Searching

Tiziana Ligorio
Hunter College of The City University of New York

Today's Plan

Midterm discussion

Searching algorithms and their analysis

Announcements

- MIDTERM:
  - Solutions will be posted after we finish grading
  - Regrade requests
  - No curve, but question-level adjustments may occur as a result of regrading

Searching

Looking for something!

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

Linear search

Most intuitive

Start at first position and keep looking until you find it

```
template <class Comparable>
int linearSearch(const std::vector<Comparable>& a, const Comparable& value)
{

    for (int i = 0; i < a.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 $O(n)$

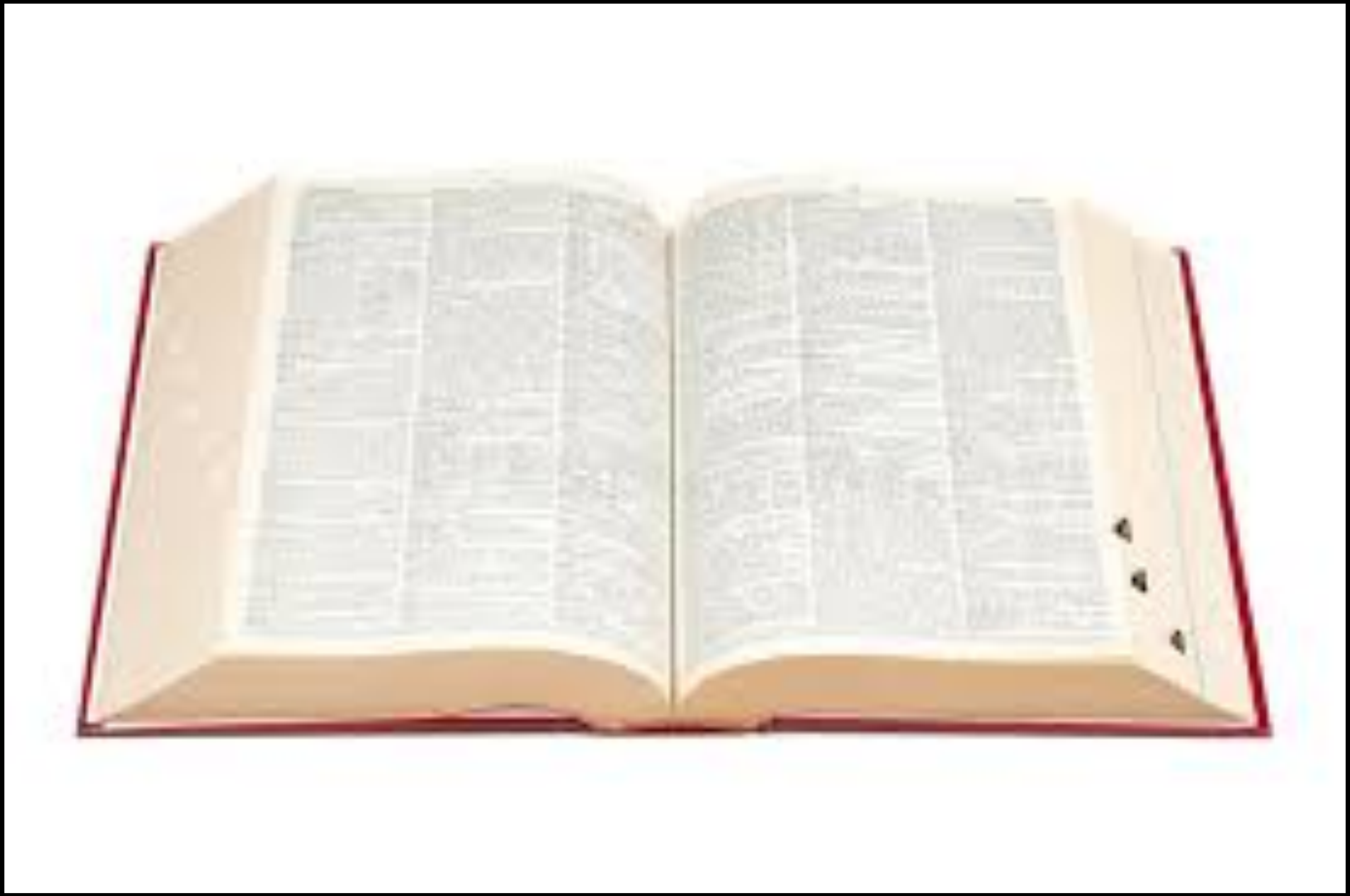
What if you know **array is sorted**?
Can you do better than linear search?

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

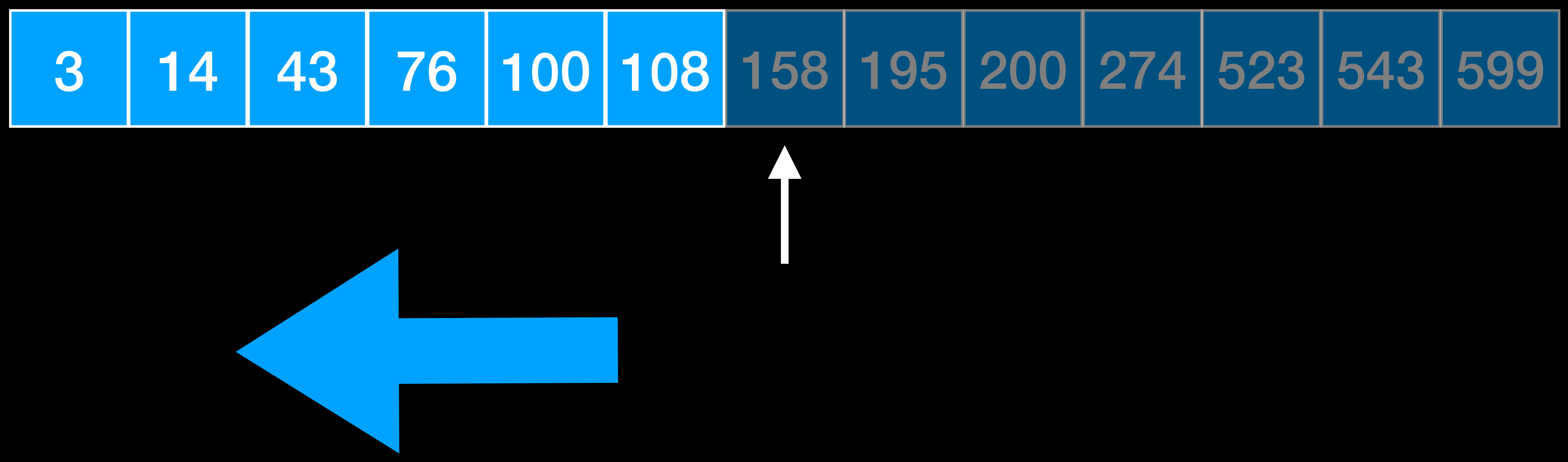


Look in ?

Binary Search

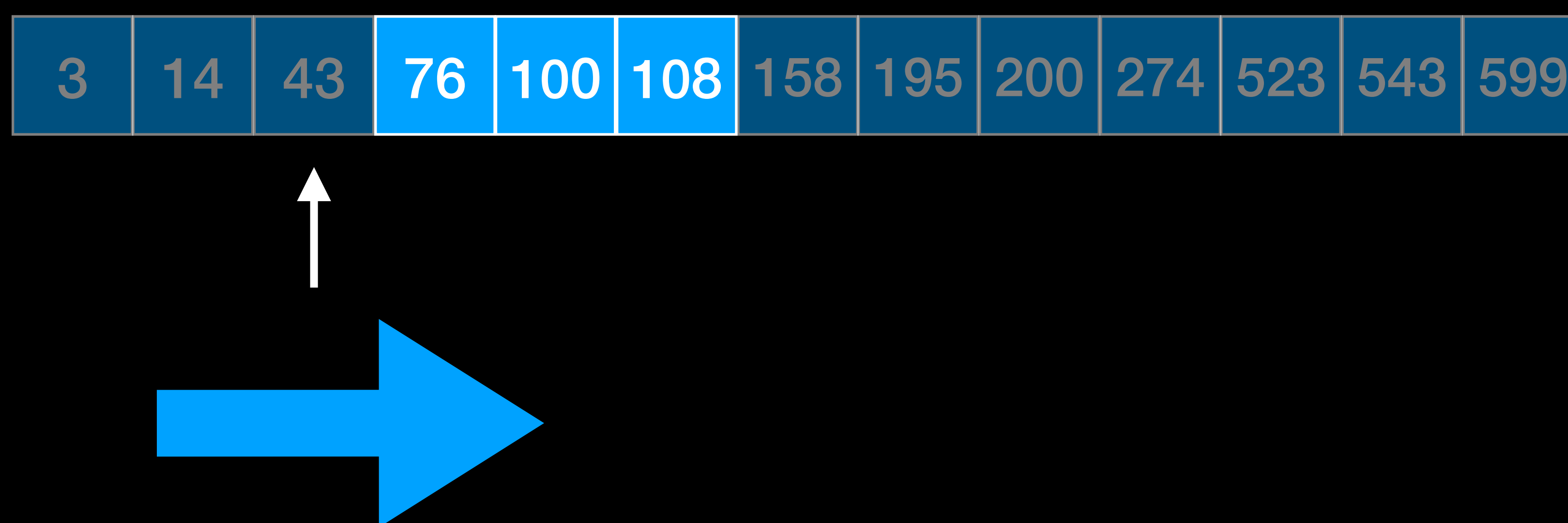
3	14	43	76	100	108	158	195	200	274	523	543	599

Binary Search

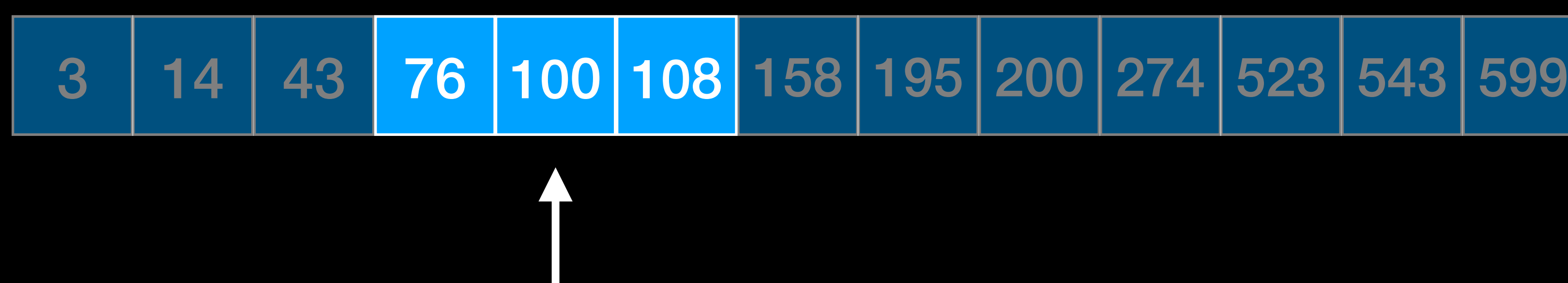


Binary Search

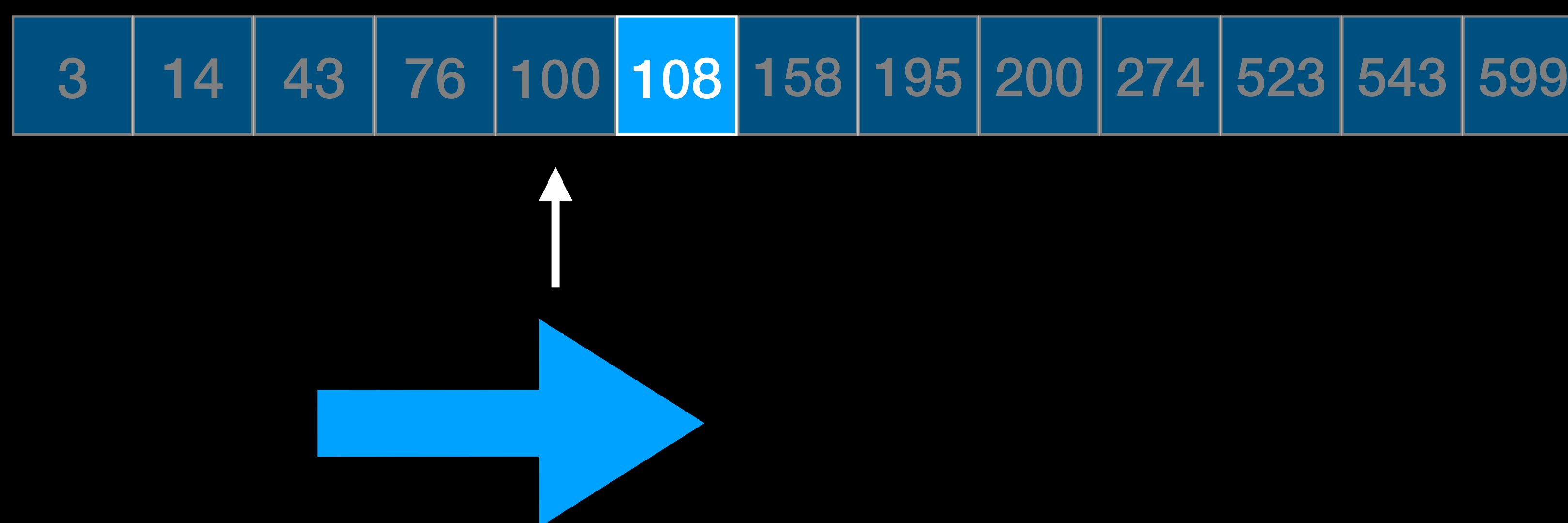
Binary Search



Binary Search



Binary Search



Binary Search

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

```cpp
template <class Comparable>
int binarySearch(const std::vector<Comparable>& v, const Comparable& x)
{
    int low = 0, high = v.size() - 1;

    while(low <= high)
    {
      int mid = (low + high) / 2;
      if(v[mid] < x)
        low = mid + 1; //search upper half
      else if (v[mid] > x)
        high = mid - 1; // search lower half
      else
        return mid;  //found

    }
    return -1; //not found
}
```

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

low high

```
template <class Comparable>
int binarySearch(const std::vector<Comparable>& v, const Comparable& x)
{
    int low = 0, high = v.size() - 1;

    while(low <= high)
    {
      int mid = (low + high) / 2;
      if(v[mid] < x)
        low = mid + 1; //search upper half
      else if (v[mid] > x)
        high = mid - 1; // search lower half
      else
        return mid;  //found

    }
    return -1; //not found
}
```

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

```
template <class Comparable>
int binarySearch(const std::vector<Comparable>& v, const Comparable& x)
{
    int low = 0, high = v.size() - 1;

    while(low <= high)
    {
      int mid = (low + high) / 2;
      if(v[mid] < x)
        low = mid + 1; //search upper half
      else if (v[mid] > x)
        high = mid - 1; // search lower half
      else
        return mid;  //found

    }
    return -1; //not found
}
```

O(?)

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

low high mid

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

The running time

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

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

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

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 $\quad$ 1

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

. . .

2^2 $\quad$ 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 $O(\log(n))$

Sorting

Rearranging a sequence into increasing (decreasing) order!

Several approaches

Can do it in many ways

What is the best way?

Let's find out using Big-O

Lecture Activity

Write **pseudocode** to sort an array.

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

There are many approaches to sorting
We will look at some comparison-based approaches here

Next time: Sorting