Adam Blank **Lecture 15** **Winter 2015**

CSE 143

Computer Programming II

Searching

Outline

1 Linear Search

2 Searching in a Sorted Array

3 Binary Search

Searching An Array 1

Idea
Check each index from left to right until we find the element we're looking for. If we've tried everything, say we couldn't find it.

Code
```java
public static boolean linearSearch(int[] a, int val) {
    for (int try : a) {
        if (try == val) {
            return true;
        }
    }
    return false;
}
```

Searching A Sorted Array 2

Now, make the extra assumption that **the array is sorted**.

Idea
Check each index from left to right until we find the element we're looking for **or** an element **larger than the one we're looking for**. If found a bigger element, say we couldn't find it.

Code
```java
public static boolean sortedLinearSearch(int[] a, int val) {
    for (int try : a) {
        if (try == val) {
            return true;
        }
        if (try > val) {
            return false;
        }
    }
    return false;
}
```

Searching Even Faster 3

Search for 24 in a

a: | ? | ? | ? | ? | ? | ? | ? |
   a[0] a[1] a[2] a[3] a[4] a[5] a[6]

a: | ? | ? | ? | 50 | ? | ? | ? |
   a[0] a[1] a[2] a[3] a[4] a[5] a[6]

a: | ? | ? | ? | X | X | X | X |
   a[0] a[1] a[2] a[3] a[4] a[5] a[6]

a: | ? | 10 | ? | X | X | X | X |
   a[0] a[1] a[2] a[3] a[4] a[5] a[6]

a: | X | X | ? | X | X | X | X |
   a[0] a[1] a[2] a[3] a[4] a[5] a[6]

a: | X | X | 12 | X | X | X | X |
   a[0] a[1] a[2] a[3] a[4] a[5] a[6]

So, 24 is not in a!

Runtime of Binary Search

Observation

Each time we check an element in the array, Binary Search rules out **half of the remaining possibilities**. If the array is of length n, we can do this $\log_2(n)$ times before getting to one element. So, Binary Search is $\mathcal{O}(\log(n))$.

Using Binary Search in Java

- `Arrays.binarySearch(int[] a, int k);`
- `Collections.binarySearch(int[] a, int k);`

Coding Binary Search

```java
private static boolean binarySearch(List<Integer> list, int value,
                                    int lo, int hi) {
    /* Handle the case where the list is empty */
    if (lo == hi) {
        return false;
    }

    /* The base case is when there is only one element left to check */
    if (lo == hi - 1) {
        return list.get(lo) == value;
    }

    /* Otherwise, figure out of the answer is on the left
     * or the right, and recurse */
    int mid = (lo + hi)/2;
    if (value < list.get(mid)) {
        /* Since our value is smaller, get rid of the right side
         * of the array (including mid) */
        return binarySearch(list, value, lo, mid);
    }
    else {
        /* Since our value is bigger or equal, get rid of everything
         * smaller than mid */
        return binarySearch(list, value, mid, hi);
    }
```

Coding Generic Binary Search

```java
private static <T extends Comparable<T>> boolean binarySearch(
    List<T> list, T value, int lo, int hi) {
    /* Handle the case where the list is empty */
    if (lo == hi) {
        return false;
    }

    /* The base case is when there is only one element left to check */
    if (lo == hi - 1) {
        return list.get(lo).equals(value);
    }

    /* Otherwise, figure out of the answer is on the left
     * or the right, and recurse */
    int mid = (lo + hi)/2;
    if (value.compareTo(list.get(mid)) < 0) {
        /* Since our value is smaller, get rid of the right side
         * of the array (including mid) */
        return binarySearch(list, value, lo, mid);
    }
    else {
        /* Since our value is bigger or equal, get rid of everything
         * smaller than mid */
        return binarySearch(list, value, mid, hi);
    }
```

Some Searching Tips!

- Understand how to take advantage of the fact that an array is sorted when searching.

- Remember to always use Binary Search to search for things in a sorted array/collection.