CSE 142

Sorting

Outline for Today

- Review
  - Sequential vs Binary Searcy
  - Arrays
- Maintaining an Ordered List
  - Sorting

Linear vs Binary Search

- Recall work needed to search a list of *n* items
  - Linear search ~ *n*
  - Binary search ~ log *n*
- For all but small lists, binary search is much, much, much faster
  - For *n* = 1,000, log *n* ~ 10
  - For *n* = 1,000,000, log *n* ~ 20
- But we can only do binary search if the list is *sorted*
  - So how do we sort a list?

Design Your Sorting Algorithm Here

A Sorted StringList

- Choices
  - Keep list sorted at all times
    Need to make adjustments in add method
  - Sort list before searching if not done already
    Need check in contains (search) method to sort if not currently sorted
- In either case, order of items in list is no longer order in which added
  - But that's presumably ok – if we want really fast searches, this is a tradeoff worth making
  - Terminology: this is a ***multiset*** or ***bag*** of strings
    /** Unordered collection of Strings, possibly with duplicate elements */
    public class StringBag { ... }

Maintaining a Sorted List

- Nothing in the client interface changes
  - Except we can no longer allow client to insert arbitrary strings in the middle of the list
- Implementation now relies on list being sorted, so it's crucial that we record this information in a comment

```
// instance variables
private String[ ] strings;      // Strings in this StringList are stored in
private int       numStrings;   //   strings[0] through strings[numStrings-1],
                                //   and the strings are stored in ascending
                                //   order:  strings[0] <= strings[1] <= ...
                                //   <= strings[numStrings-1]
```

Method add

- **Only method from original StringList that needs to be changed (true?)**

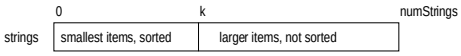
```
/** Add str to this StringBag.  Return true if successful, otherwise return false */
public boolean add(String str) {
    if (this.numStrings == this.strings.length) {
        return false;
    }
    // find correct location to place str
    ...
    // shift larger elements one position to the right
    ...
    // place str in correct location
    ...
    numStrings++;
    return true;
}
```


Modified method add

- **Picture:**
- **Implementation details: exercise**

Selection Sort

- **Sort elements in unordered list**
- **Idea: At each step, pick smallest element in not-yet-sorted part of array and move it to the front**
- **Picture**



- **Detailed step (repeat until sorted)**
  - **Find smallest item in strings[k]..strings[numStrings-1]**
  - **Swap that item with item in strings[k]**
  - **Increase k and repeat**

Code For Selection Sort

Code for Finding Minimum Element

Test

- **Invent some data, check the code**

Embedding in a String Collection Class

- Our original StringList class can be changed to sort the list as needed to allow binary search for contains
  - Add an instance variable to record whether the list is sorted
  - In method add, set this variable to false
  - In method contain, call the sort method if this variable is false, then do a binary search after the sort finishes
  - In method sort, set the variable to true after sorting

Conclusion

- Performance Tradeoffs
  - Sorting is relatively expensive
  - Pays off if searches are frequent and clustered together compared to additions to the list
- Can either maintain list in sorted order at all times (expensive add operation) or sort when needed (potentially expensive lookup)
- For both algorithms, the diagrams give the key ideas
  - The code is relatively straightforward from there