# Searching

2024 Winter APS 105: Computer Fundamentals
Jon Eyolfson

Lecture 34
1.0.2

## When You Have a Lot of Data, You Need to Search

We previously searched for nodes in a linked list, today we'll focus on arrays

We want to find the index of a value in an array
(we'll return -1 if not found)

## The First Searching Algorithm is Linear Search

We iterate through the array looking for the value,
if we find a match, return the current index

## An Implementation of Linear Search

```
int linearSearch(int array[], int arrayLength, int val) {
    for (int i = 0; i < arrayLength; ++i) {
        if (array[i] == val) {
            return i;
        }
    }
    return -1;
}
```

## However, What If We Searched Sorted Data?

Do we search to find the minimum or maximum value in a sorted array?

## We Can Check the Midpoint and Cut Our Search in Half

If the array is sorted, check the middle value
  If the middle value is greater, then search the left half
  If the middle value is less, then search the right half

We can keep searching while we have valid array bounds,
  or we find the value (this is called binary search)

# Implementation of Iterative Binary Search

```
int binarySearch(int array[], int arrayLength, int val) {
    int low = 0;
    int high = arrayLength - 1;
    while (low <= high) {
        int mid = (high + low) / 2;
        if (val == array[mid]) {
            return mid;
        }
        else if (val > array[mid]) {
            low = mid + 1;
        }
        else {
            high = mid - 1;
        }
    }
    return -1;
}
```

## Can We Write Our Binary Search Another Way?

If the array is sorted, check the middle value
- If the middle value is greater, then search the left half
- If the middle value is less, then search the right half

This sounds recursive too! What would our base cases be?

# Implementation of Recursive Binary Search

```
int recursiveBinarySearchHelper(int array[], int low, int high, int val) {
    /* Base cases */
    if (low > high) { return -1; }
    int mid = (low + high) / 2;
    if (val == array[mid]) { return mid; }
    /* Rescursive step */
    else if (val > array[mid]) {
        return recursiveBinarySearchHelper(array, mid + 1, high, val);
    }
    else {
        return recursiveBinarySearchHelper(array, low, mid - 1, val);
    }
}

int recursiveBinarySearch(int array[], int arrayLength, int val) {
    return recursiveBinarySearchHelper(array, 0, arrayLength -1, val);
}
```

## We're Not Guaranteed What Matches

If there are multiple matching values, what should the matching index be?

How can we change the code to always get the first matching index?

# Modifying Binary Search to Return the First Matching Index

```
int binarySearchFirst(int array[], int arrayLength, int val) {
    int low = 0;
    int high = arrayLength - 1;
    while (low <= high) {
        int mid = (high + low) / 2;
        if ((mid == 0 || val > array[mid - 1]) && val == array[mid]) {
            return mid;
        }
        else if (val > array[mid]) {
            low = mid + 1;
        }
        else {
            high = mid - 1;
        }
    }
    return -1;
}
```

## What If We Want to Know the Last Matching Index?

Can we re-write our code to return the last index instead?

# Modifying Binary Search to Return the Last Matching Index

```
int binarySearchLast(int array[], int arrayLength, int val) {
    int low = 0;
    int high = arrayLength - 1;
    while (low <= high) {
        int mid = (high + low) / 2;
        if ((mid == arrayLength - 1 || val < array[mid + 1]) && val == array[mid]) {
            return mid;
        }
        else if (val >= array[mid]) { /* Tricky change! */
            low = mid + 1;
        }
        else {
            high = mid - 1;
        }
    }
    return -1;
}
```

## We Can Use Both Functions to Find the Number of Matches

```
void binarySearch(int array[], int arrayLength, int val) {
    int first = binarySearchFirst(array, arrayLength, val);
    if (first == -1) {
        printf("%d not found\n", val);
        return;
    }
    int last = binarySearchLast(array, arrayLength, val);
    int matches = last - first + 1;
    printf("%d found (%d matches)\n", val, matches);
}
```

## We Can Search a Sorted Array Faster

If we have an unsorted array, we need to use linear search

However, if we search a sorted array, we can use binary search