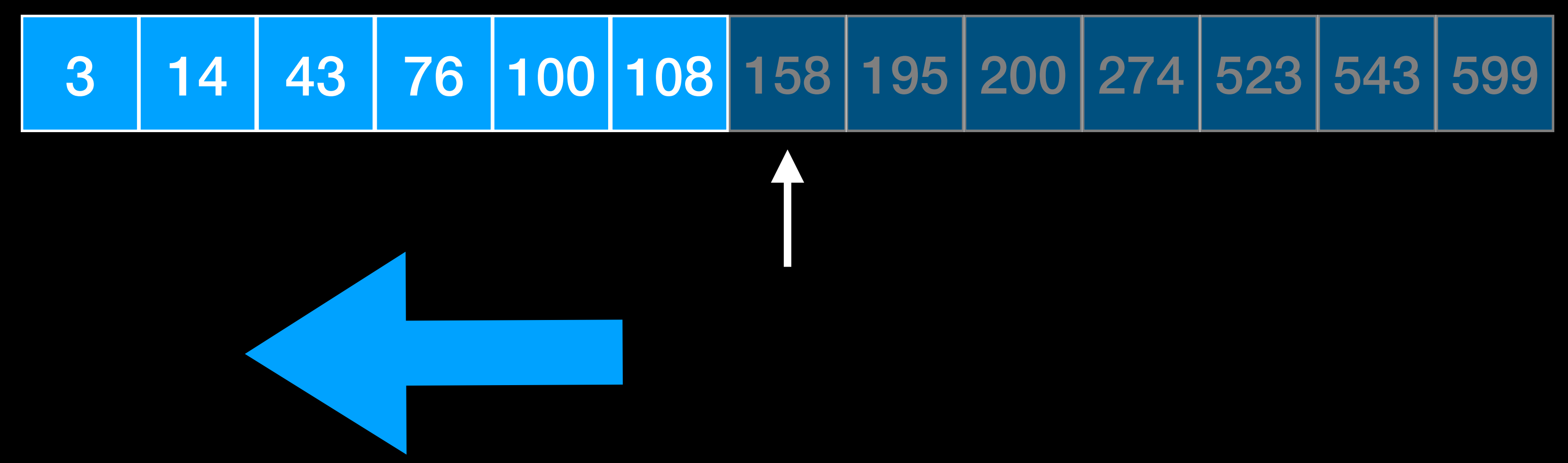


Look in ?

Binary Search

3	14	43	76	100	108	158	195	200	274	523	543	599

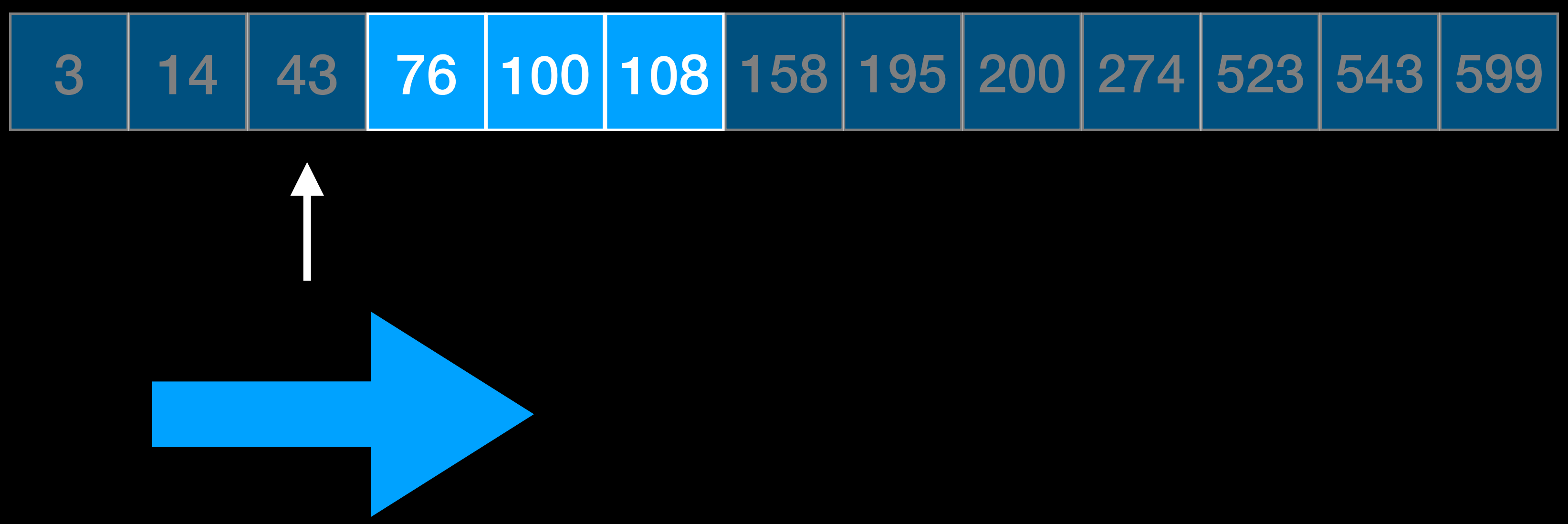
Binary Search



Binary Search

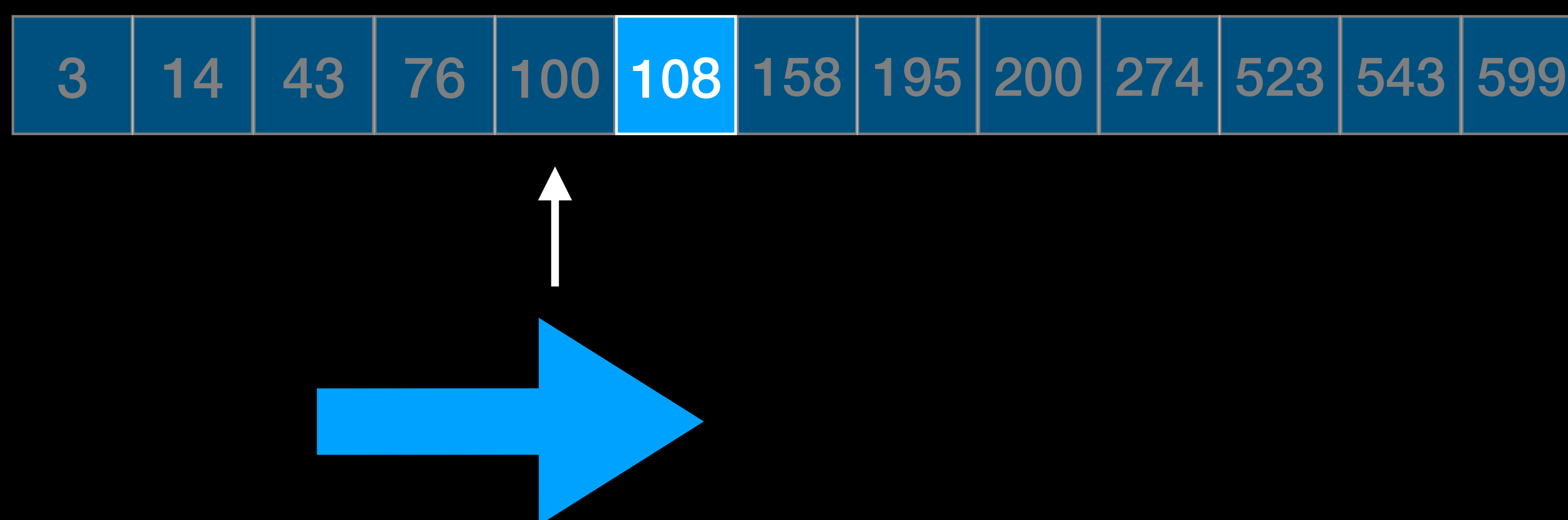
3	14	43	76	100	108	158	195	200	274	523	543	599

Binary Search

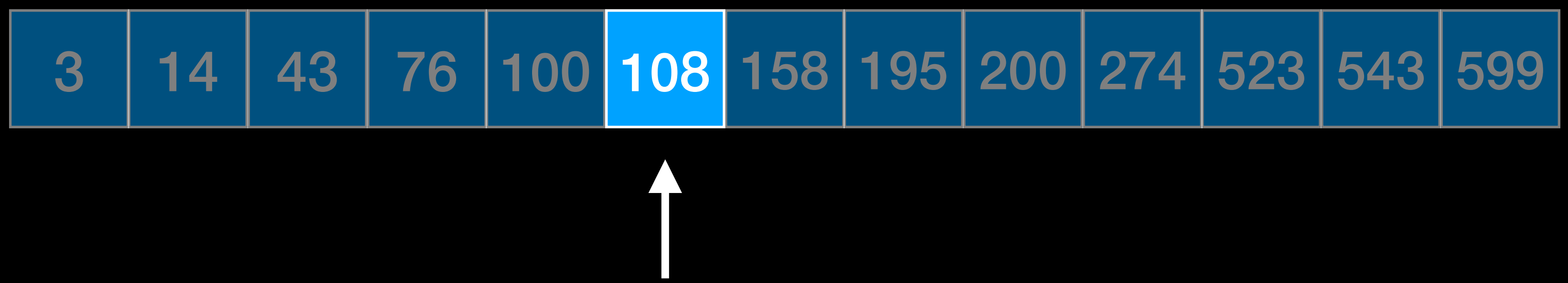


Binary Search

Binary Search



Binary Search



Binary Search

What is happening here?

Binary Search

What is happening here?

Size of search is **cut in half** at each step

Binary Search

What is happening here?

Size of search is **cut in half** at each step

Simplification: assume n is a power of 2 so it can be evenly divided in two parts

The running time

Let $T(n)$ be the running time and **assume $n = 2^k$**

$T(n) = T(n/2) + 1$

One comparison

Search lower OR upper half

Binary Search

What is happening here?

Size of search is **cut in half** at each step

Let $T(n)$ be the running time and **assume $n = 2^k$**

$T(n) = T(n/2) + 1$

$T(n/2) = T(n/4) + 1$

One comparison

Search lower OR upper half of n/2

Binary Search

What is happening here?

Size of search is **cut in half** at each step

Let $T(n)$ be the running time and **assume $n = 2^k$**

$T(n) = \boxed{T(n/2)} + 1$

$T(n/2) = T(n/4) + 1$

$T(n) = \boxed{T(n/4) + 1} + 1$

Binary Search

What is happening here?

Size of search is **cut in half** at each step

Let $T(n)$ be the running time and **assume $n = 2^k$**

$T(n) = T(n/2) + 1$

2^1 1

$T(n) = T(n/4) + 2$

2^2 2

. . .

Binary Search

What is happening here?

Size of search is **cut in half** at each step

Let $T(n)$ be the running time and **assume $n = 2^k$**

$T(n) = T(n/2) + 1$

$T(n) = T(n/4) + 2$

. . .

$T(n) = T(n/2^k) + k$

Binary Search

What is happening here?

Size of search is **cut in half** at each step

Let $T(n)$ be the running time and **assume $n = 2^k$**

$T(n) = T(n/2) + 1$

$T(n) = T(n/4) + 2$

. . .

$T(n) = T(n/2^k) + k$

$T(n) = T(1) + \log_2(n)$

The number to which I need to raise 2 to get n
And we said $n = 2^k$

$n/n = 1$

Binary Search

What is happening here?

Size of search is **cut in half** at each step

Let $T(n)$ be the running time and **assume $n = 2^k$**

$T(n) = T(n/2) + 1$

$T(n) = T(n/4) + 2$

. . .

$T(n) = T(n/2^k) + k$

$T(n) = T(1) + \log_2(n)$

Binary search is $O(\log(n))$