#### Introduction

- two of the most fundamental concepts in computer science are, given an array of values:
  - search through the values to see if a specific value is present and, if so, where
  - sort the values in nondecreasing order

### Introduction

- two of the most fundamental concepts in computer science are, given an array of values:
  - search through the values to see if a specific value is present and, if so, where
  - sort the values in nondecreasing order
- as a computer scientist, you must be able to understand and program several different algorithms for each of these tasks

### Introduction

- two of the most fundamental concepts in computer science are, given an array of values:
  - search through the values to see if a specific value is present and, if so, where
  - sort the values in nondecreasing order
- as a computer scientist, you must be able to understand and program several different algorithms for each of these tasks
- in all of these slides, "array" is a generic term
- it means either an old-fashioned C-array or a modern STL vector

- an array that contains an arbitrary number of values
- each value is in a specific array location, but the values are not sorted

17	23	5	11	2	29	3
----	----	---	----	---	----	---

- an array that contains an arbitrary number of values
- each value is in a specific array location, but the values are not sorted

17	23	5	11	2	29	3
----	----	---	----	---	----	---

- search for the value 11
- do steps on 17, 23, 5, 11 (4 steps)
- return 3 (the position where 11 was found)

- an array that contains an arbitrary number of values
- each value is in a specific array location, but the values are not sorted



- search for the value 11
- do steps on 17, 23, 5, 11 (4 steps)
- return 3 (the position where 11 was found)
- search for the value 7
- do steps on 17, 23, 5, 11, 2, 29, 3 (7 steps) return a notfound indicator



#### Linear Search

Program 8-1, page 464, a function to do this

```
int linearSearch(const int arr[], int size, int value);
```

- up to size elements of the array arr are searched for the existence of value
- if value is found within the first size elements of arr, the first position where value is encountered is returned by the function
- if value is not found, the sentinel value -1 is returned

### Linear Search

Program 8-1, page 464, a function to do this

int linearSearch(const int arr[], int size, int value);

- up to size elements of the array arr are searched for the existence of value
- if value is found within the first size elements of arr, the first position where value is encountered is returned by the function
- ullet if value is not found, the sentinel value -1 is returned
- the search is called linear search because the algorithm searches along the line of array elements, one by one, starting at the beginning, trying to find value

## Searching a Sorted List

- linear search works perfectly on an unsorted array of values
- for unsorted values, it is the only practical searching algorithm

# Searching a Sorted List

- linear search works perfectly on an unsorted array of values
- for unsorted values, it is the only practical searching algorithm
- linear search also works perfectly on a sorted array of values
- however, is it not the only search algorithm for sorted values
- furthermore, it is not a even a good algorithm in this case

# Linear Search Analysis

- it is traditional to use *n* to denote the number of elements of an array (this is the same thing as vector's size)
- do the following experiment (please review the source code for ths simulation attached with the lecture):
  - 1. put 1,000 random values, chosen from 0 10,000, into a vector
  - 2. at random, pick a value in the range 0 10,000 and
    - 2.1 see if it is in the vector or not
    - 2.2 count how many steps it takes to find out
  - 3. repeat step 2 10,000 times
  - 4. calculate the average number of steps from step 2

## Linear Search Analysis

results:

```
Total hits: 930 Total misses: 9070 Minimum steps: 1 Maximum steps: 1000 Average number of steps: 954.747
```

- with 1,000 random values from 0 to 10,000
- the average number of steps is about 955
- have to search almost the entire array almost every time

### Linear Search

#### Linear Search Pros

- very easy algorithm to understand
- easy algorithm to code correctly
- the only practical algorithm for unsorted values

#### Linear Search Cons

- very inefficient for sorted values
- must examine n/2 elements on average for a value that is in the array
- must examine all n elements for a value not in the array

### Linear Search

#### Linear Search Pros

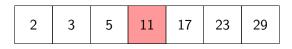
- very easy algorithm to understand
- easy algorithm to code correctly
- the only practical algorithm for unsorted values

#### Linear Search Cons

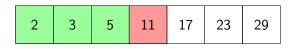
- very inefficient for sorted values
- must examine n/2 elements on average for a value that is in the array
- must examine all n elements for a value not in the array
- for sorted values, we can do better

2	3	5	11	17	23	29
---	---	---	----	----	----	----

1. divide the range of elements to search into 3:



- 1. divide the range of elements to search into 3:
  - 1.1 the very middle element



- 1. divide the range of elements to search into 3:
  - 1.1 the very middle element
  - 1.2 the elements to the left of middle



- 1. divide the range of elements to search into 3:
  - 1.1 the very middle element
  - 1.2 the elements to the left of middle
  - 1.3 the elements to the right of middle



- 1. divide the range of elements to search into 3:
  - 1.1 the very middle element
  - 1.2 the elements to the left of middle
  - 1.3 the elements to the right of middle
- 2. if the range of elements is empty, return not-found sentinel



- 1. divide the range of elements to search into 3:
  - 1.1 the very middle element
  - 1.2 the elements to the left of middle
  - 1.3 the elements to the right of middle
- 2. if the range of elements is empty, return not-found sentinel
- 3. else if the searched-for value is the middle element, you've found it and you're done



- 1. divide the range of elements to search into 3:
  - 1.1 the very middle element
  - 1.2 the elements to the left of middle
  - 1.3 the elements to the right of middle
- 2. if the range of elements is empty, return not-found sentinel
- 3. else if the searched-for value is the middle element, you've found it and you're done
- 4. else if the searched-for value is **smaller** than the middle element, repeat step 1 on the **left half**



- 1. divide the range of elements to search into 3:
  - 1.1 the very middle element
  - 1.2 the elements to the left of middle
  - 1.3 the elements to the right of middle
- 2. if the range of elements is **empty**, return not-found sentinel
- 3. else if the searched-for value is the middle element, you've found it and you're done
- 4. else if the searched-for value is **smaller** than the middle element, repeat step 1 on the **left half**
- 5. else repeat step 1 on the right half

2 3 5	11	17	23	29	
-------	----	----	----	----	--

- search for 11
- do a comparison on 11 (1 step)
- return 3 (the position where 11 was found) done

2	3	5	11	17	23	29
---	---	---	----	----	----	----

- search for 11
- do a comparison on 11 (1 step)
- return 3 (the position where 11 was found) done
- search for 7
- do a comparison on 11, 3, 5 (3 steps) report not-found —
- done

## Binary Search Implementation

• Please review the attached binary search algorithm:

program\_8\_2.cpp

#### Animation:

https://www.cs.usfca.edu/~galles/visualization/Search.html

.

### Binary Search Analysis

- do the same simulation with binary search as with linear search
- 10,000 times, search for a random number 0 1,000
- results:

Total hits: 971 Total misses: 9029 Minimum steps: 1 Maximum steps: 24 Average number of steps: 9.876

remember the linear results:

Total hits: 930 Total misses: 9070 Minimum steps: 1 Maximum steps: 1000 Average number of steps: 954.747

- on average linear search takes hundred times as many steps as the binary search!
- In this case, binary search takes at most ten steps



# Binary Search Analysis

- let's dig a little deeper into exactly how long the two searches take
- linear search starts with a search space of size n
- with each comparison, the search space decreases by one
- if the value is not in the array, clearly it takes n steps to decrease the search space to size zero and determine notfound

• binary search also starts with a search space of size n

# Binary Search Analysis

- let's dig a little deeper into exactly how long the two searches take
- linear search starts with a search space of size n
- with each comparison, the search space decreases by one
- if the value is not in the array, clearly it takes n steps to decrease the search space to size zero and determine notfound
- binary search also starts with a search space of size n
- with each comparison, the search space decreases by half
- how many steps does it take to repeatedly divide a number by 2 until you get to zero?
- the term for this is logarithm

## Logarithms

- logarithms are particularly important in algorithm analysis
- computer scientists deal mainly with base-2 logarithms
- the simplest working definition of a base-2 logarithm is

### Base-2 Logarithm

How many times can you divide a number n by 2 using integer division before the result is 1 or 0?

- the answer to this question is approximately the base-2 logarithm of n
- you can estimate base-2 logarithms directly from the powers of 2 table

### Powers of 2

you should learn all the powers of 2 from 0 to 10

$$2^{0} = 1$$
 $2^{1} = 2$ 
 $2^{2} = 4$ 
 $2^{3} = 8$ 
 $2^{4} = 16$ 
 $2^{5} = 32$ 
 $2^{6} = 64$ 
 $2^{7} = 128$ 
 $2^{8} = 256$ 
 $2^{9} = 512$   $log_{2}(512) = 9$ 
 $2^{10} = 1024$ 

- remember the simulation results searching in an array with 1,000 elements
- the most steps ever needed was 10
- the base-2 logarithm of 1,000 is approximately 10
- Hence, the number of steps needed for binary search for an array of size n is at most log 2 n

- remember the simulation results searching in an array with 1,000 elements
- the most steps ever needed was 10
- the base-2 logarithm of 1,000 is approximately 10
- the number of steps needed for binary search for an array of size n is at most log 2 n
- compare this to linear search, which needs at most 1,000 steps for a 1,000-element array

#### Binary Search Pros

- much more efficient than linear search
- requires at most log<sub>2</sub> n comparisons
- easy algorithm to understand and code correctly

#### Binary Search Cons

 requires array elements to already be sorted

 clearly, sorting is an issue, and we will discuss sorting in our next lecture