CS1007 lecture #20 notes

thu 11 apr 2002

- news
- sorting
- reading: *ch 6.3*

news.

- the midterm will be back on Tue Apr 16
- homework #5 is due Tue Apr 16
- the FINAL EXAM SCHEDULE:
  - **AM class: Tue May 14, 9.10am-12noon, 209 HAV**
  - **PM class: Thu May 16, 1.10pm-4pm, 301 Pupin**

sorting (1).

- sorting is one of the classic tasks done in computer programming
- the basic idea with sorting is to rearrange the elements in an array so that they are in a specific order — usually ascending or descending, in numeric or alphabetic order
- we will discuss 4 sorting algorithms (i.e., methods for sorting):
  - blort sort
  - insertion sort
  - selection sort
  - bubble sort

sorting (2).

- some sorts require an extra “auxiliary” array during sorting
  - the elements are moved from the original array into the auxiliary array, one at a time
  - at the end of the sort, the auxiliary array contains all the elements in sorted order
  - the final step is to copy the elements from the auxiliary array back into the original array
  - insertion and selection sorts are this type
- some sorts do not use an auxiliary array during sorting, but just move the elements around within the original array
  - these sorts involve the use of a swap() function, to switch the locations of two entries in the array
  - blort and bubble sorts are this type

swap.

- most sorts use a utility method called `swap()` to swap two elements in an array or Vector
- the methodology works like this
  - given two variables A and B, you want to switch the values so that the value of A gets the value of B and vice versa
  - you can't just simply copy one to the other and then vice versa because you'll lose the first value you copy to, so you need a temporary variable
  - here's the steps:
    1. $\boxed{\text{temp}} \leftarrow \boxed{\text{A}}$
    2. $\boxed{\text{A}} \leftarrow \boxed{\text{B}}$
    3. $\boxed{\text{B}} \leftarrow \boxed{\text{temp}}$
- example code: in `Vex.java`, the `swap()` method

blort sort.

- blort sort is the “fun but stupid” sort:
  1. check to see if the Vector is in sorted order
  2. if it is, then blort sort is done
  3. if it isn’t, then randomly permute the elements being sorted and loop back to the first step
- example code: three methods, in `Vex.java`:
  - `isSorted()`
  - `permute()`
  - `blortSort()`

insertion sort.

- insertion sort uses an *auxiliary* Vector to store the sorted elements temporarily
    1. it takes elements one at a time from the front the Vector being sorted and *inserts* them in sorted order into the auxiliary Vector
    2. it copies the content of the auxiliary Vector back into the original Vector, in sorted order
- two versions of the method are shown in `Vex.java`
    - the first version uses an auxiliary Vector explicitly
        - `insertionSort1()`
    - the second version does so implicitly, by keeping the sorted items at one end of the original Vector and the unsorted items at the other
        - `insertionSort2()`

selection sort.

- selection sort also uses an *auxiliary* Vector to store the sorted elements temporarily
  1. it *selects* elements one at a time in sorted order from the Vector being sorted and appends them to the end of the auxiliary Vector
  2. it copies the content of the auxiliary Vector back into the original Vector, in sorted order
- two versions of the method are shown in `Vex.java`
  - the first version uses an auxiliary Vector explicitly
    - `selectionSort1()`
  - the second version does so implicitly, by keeping the sorted items at one end of the original Vector and the unsorted items at the other
    - `selectionSort2()`
  - both use a utility method for finding the smallest element
    - `findMin()`

bubble sort.

- bubble sort repeatedly performs pairwise comparisons with neighboring elements in the Vector
- bubble sort always performs the number of passes equal to the size of the Vector minus 1
- one version of the method is shown in `Vex.java`
  - `bubbleSort()`