Search

Searching

- Searching is the process of finding a target element within a group of items (called a *search pool*).
- The target may or may not be in the search pool.
- We want to perform searching efficiently, minimizing the number of comparisons we make.

Searching

- Search requires a way to compare items.
- Typically, this will be the overridden equals method.

LINEAR SEARCH

Linear Search

- A linear search begins at one end of a list and examines each element in order.
- Either the item is found or we reach the end of the list.
- Linear search is $O(n)$.

Linear Search

- Linear search can be performed on sorted or unsorted data.
  - If performed on sorted data, we can be more efficient!
- Linear search can be implemented iteratively or recursively.
- Linear search can work for arrays or linked node implementations.

Linear Search- Iterative

```
boolean found=false;
for (int i=0; i<data.length; i++) {
    if (target.equals(data[i]) ) {
        found = true;
    }
}
return found;
```

Linear Search- Improved Iterative

```
boolean found=false;
for (int i=0; i<data.length && !found; i++) {
    if (target.equals(data[i]) ) {
        found = true;
    }
}
return found;
```

- Still O(n), but more efficient in the real world.
- We could also use a break or return inside the conditional.

Linear Search- Improved Iterative for Sorted Lists Only

```
boolean found=false;
boolean pastIt = false;
for (int i=0; i<data.length && !found && !pastIt; i++) {
    if (target.equals(data[i]) ) {
        found = true;
    } else if(target.compareTo(data[i]) < 0)) {
        // target is smaller than the data- so we should
        // have seen it by now- it’s not in the data
        pastIt = true;
    }
}
return found;
```

- Still O(n), but more efficient in the real world.
- What must be true of the type of objects in the array?

Linear Search- Recursive

```
boolean linearSearch(int first, int last,
                      Object[] data, Object target) {
   if(first > last) {
      return false; // indices cross over
   } else if(target.equals(data[first])) {
      return true;  // we found it!
   } else {
      return linearSearch(first+1, last, target, data);
      // keep looking
   }
}
```

Linear Search- Recursive

```
boolean linearSearch(int first, int last,
                      Object[] data, Object target) {
   if(first > last) {
      return false; // indices cross over
   } else if(target.equals(data[first])) {
      return true;  // we found it!
   } else {
      return linearSearch(first+1, last, target, data);
      // keep looking
   }
}
```

- Can you modify this to be more efficient for a sorted list?

Example

- Review the search code and examples.

Linear Search ERROR!

- This is a common mistake!

```
boolean found;
for (int i=0; i<length; i++)
    if (searchValue.equals(entry[i]) ) {
        found = true;
    } else {
        found = false;
    }
}
return found;
```

Linear Search- Improved Iterative for Nodes

```
boolean linearIterativeNodeSearch(Node head,
                                  Object target) {
   boolean found = false;
   Node current = head;
   while(current != null && !found) {
      if(target.equals(current.data)) {
         found = true;
      } else {
         current = current.next;
      }
   }
   return found;
}
```

Linear Search- Recursive for Nodes

```
boolean linearRecursiveNodeSearch(Node head,
                             Object target) {
   Node current = head;
   if(current == null) {
      return false;
   } else if(target.equals(current.data)) {
      return true;
   } else {
      return linearRecursiveNodeSearch(
                        current.next, target));
}
```

BINARY SEARCH

Binary Search

- A *binary search* assumes the list of items in the search pool is already sorted.
- Binary search eliminates a large part of the search pool with a single comparison.
  - Each comparison eliminates about half of the remaining data.
- Binary search is O(log n).

Binary Search

- Binary search can be implemented iteratively or recursively.
- Binary search does not make sense for linked node implementations!

Binary Search

- A binary search first examines the middle element of the list.
  - If it matches the target, the search is over.
  - If it doesn't match the target, we only need to search half of the remaining elements (since they are sorted).
- This process continues by comparing the middle element of the remaining viable candidates.
- Eventually, we find the target or exhaust the data.

Hi-Lo Guessing Game

- You think of a number between 1 and 100 and I try to guess it. You tell me if I am too high or low.
- If we play this *smartly*, what would the first guess be? If you make smart guesses, how many guesses will it take (in the worst case)?

Hi-Lo Guessing Game

- The smart first guess is the halfway point, so 50. Then, if 50 is too low, you should guess the new halfway point (75), and so on.

Hi-Lo Guessing Game

Range	Half-Way Value (The Guess)	Value is...	Guess Number
1-100	50	too low	1
51-100	75	too high	2
51-74	62	too low	3
63-74	68	too high	4
63-67	65	too low	5
66-67	66	too low	6
67-67	67	equal!	7

- If the number was 67, it took 7 (smart) guesses to find it.
- This is because the number of times that we guessed the halfway value was log(n) and log(100) = 7.
- We should always be able to guess the number in 7 or less guesses!

Binary Search- Iterative

```
boolean binarysearch(int[] numbers, int target) {
   boolean found = false;
   int first = 0;
   int last = numbers.length - 1;
   while (first <= last && !found) {
      int mid = (first + last) / 2;
      if (numbers[mid] == target) {
         found = true;
      } else if (numbers[mid] < target) {
         first = mid + 1;
      } else { // numbers[mid] > target
         last = mid - 1;
      }
   }
   return found;
}
```

Binary Search- Iterative

```
boolean binarysearch(int[] numbers, int target) {
   boolean found = false;
   int first = 0;
   int last = numbers.length - 1;
   while (first <= last && !found) {
      int mid = (first + last) / 2;
      if (numbers[mid] == target) {
         found = true;
      } else if (numbers[mid] < target) {
         first = mid + 1;
      } else { // numbers[mid] > target
         last = mid - 1;
      }
   }
   return found;
}
```

Binary Search- Recursive

```
boolean binarySearch(int first, int last, int[] data, int target
{
   int mid = ( (last - first) / 2 ) + first;
   if(first > last) {
      return false; // indices cross over
   } else if(target==data[mid]) {
      return true;  // we found it!
   } else if (target < data[mid]) {
      return binarySearch(first, mid-1, target, data);
      // keep looking in left half
   } else { // target > data[mid]
      return binarySearch(mid+1, last, target, data);
      // keep looking in right half
   }
}
```

Binary Search- Recursive

```
boolean binarySearch(int first, int last, int[] data, int target
{
   int mid = ( (last - first) / 2 ) + first;
   if(first > last) {
      return false; // indices cross over
   } else if(target==data[mid]) {
      return true;  // we found it!
   } else if (target < data[mid]) {
      return binarySearch(first, mid-1, target, data);
      // keep looking in left half
   } else { // target > data[mid]
      return binarySearch(mid+1, last, target, data);
      // keep looking in right half
   }
}
```

Binary Search

- Two ways to choose mid:
  - mid = ((last - first) / 2) + first;
  - mid = (first + last) / 2
- The second is simpler. It will work unless large numbers- overflow

- With the algorithms shown, what would be passed in as first and last?

Example

- Review the search code and examples.

Efficiencies of Searches

- Linear search is O(n)
- Binary search is O(log n)

- Binary is much more efficient than linear!
- But binary requires sorted data!
  - Even “fast” sorting algorithms are O(n log n).

- You have to know your data and the task required!

Choosing a Search

- Do you have linked nodes? yes!
  - Use linear (sequential) search!

Choosing a Search

- Do you have an array? yes!
- Is it sorted? no
  - Use linear (sequential) search!
  - Or consider sorting! This depends… How often do you plan to search? (If only a few times, probably not worth the sort.) How often will you have to sort? What is the size of your data?

Choosing a Search

- Do you have an array? yes!
- Is it sorted? yes
  - (This assumes items implement Comparable!)
  - Use binary search (but linear is okay too)

Search in the Java Standard Library

- List interface (implemented by ArrayList and LinkedList) has the indexOf method and the lastIndexOf method. These use linear search.

- Arrays class has a collection of static binary search methods.
  - Throws a runtime exception if the objects are not Comparable!
  - Invoke as: Arrays.binarySearch(data, target);