Exam Advice

Exam 1 Details

- Wednesday, October 9 in this classroom at this time.
  - If you have SDS exam accommodations, **you are responsible for scheduling that with Weingarten.**
- TA Exam Review Session on the evening of October 7
- Covers anything in lecture and homeworks up until (and including) today.
  - Don't have to memorize every PennDraw function, but you should know all of the functions used in `MyHouse.java` and you should know how to set up an animation loop
  - Records on Wednesday are not included

How to Study

- For now, finish HW3
- After Wednesday, take inventory of all of the "tricks" and techniques you used in the homeworks
  - e.g. animation loops, parsing `args`, reading user input, shifting chars
- Enjoy Fall Break
- **PRINT OUT** and take either *Fall 2023* or *Fall 2023 Practice Exam* timed for 60 minutes, pencil & paper, no notes
  - Grade yourself, use your performance to guide the rest of your practice.
  - More practice questions, go to OH, SRS, etc.

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 **linear search** to find an element inside an array
- To be able to use **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

Problem: Search

Formally, given a **sequence of values** and a **target value**, we want to determine if the target value is in the sequence, and if so, where it is located.

- in our case, the "sequence of values" is an array
- the "target value" is the value we are searching for
- the location is the index of the value in the array, or `-1` if it's not present.

Concept: The "Feasible Region"

In any problem, the **feasible region** is the name for the set of possible values that might be a solution.

- In the context of search, the feasible region refers to the set of indices in the array that might contain the target value.
- A set of indices is functionally a region of the array where the target value might be found.

In array search, we repeatedly reduce the feasible region until we find the target value, or until we determine that the target value is not present in the array.

The Case of the Missing Keys

You're getting ready to leave your apartment in the morning and you realize you can't find your keys! You know they must be somewhere in the apartment because you let yourself in last night. So, you go hunting for your keys throughout each of your rooms...

- What's a reasonable search strategy?
- What is the feasible region for that search, and how does it change over time?

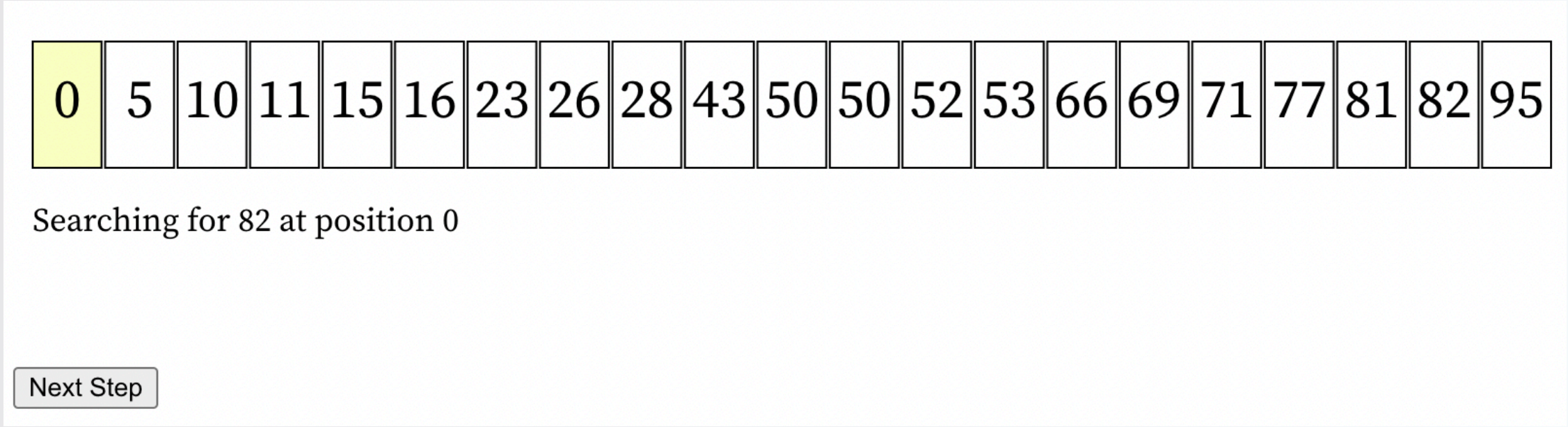
Linear Search

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

With each iteration, we reduce the feasible region by one element.

Linear Search Example



(this image is a link)

Linear Search

```java
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;
}
```

Linear Search: Thinking Critically

How many iterations of the for loop will we need if...

- the target is the first element in the array?
- the target is the 10th element in the array?
- the target is not in the array?

Linear Search: Thinking Critically

How many iterations of the for loop will we need if...

- the target is the first element in the array? **1**
- the target is the 10th element in the array? **10**
- the target is not in the array? **A.length**

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`!

Caryn	Debbie	Dustin	Elliot	Jacquie	Jon	Rich
0	1	2	3	4	5	6
low			middle			high

- `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`!

Caryn	Debbie	Dustin	Elliot	Jacquie	Jon	Rich
0	1	2	3	4	5	6
low	middle	high				

- `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`!

Caryn	Debbie	Dustin	Elliot	Jacquie	Jon	Rich
0	1	2	3	4	5	6
		low, middle, high				

- `middle = (low + high) / 2 = 2`
- `names[middle]` is `"Dustin"`, which is the target element! So, we return `middle`.

Binary Search: Searching for an Element not Present

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

Caryn	Debbie	Dustin	Elliot	Jacquie	Jon	Rich
0	1	2	3	4	5	6
low			middle			high

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

Binary Search: Searching for an Element not Present

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

Caryn	Debbie	Dustin	Elliot	Jacquie	Jon	Rich
0	1	2	3	4	5	6
low	middle	high				

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

Binary Search: Searching for an Element not Present

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

Caryn	Debbie	Dustin	Elliot	Jacquie	Jon	Rich
0	1	2	3	4	5	6
		low, middle, high				

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

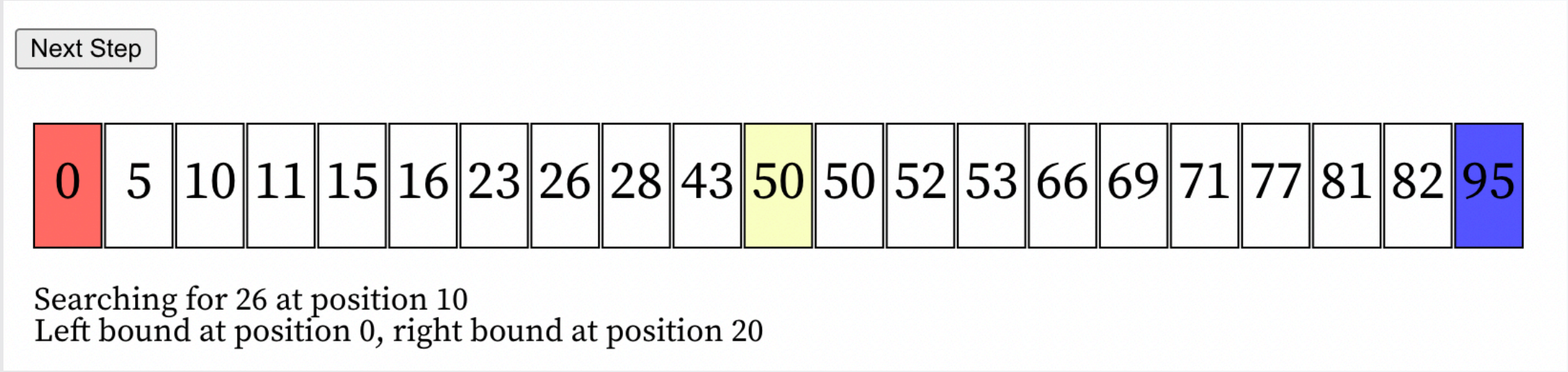
Binary Search: Searching for an Element not Present

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

Caryn	Debbie	Dustin	Elliot	Jacquie	Jon	Rich
0	1	2	3	4	5	6
	high	low				

- `high` is now less than `low`. The "feasible search area" is now totally empty.
- So, we return `-1` to indicate that the target was not found in the array.

Binary Search, Interactive



(this image is a link)

Binary Search

```java
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 😄🎉🆒
  - Per iteration, binary search shrinks the feasible region by half the remaining elements, linear search only by one element.
  - On average, binary search requires fewer iterations of the search loop
  - (when is binary search not faster then linear search?)
- Binary search runs on sorted data 😢😠❗
  - 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?

Length of the array	Linear Search	Binary Search
2	2	2
4	4	3
8	8	4
16	16	5
100	100	7