CSE 142

Searching

Outline for Today

- Review – sequential (linear) search of a list
- Binary search
- Comparing algorithms

Searching

- Many information processing applications involve searching for something
- Sometimes the search must be *wide*
  - Example: finding the best airfare for a trip to New York
  - What exactly gets searched is vague
  - Where do you even start?
  - When you get an answer, is it even right?

- A more *narrow* but very common use of searching: looking for something specific in a single collection
  - We know where to look
  - We know what the answer looks like

Searching a List

- For this lecture, assume that we've got a list, and some collection of strings has been added to the list

```
ArrayList names = new ArrayList( );
names.add("frog");
names.add("rabbit");
names.add("aardvark");
```

- Problem: Look for a particular name in the list
- Clearly, this is a narrowly defined search

Searching a List

- Problem: Look for a particular name in a list of strings
- Still have to define some details
- What form should the result be?
- One common choice: report a boolean to say if the item is found
- Another very common choice:
  - If found, report the *position* of the item sought
  - If not found, report something to indicate "not found"
- As almost always in programming specifications, "report" means "return a value", *not* "print something".

Linear Search

- Locate a string in the list

```
/* Assume the list to be searched is an instance variable called names */
/** Return position of str in the list, or –1 if not present */
public int find(String str) {

/* Walk the list and compare each item to the parameter */




}
```


Can we do better?

- How much work does linear search do?
- Can we do it faster?
  - No, if we don't know anything about the order of elements in the list
  - Yes, if the list is in order

Binary Search – Informal

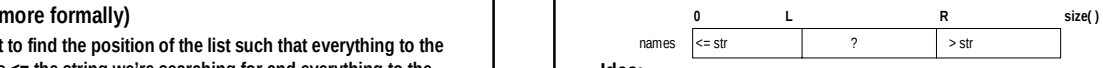
- Idea
  - Look in the *middle* of the list
  - If we haven't found what we're looking for, we can ignore half of the list and look at the other half
- The list *must* be in order (sorted) for this to work
- Such a requirement is an example of a *precondition*
  - A precondition allows the programmer to make an assumption
  - We'll assume names.get(0) <= names.get(1) <= ... <= names.get(names.size( )-1)

Binary Search – Goal

- Recall: the return value is a *position*
- Goal (more formally)
  - Want to find the position of the list such that everything to the left is <= the string we're searching for and everything to the right is >
  - Such a position always exists, even if the value sought is not in the list!
- Picture:

Binary Search – Strategy

- On a typical iteration, we have



- Idea:
  - Let mid = (L+R)/2
  - If names.get(mid) <= str, move the L index (in which direction?)
  - If names.get(mid) > str, move the R index

(Note: In the book, Nino & Hosch use a slightly different invariant. For them, names.get(low) to names.get(high) is the *unexamined* region. In these slides, the unexamined region is names.get(L+1) to names.get(R-1). Either can be made to work correctly.)

Detail: String Comparisons

- We need to compare Strings to determine ordering, not just equality
- Can't use <, <=, etc. on objects
- Solution: method *compareTo* in class *String*

  s.compareTo(t)

  returns

  negative integer if s comes before t
  zero if s and t have identical values
  positive integer if s comes after t

For Strings, "before" and "after" are determined by the values of the Unicode character set

As a first approximation, think "case-sensitive alphabetical order"

compareTo Footnote

- s.*compareTo(t)* in class *String* returns

  negative integer if s comes before t
  zero if s and t have identical values
  positive integer if s comes after t

- Many Java classes define a compareTo method with this kind of return value scheme
- An example of a common programming convention
  - Not enforceable, except as a social contract

Binary Search – Code

```
/** Return location of str in the list, or –1 if not present */
public int find(String str) {

    while ( ____________________________ ) {



    }

}
```


Binary Search – Test

- Invent some data, try the algorithm

Binary Search – Test

Binary Search – Performance

- Is the extra complexity worth it?
- How much work is done to search a list of a given size?
- or, How big a list can be searched with *n* comparisons?

Binary & Linear Search Compared

- Linear search: work ~ size
- Binary search: work ~ $\log_2$ size
  - This is a fundamental difference – not just a constant speedup
  - But it requires a sorted list
- Graph: