CSE 2123
Search

Jeremy Morris
Problem Specification: Search

- Given a list of values and an element $k$, find the index of $k$ in the list
- Output “not found” if $k$ is not in list
Problem Specification: Search

- For our lecture examples, we will make a few assumptions:
  - Underlying collection type: simple array
  - Items to be searched: integers

- Note that all of the search algorithms are extensible to other data types, collection types
  - Need only minor modifications
Comparing Search Algorithms

- How can we tell if one algorithm (A) is better than another one (B)?
  - Count how many steps it takes A to find an element
  - Count how many steps it takes B to find an element
  - Which one took fewer steps?
  - To be accurate, we need to consider worst case performance for both algorithms
Sequential (Linear) Search

Algorithm

1. Start at the first element in the list \((i=0)\)
2. If \(\text{list}[i]==k\), return \(i\)
3. Otherwise, \(i=i+1\)
4. If \(i > \text{list}.\text{length}\), return “not found”
5. Go back to step 2

A very simple algorithm
public static int linearSearch(int[] a, int k) {

    for (int i=0; i<a.length; i++) {
        if (a[i] == k) {
            return i;
        }
    }
    return -1;
}
Efficiency of Linear Search

- Take a = {4, 16, 20, 11, 7, 3, 21}

- Count the number of comparisons
  - Best case: k=4 … 1 comparison
  - Worst case: k=22 … n comparisons
  - Average case: … n/2 comparisons

- #comparisons ≤ n
Efficiency of Linear Search

What if our list were sorted?

- $a = \{3, 4, 7, 11, 16, 20, 21\}$

- Can we exploit the fact that we know it is sorted to perform a more efficient (i.e. faster) search?
Binary Search

Algorithm

1. Find the middle index of the list, call it i
2. If list[i] == k, return i
3. Otherwise, eliminate half the list
4. Set i to the middle index of the new list
5. Go back to step 2
Binary Search - Example

- Search for $k=16$

- Algorithm
  1. Find the middle index of the list, call it $i$
Binary Search - Example

- Search for $k=16$
- Find the middle index of the list ($i$)
  - $\text{list}[i] \neq 16$
Binary Search - Example

Search for k=16

Find the middle index of the list (i)
- list[i] != 16
- list[i] < 16
- Throw away the portion of the list ≤ i
Binary Search - Example

- Search for $k=16$
  - Throw away the portion of the list $\leq i$

```
3 4 7 11 16 20 21
```

$i=3$
Binary Search - Example

Search for $k=16$

- Throw away the portion of the list $\leq i$
- Find the middle value of the new list and make it $i$
- $list[i] \neq 16$
- $list[i] > 16$
Binary Search - Example

Search for k=16

- Throw away the portion of the list ≥ i
- Find the middle value of the new list and make it i
- list[i] == 16
- return i=4
Binary Search as a Recursive Problem

- Divide & Conquer
  - Base Case
    - If list[i] == k, return i
    - If the sublist being searched is of length less than 1, return “not found”
  - General Case
    - If list[i] < k
      - Return binarySearch on the sublist of elements greater than i
    - Otherwise
      - Return binarySearch on the sublist of elements less than i
public static int binarySearch(int[] a, int k, int start, int end) {
    if (start > end) {
        return -1; // Base Case: Not found
    }
    int mid = (start + end) / 2;

    if (a[mid] == k) {
        return mid; // Base Case: Found
    } else if (a[mid] < k) {
        return binarySearch(a, k, mid + 1, end);
    } else {
        return binarySearch(a, k, start, mid - 1);
    }
}
Binary Search as an Iterative Problem

- We can also view Binary Search as an iterative problem
  - Set our start and end indices
  - While start < end:
    - Find the mid point
    - Check to see if it is our value
    - If not:
      - If it is less than our value, set our start point so we’re not looking at values less than the mid point
      - If it is greater than, set our end point so we’re not looking at values greater than our mid point
    - Eventually we will find our value or start will become larger than end, meaning “not found”
public static int iterBinarySearch(int [] a, int k) {

    int start=0;
    int end=a.length-1;

    while (start<=end) {
        int mid = (start+end)/2;
        if (k==a[mid])
            return mid;
        else if (a[mid]<k) {
            start=mid+1;
        }
        else {
            end=mid-1;
        }
    }
    return -1;
}
Efficiency of Binary Search

- We can visualize a binary search using a *binary decision tree*
Efficiency of Binary Search

- \( k=11 \) … 1 comparison
- \( k=4 \) … 2 comparisons
- \( k=20 \) … 2 comparisons
- \( k=3 \) … 3 comparisons
- \( k=7 \) … 3 comparisons
- \( k=16 \) … 3 comparisons
- \( k=21 \) … 3 comparisons

- Best case – 1 comparison
- Worst case – 3 comparisons
  - \#comparisons \( \leq \log n \)
Efficiency of Binary Search

- Binary search is a big improvement over linear search...
  - ... so long as our list of elements is *sorted*