Sorting and Searching

15-110 Summer 2010

Margaret Reid-Miller

Selection Sort Algorithm

- Given an array of items, arrange the items so that they are sorted from smallest to largest.
- **Select** next item, in turn, that will be appended to the sorted part of the array:
  - Scan the array to find the smallest value, then swap this value with the value at cell 0.
  - Scan the remaining values (all but the first value), to find the next smallest, then swap this value with the value at cell 1.
  - Scan the remaining values (all but the first two) to find the next smallest, then swap this value with the value at cell 2.
  - Continue until the array is sorted.

Example: Selection Sort

[0]	[1]	[2]	[3]	[4]	
5	1	3	7	2	find min
1	5	3	7	2	swap to index 0
1	5	3	7	2	find min
1	2	3	7	5	swap to index 1
1	2	3	7	5	find min
1	2	3	7	5	swap to index 2
1	2	3	7	5	find min
1	2	3	5	7	swap to index 3

Selection Sort Implementation

on an array of `int`

```
public static void selectionSort(int[] data) {

  for (int numSort = 0 ; numSort <  data.length-1 ; numSort++){

     // find the next minimum
     int minPos =  numSort  : // initial position of next min
     for (int pos =  numSort+1  ; pos < data.length; pos++) {
        if (data[minPos] > data[pos])
            minPos = pos; // found new min
     }

     // swap min to next position in sorted list
     int temp = data[minPos];
     data[minPos] = data[ numSort ];
     data[ numSort ] = temp;
  }
}
```

The `compareTo` method

- To determine the relative ordering of two strings, the `String` class has a `compareTo` method:

  `public int compareTo(String other)`

- ***By convention***, the `compareTo` method returns a negative integer, zero, or positive integer if this object (through which the method was invoked) is "less than", "equal to", or "greater than" the object specified in the parameter, respectively.
- For example:

`(str1.compareTo(str2) < 0)` means *str1* < *str2*

Lexicographical ordering

- The `String` class `compareTo` method compares two strings *lexicographically*, which is similar to alphabetically except that it includes digits and other symbols:
  - Space comes before digits
  - Digits come before uppercase letters
  - Uppercase letters come before lower case letters
- Example: The following are in lexicographical order:

  `"01234" "012AB" "ABC" "ABC D" "ABCD"`

  `"XYZ" "XyZ" "abc" "bc"`

Using `compareTo` with Strings

```
public void insertInArtistOrder(Song newSong) {

  if (numSongs == songList.length){
   doubleLength();
  }

  // Search for the location to insert in order
  while (index < numSongs && newSong.getArtist().
    compareTo(songList[index].getArtist()) > 0) {
     index++;
  }
  ...
```


Example: `Date` class

- Suppose a `Date` class is defined as follows:

```
public class Date {
   int year;
   int month;
   int day;
   …
 }
```

- How would you write a `compareTo` method for the Date class?

compareTo for Date class

```
public class Date {

  public int compareTo(Date other) {

    if (this.year != other.year)
      return this.year - other.year;
    else if (this.month != other.month)
      return this.month - other.month;
    else
      return this.day - other.day;
  }
```


equals for Date class

```
  public boolean equals(Date other) {

    return this.compareTo(other) == 0);
  }

  // other methods not shown

} // end Date class
```


Selection Sort Implementation

on an array of String objects

```
public static void selectionSort(String[] data) {

  for (int numSort = 0; numSort < data.length-1; numSort++){

     // find the next minimum
     int minPos = numSort; // initial position of next min
     for (int pos = numSort+1; pos < data.length; pos++) {
        if (data[minPos].compareTo(data[pos]) > 0)
            minPos = pos; // found new min
     }

     // swap in min to next position in sorted list
     String temp = data[minPos];
     data[minPos] = data[numSort];
     data[numSort] = temporary;
  }
}
```


Selection Sort Implementation

on an array of Date objects

```
public static void selectionSort(Date[] data) {

  for (int numSort = 0; numSort < data.length-1; numSort++){

     // find the next minimum
     int minPos = numSort; // initial position of next min
     for (int pos = numSort+1; pos < data.length; pos++) {
        if (data[minPos].compareTo(data[pos]) > 0)
            minPos = pos; // found new min
     }

     // swap in min to next position in sorted list
     Date temp = data[minPos];
     data[minPos] = data[numSort];
     data[numSort] = temp;
  }
}
```

Insertion Sort Algorithm

- Given an array of items, arrange the items so that they are sorted from smallest to largest.
- For each item, in turn, **insert** the item into the sorted part of the array:
  - The first item is sorted in a list of one item
  - Insert second item in sorted list of one item.
  - Insert third item in sorted list of two items.
  - Insert fourth item in sorted list of three items.
  - Continue until all the items are sorted.

See http://math.hws.edu/TMCM/java/xSortLab/ for animations

Example: Insertion Sort

[0]	[1]	[2]	[3]	[4]	
5	1	3	7	2	insert [1]
1	5	3	7	2	shift
1	5	3	7	2	insert [2]
1	3	5	7	2	shift
1	3	5	7	2	insert [3]
1	3	5	7	2	shift
1	3	5	7	2	insert [4]
1	2	3	5	7	shift

Insertion Sort Implementation
on an array of `int`

```
public static void insertionSort(int[] data) {
   for (int numSort = 1 ; numSort < data.length ; numSort++){
      // save next item to insert into sorted list
      int temp = data[ numSort ];

      // move the hole to insertion position
      int hole = numSort ;
      while ( hole > 0 && temp < data[hole-1]) {
         data[hole] = data[hole-1];
         hole--;
      }
      data[hole] = temp; // insert at hole
   }
}
```


Binary Search Algorithm

- Given an array of items sorted in increasing order and an item, find the position of the item in the array:
  - Guess that it is the middle item.
  - If it is, then return the middle index.
  - Otherwise, determine if it is in the upper or lower half.
  - Repeat on this half to find in which quarter it is.
  - Repeat until either find the item or there are no values left.

Example: Binary Search

search for 64

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	
11	18	19	22	24	35	37	64	68	Find middle
11	18	19	22	24	35	37	64	68	Find half
11	18	19	22	24	35	37	64	68	Find middle
11	18	19	22	24	35	37	64	68	Find half
11	18	19	22	24	35	37	64	68	Find middle
									Found

Example: Binary Search

search for 20

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	
11	18	19	22	24	35	37	64	68	Find middle
11	18	19	22	24	35	37	64	68	Find half
11	18	19	22	24	35	37	64	68	Find middle
11	18	19	22	24	35	37	64	68	Find half
11	18	19	22	24	35	37	64	68	Find middle
11	18	19	22	24	35	37	64	68	Find half
11	18	19	22	24	35	37	64	68	Find middle
									Not found

Binary Search Implementation

```
public static int binarySearch(String[] array,
                               String key) {
    int begin = 0, end = array.length-1, mid = 0;
    boolean found = false;
    // key is between begin and end
    while (!found && begin <= end) {
        mid = (begin + end) / 2;    // integer division
        if (key.compareTo(array[mid]) == 0)
            found = true;
        else if (key.compareTo(array[mid]) < 0)
            end = mid-1;
        else  begin = mid+1
    }
    if (found) return mid;
    else return -1;
}
```