Searching
Overview

We often need to search for an item in a collection

- *Is this student in this recitation roster?*
- *Is this username in our user database?*
- *Is there any data point in our dataset that matches this description?*

In this module, we will learn about how to search for an element in an array.
Learning Objectives

- To be able to use \textbf{linear search} to find an element inside an array
- To be able to use \textbf{binary search} to find an element inside an array
- To be able to know when to use linear search and when to use binary search
Linear Search

Used to search for a value (the target) in an unsorted array

- Use a loop to iterate over the values
- Start at the first element and move to the next element until the target is found
- Returns the position of the target if it was found in the array, or -1 if the target was not found in the array
Linear Search Example

Searching for 82 at position 0

(this image is a link)
public static int linearSearch(int[] A, int k) {
    // for each index in the array...
    for (int i = 0; i < A.length; i++) {
        // compare the current value (A[i]) to the target k
        if (A[i] == k) {
            return i; // if there's a match, stop & return THE INDEX
        }
    }
    // we only get to this line if we didn't find anything!
    return -1;
}
Binary Search

Used to search for a target value in a **sorted array only**

- Compares the target with the value at the middle index (middle element)
  - If the middle element is the target element, then we're done!
  - If the target is less than the middle element, then we search for the target in the **left half of the array** (the positions before the middle element)
  - If the target is greater than the middle element, then we search the target in the **right half of the array** (the positions after the middle element)
- Repeat on the remaining search area of the array until
  - the element is found
  - there is no feasible search area left
Binary Search

Returns the position of the middle element if it is equal to the target

Returns -1 if the target was not found in the array
Binary Search

Searching for "Dustin" in the array names!

<table>
<thead>
<tr>
<th>Caryn</th>
<th>Debbie</th>
<th>Dustin</th>
<th>Elliot</th>
<th>Jacquie</th>
<th>Jon</th>
<th>Rich</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

- middle = (low + high) / 2 = 3
- names[middle] is "Elliot", which comes after "Dustin" alphabetically.
- So, if "Dustin" is present, it must be between positions 0 and middle - 1.
Binary Search

Searching for "Dustin" in the array names!

<table>
<thead>
<tr>
<th>Caryn</th>
<th>Debbie</th>
<th>Dustin</th>
<th>Elliot</th>
<th>Jacquie</th>
<th>Jon</th>
<th>Rich</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>low</td>
<td>middle</td>
<td>high</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- middle = (low + high) / 2 = 1
- names[middle] is "Debbie", which comes before "Dustin" alphabetically.
- So, if "Dustin" is present, it must be between positions middle + 1 and 2.
**Binary Search**

Searching for "Dustin" in the array `names`!

<table>
<thead>
<tr>
<th>Caryn</th>
<th>Debbie</th>
<th>Dustin</th>
<th>Elliot</th>
<th>Jacquie</th>
<th>Jon</th>
<th>Rich</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

- `middle = (low + high) / 2 = 2`
- `names[middle] is "Dustin", which is the target element! So, we return middle.`
Searching for 26 at position 10
Left bound at position 0, right bound at position 20

(this image is a link)
public static int binarySearch(int[] array, int target) {
    // to start, the search area is the entire array, [0, array.length - 1]
    int lowIndex = 0;
    int highIndex = array.length - 1;

    // there's a valid search area as long as lowIndex <= highIndex
    while (lowIndex <= highIndex) {
        int middleIndex = (lowIndex + highIndex) / 2;
        if (target < array[middleIndex]) {
            highIndex = middleIndex - 1; // throw away *right* half
        } else if (target > array[middleIndex]) {
            lowIndex = middleIndex + 1; // throw away *left* half
        } else { // happens when target == array[middleIndex]
            return middleIndex; // success!
        }
    }
    return -1; // couldn't find element, so return default -1
}
Linear Search vs. Binary Search

- Binary search is faster than linear search 😊凉快
  - On average, it will require fewer iterations of the search loop
  - (when is binary search not faster than linear search?)
- Binary search runs on sorted data 😞生气 😁
  - why?
- Linear search runs on unsorted data 😊😊😊
# Linear Search vs. Binary Search

**Runtime analysis:** how many comparisons will it take to determine that the target is not in the array?

<table>
<thead>
<tr>
<th>Length of the array</th>
<th>Linear Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>7</td>
</tr>
</tbody>
</table>