Overview

- Topics
  - Maintaining an ordered list
  - Sequential and binary search
  - Recursion
  - Sorting: insertion sort and QuickSort
- Reading
  - Textbook: ch. 13 & sec. 17.1-17.3

Problem: A Word Dictionary

- Suppose we want to maintain a real dictionary. Data is a list of <word, definition> pairs -- a "Map" structure
  - "aardvark", "an animal that starts with an A and ends with a K"
  - "apple", "a leading product of Washington state"
  - "banana", "a fruit imported from somewhere else"

- We want to be able to do the following operations efficiently
  - Look up a definition given a word (key)
  - Retrieve sequences of definitions in alphabetical order

Representation

- Need to pick a data structure
- Analyze possibilities based on cost of operations
  - unordered list
  - hash map
  - ?

Ordered List

- One solution: keep list in alphabetical order
- To simplify the explanations for the present: we’ll treat the list as an array of strings, and assume it has sufficient capacity to add additional words/defs when needed

```
0  aardvark    // instance variable of the Ordered List class
1  apple       String[] words; // list is stored in words[0..size-1]
2  banana      int size;      // # of words
3  cherry
4  kumquat
5  orange
6  pear
7  rutabaga
```

Sequential (Linear) Search

- Assuming the list is initialized in alphabetical order, we can use a linear search to locate a word

```
// return location of word in words, or -1 if found
int find(String word) {
    int k = 0;
    while (k < size && !word.equals(words[k])) {
        k++
    }
    if (k < size) { return k; } else { return –1; }
}
```

- Time for list of size n:
Can we do better?

- Yes! If array is sorted
- Binary search:
  - Examine middle element
  - Search either left or right half depending on whether desired word precedes or follows middle word alphabetically

The list being sorted is a precondition of binary search.
- The algorithm is not guaranteed to give the correct answer if the precondition is violated.

---

Binary Search

```java
int find(String word) {
    return bSearch(0, size-1);
}
```

```java
int bSearch(String word, int lo, int hi) {
    // return -1 if interval lo..hi is empty
    if (lo > hi)   { return  –1; }
    // search words[lo..hi]
    int mid = (lo + hi) / 2;
    int comp = word.compareTo(words[mid]);
    if (comp == 0)  { return mid; }
    else if (comp < 0)  { return _________________________ ; }
    else /* comp > 0 */ { return _________________________ ; }
}
```

---

"The Word Must Be Where?" Three Cases

```java
int comp = word.compareTo(words[mid]);
if (comp == 0) {
    //the word must be where? _______________
    return _____________________ ;
}
else if (comp < 0)  {
    //the word must be where? _____________
    return _________________________ ;
}
else  { //comp > 0
    //the word must be where? _____________
    return _________________________ ;
}
```

---

"Where?" Answered

```java
int comp = word.compareTo(words[mid]);
if (comp == 0) {
    //the word must be where? at position "mid"
    return _______________ ;
}
else if (comp < 0)  {
    //the word must be where? in the lower half of the array
    return _______________ ;
}
else  { //comp > 0
    //the word must be where? in the upper half of the array
    return _______________ ;
}
```

---

Return Values: Three Cases

```java
int comp = word.compareTo(words[mid]);
if (comp == 0) {
    //the word must be where? at position "mid"
    return mid;  
}
else if (comp < 0)  {
    //the word must be where? in the lower half of the array
    return /*the result of searching the lower half of the array*/ _________________________ ;
}
else  { //comp > 0
    //the word must be where? in the upper half of the array
    return /*the result of searching the upper half of the array*/ _________________________ ;
}
```

---

What is "The Lower Half"?

```java
...  
else if (comp < 0) {  
    //the word must be where? in the lower half of the array
    return /*the result of searching the lower half of the array*/ _________________________ ;
}
```
**Comments Complete, Code Incomplete**

```java
int comp = word.compareTo(words[mid]);
if (comp == 0) {
    //the word must be where?
    return mid;
} else if (comp < 0) {
    //the word must be where?
    in the lower half of the array
    return /*the result of searching from lo to mid-1*/;
} else { //comp > 0
    //the word must be where?
    in the upper half of the array
    return /*the result of searching from mid+1 to hi*/;
}
```

---

**Last Piece of the Puzzle**

```java
return /*the result of searching from lo to mid-1*/;
}
```

How can we get the "result of searching from lo to mid-1"?

We have a method called bSearch that can search an array within a range of indexes.

```java
int bSearch(String word, int x, int y)
```

Let x be lo, let y be mid-1
bSearch(String word, int lo, int mid-1)

---

**Recursion**

- A method (function) that calls itself is **recursive**
- Nothing really new here
- Method call review:
  - Evaluate argument expressions
  - Allocate space for parameters and local variables of function being called
  - Initialize parameters with argument values
  - Then execute the function body
- What if the function being called is the same one that is doing the calling?
  - Answer: no difference at all!

---

**Wrong Way to Think About It**

```
bSearch(array, lo, mid-1)
```

---

**Right Way to Think About It**

```
bSearch
    bSearch(array, lo, mid-1)
```

---

**Trace**

- Trace execution of find("orange")
  - 0 aardvark
  - 1 apple
  - 2 banana
  - 3 cherry
  - 4 kumquat
  - 5 orange
  - 6 pear
  - 7 rutabaga
Trace

- Trace execution of `find("kiwi")`

  0 aardvark
  1 apple
  2 banana
  3 cherry
  4 kumquat
  5 orange
  6 pear
  7 rutabaga

Performance of Binary Search

- Analysis
  - Time (number of steps) per each recursive call:
  - Number of recursive calls:
  - Total time:
  - A picture helps

Binary Search Sizes

All paths from the size N case to a size 0 case are the same length: $1 + \log_2 N$

Linear Search vs. Binary Search

- Compare to linear search
- Time to search 10, 100, 1000, 1,000,000 words
- Linear
- Binary
- What is incremental cost if size of list is doubled?
- Why is Binary search faster?
  - The data structure is the same
  - The precondition on the data structure is different: stronger
  - Recursion itself is not an explanation
  - One could code linear search using recursion

More About Recursion

A recursive function needs three things to work properly

1. One or more base cases that are not recursive
   - if (lo > hi) { return -1; }
   - if (comp == 0) { return mid; }
2. One or more recursive cases that handle a else if
   - (comp < 0) { return bsearch(word,lo,mid-1); }
   - else /* comp > 0 */ { return bsearch(word,mid+1,hi); }
3. The recursive cases must lead to “smaller” instances of the problem
   - “Smaller” means: closer to a base case
   - Without “smaller”, what might happen?

Recursion vs. Iteration

- Recursion can completely replace iteration
- Some rewriting of the algorithm is necessary
- Usually minor
- Some languages have recursion only
- Recursion is often more elegant but less efficient
- Recursion is a natural for certain algorithms and data structures (where branching is required)
- Useful in “divide and conquer” situations
- Iteration can completely replace recursion
- Some rewriting of the algorithm is necessary
- Often major
- A few (mostly older languages) have iteration only
- Iteration is not always elegant but is usually efficient
- Iteration is natural for linear (non-branched) algorithms and data structures
Recursion and Elegance

- Problem: reverse a linked list
- Constraints: no last pointer, no numElems count, no backward pointers, no additional data structures
- Non-recursive solution:
  - try it!

Recursive Solution: Simple, Elegant

- Problem: reverse a linked list
- Constraints: no last pointer, no numElems count, no backward pointers, no additional data structures

\[
\text{newList} = \text{reverse(oldList.first)};
\]

...  
\[
\text{List reverse(\text{Link firstLink})} \{
\text{if (firstLink == null) \{ return new SimpleList(); \}} \n\text{return reverse(firstLink.next).add(firstLink.data)}; \}
\]

- Better hope this is a question on your Microsoft job interview!
- PS: Did we cheat??