Laboratory 9B

Searching for the Right Sort

Objective

- Investigate sorting and searching algorithms and watch them work.

References

Software needed:

1) A web browser (Internet Explorer or Netscape)
2) Applets from the CD-ROM:
   a) Palgo
   b) Searching
   c) Sorting

Textbook reference: Chapter 9, pp. 297–308

Background

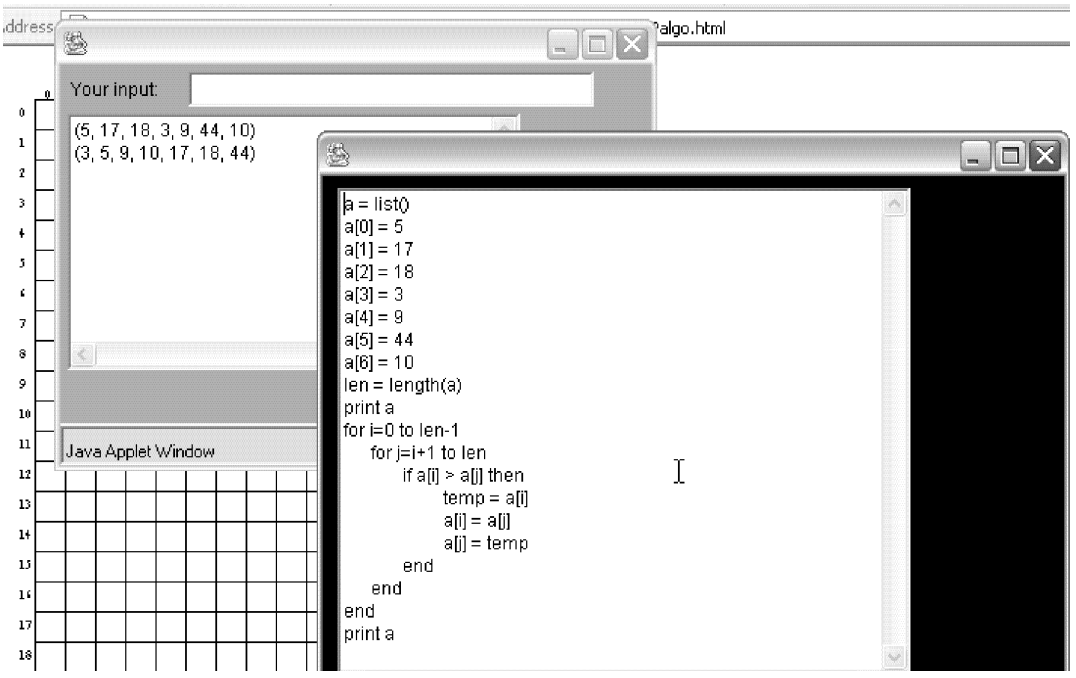
Chapter 9, "Abstract Data Types and Algorithms," gets deeper into programming with abstract data types and data structures. Two programming tasks the chapter focuses on are searching and sorting, illustrating various algorithms for both.

Activity

Part 1

Searching and sorting are venerable, respected topics in computer science. In fact, Volume 3 of Donald Knuth's famous series of books, *The Art of Computer Programming*, focuses exclusively on searching and sorting.

You'll be using several applets to explore searching and sorting. The "Palgo" applet provides a complete programming language with arrays (also called lists). Start the applet, select *Example 8—Sorting*, and run the program. Here's a screenshot:



Study the code shown in the screenshot. Can you figure out which of the three sorting algorithms discussed in the textbook was used for this sorting program? Your choices are selection sort (pp. 298–299), bubble sort (pp. 300–201), or quicksort (pp. 301–305).

Notice that the array holding the integers in this program is identified by the variable `a`. This variable is set to an empty array by the first line:

```
a = list()
```

Then the elements are filled in, one by one:

```
a[0] = 5
```

The length of the array can be discovered by the length function:

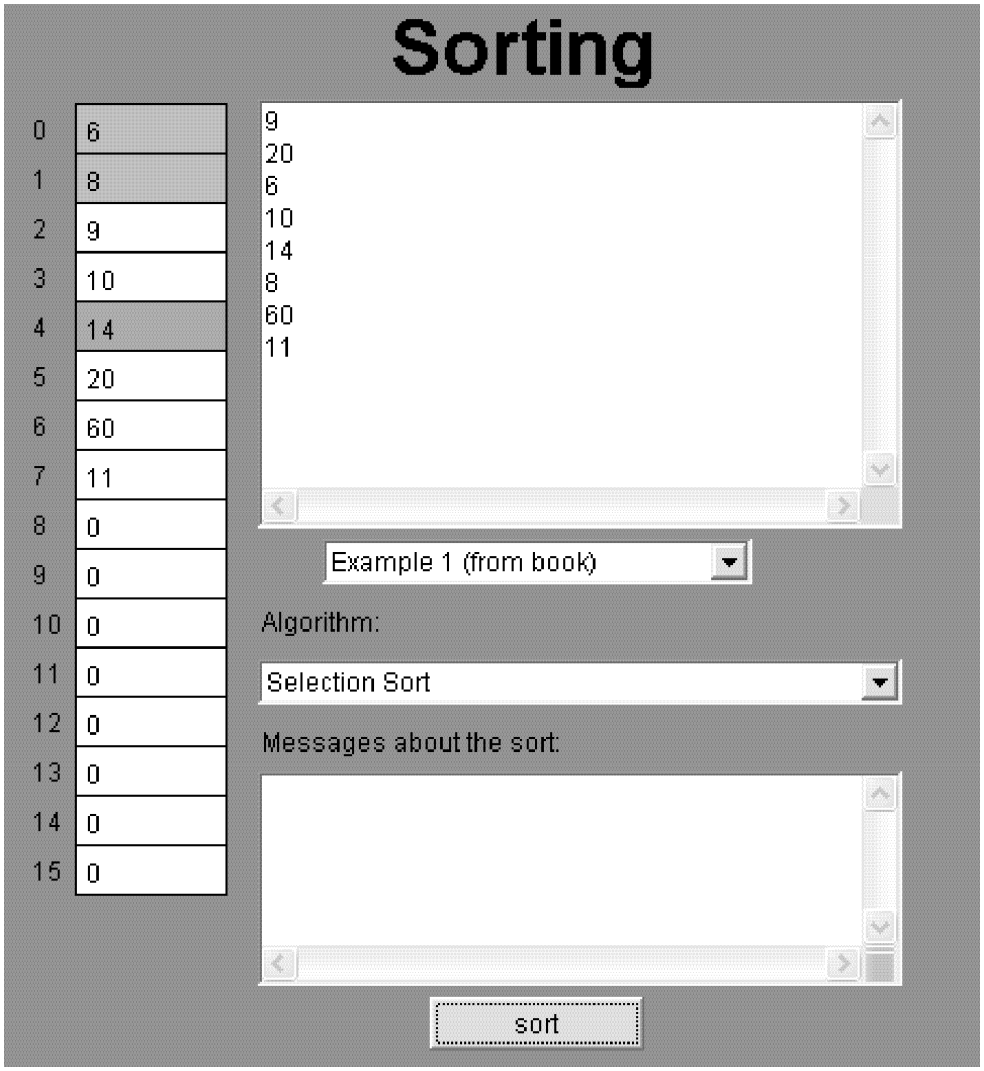
```
len = length(a)
```

Unlike some older, better-known languages like C, the programming language in Palgo is fairly easy and forgiving. In C, you have to declare the size of an array when you create it. There are some complex reasons why C requires this but Palgo doesn't—the main reason is that Palgo is *interpreted*, while C is *compiled*. The difference between these two ways of translating high-level code is discussed on pp. 236–238 of the textbook.

Experiment with the Palgo sorting program. As an extra challenge, you might try to encode one of the other sorting or searching algorithms into Palgo language and get it to run (much easier said than done, due to the nature of programming)! If you run Palgo as a standalone application, you can save your programs in a file on disk.

Part 2

Now we will examine the "Sorting" applet. Start your browser and open the "Sorting" applet. The large text area is where you type in a list of numbers. You can load several example datasets by selecting one from the pull-down menu near the bottom of the screen. Then choose a sorting algorithm from the *Sort Choice* pull-down menu in the middle of the window, and finally sort by pressing the *sort* button.



Each of the three algorithms colors the cells slightly differently as they work. The selection sort algorithm colors each cell pink as it examines it. If it decides to swap two cells, it briefly colors those cells red. The gray cells at the top represent the already sorted sublist.

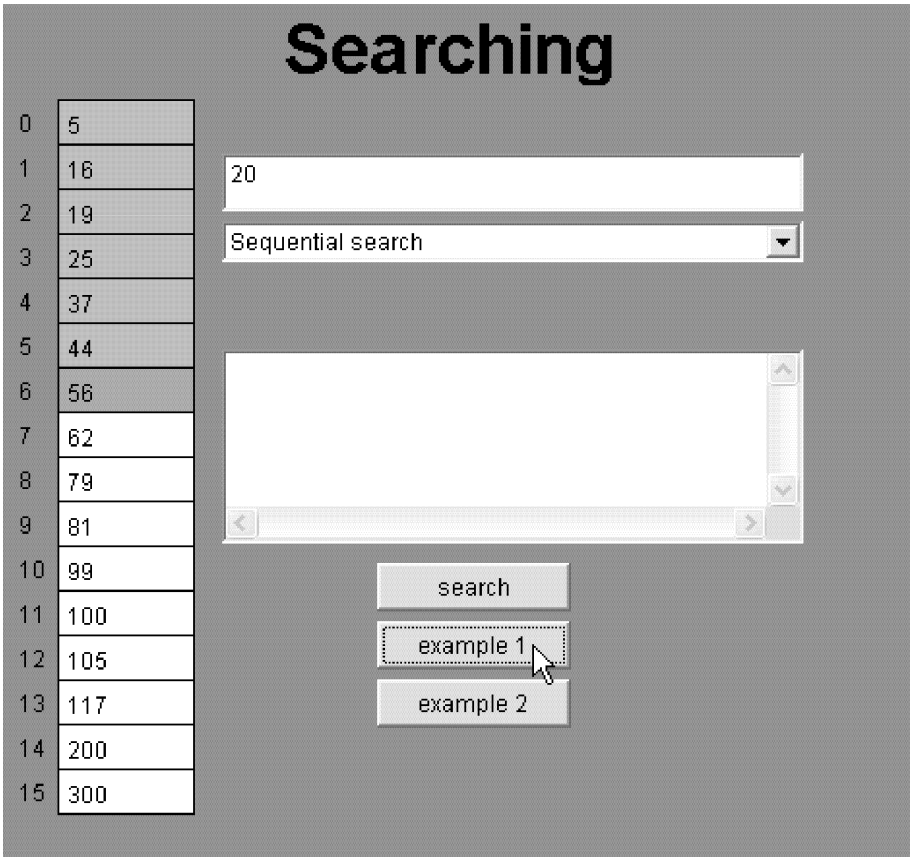
Look at several datasets and use the three different sorting algorithms on them, watching them as they work. The text area above the *sort* button gives the total number of comparisons that the program used while sorting. This can begin to give you a feel for which algorithm is best, and under which conditions it is best.

Of course, there are more than three different sorting algorithms! There is shell sorting, insertion sorting, radix sorting, heap sorting, merge sorting, and others. Even with all these choices, quicksort is perhaps used most often in the "real world." It is found in databases, in the Unix system's sort utility, and in many other places.

Part 3

Closely related to sorting is searching, in which we are trying to find whether a piece of information (such as a number, a character string, or a picture) resides in a collection. Collections are often, but not always, implemented by arrays. Sometimes collections have so many millions of items in them that trees or even more exotic data structures are needed. Trees are discussed on pp. 310–320 of your textbook, and a searching algorithm that is appropriate only for trees is shown in the green box at the bottom of p. 315.

To begin, start the "Search" applet and pull down to select *Example 1*. The applet uses the sequential search algorithm to search for the number 20 in the list. Since 20 is not to be found there, the applet will have to go through the maximum amount of work to find out this fact and report it to us.



The number of comparisons that it made is reported in the text area above the *search* button.

Type in a number that is found in the list on the left-hand side (such as 56) and click on *search*. This applet intentionally runs slowly so you can watch it at work. In real applications, of course, you would want the fastest program you could get, and animating the search would not be necessary.

Experiment with binary versus sequential search. To use binary search, click on the down-arrow in the menu box in the search applet where it currently says "Sequential Search." What generalizations can you make about their relative speeds? What additional requirements does binary search impose on the dataset?

Exercise 1

Name _____ Date _____

Section _____

1) Start the "Sorting" applet.

2) Make a table in which you can record the performance of the three algorithms on the four different data sets:

	Ex 1	Ex 2	Ex 3	Ex 4
selection				
bubble				
quicksort				

3) Run the applet 12 times and record the numbers it generates in the big white text area.

4) Which algorithm is usually the fastest?

5) Is any one algorithm always faster than the others?

6) Are any two algorithms similar?

7) Did the four datasets cover all the major "cases" you can think of, or are there others? If so, describe them.

Exercise 2

Name _____ Date _____

Section _____

1) Let's experiment with datasets other than those in the examples. Start the "Sorting" applet.

2) In the text area where it says "Your data set goes here," remove the text and type in these numbers. Press return after every number except the last.

```
1
99
2
98
3
97
4
96
5
95
6
94
7
93
8
```

3) Are these numbers sorted? Are they arranged randomly? What will the final sorted list look like?

4) Now let's see how our three sorting algorithms perform. Run each of the three algorithms and write down how many comparisons they required to finish the job. (Note: you don't have to retype the numbers in after each sort because the applet refreshes the numbered list on the left with the values you typed into the big text area but does not change the big text area.)

   Selection sort _____

   Bubble sort _____

   Quicksort _____

5) Which is the fastest? Does that surprise you?

6) Rerun the quicksort algorithm and write down any patterns you see occurring in the numbered list to the left. For instance, maybe the algorithm quickly moves all of the larger numbers to the top or bottom section.

Exercise 3

Name _____ Date _____

Section _____

1) Now for yet another dataset. Start the "Sorting" applet.

2) In the text area where it says "Your data set goes here," remove the text and type in these numbers. Press return after every number except the last.

```
1
3
8
15
26
45
50
97
92
84
79
72
70
68
51
```

3) Are these numbers sorted? Are they arranged randomly? What will the final sorted list look like?

4) Now let's see how our three sorting algorithms perform. Run each of the three algorithms and write down how many comparisons they required to finish the job.

   Selection sort _____

   Bubble sort _____

   Quicksort _____

5) Which is the fastest?

6) Rerun the quicksort algorithm and describe what happens to the number 97.

7) How is quicksort's behavior on this dataset different than its behavior on the previous dataset?

Exercise 4

Name _____ Date _____

Section _____

1) Start the "Searching" applet. The dataset in the numbered list on the left is fixed—you can't change it. However you can select an algorithm and a value to search for.
2) Run all four searches that are in the examples and write down how many comparisons were made:
   _____ Example 1: Something in the list (sequential)
   _____ Example 2: Something not in the list (sequential)
   _____ Example 3: Something in the list (binary)
   _____ Example 4: Something not in the list (binary)
3) Select "binary search" as your algorithm and try to find 62. How many tries does it need to find it?

4) Try to find 62 using sequential search. How many tries does it need?

5) Which searching algorithm is clearly superior on this dataset?

6) Is the numbered list on the left sorted? Is this a prerequisite for binary search? Is it a prerequisite for sequential search? (Hint: see p. 306 of your textbook.)

Exercise 5

Name _____ Date _____

Section _____

1) Start the "Searching" applet. Pull down the algorithm menu so that "Binary search" is chosen.

2) This may be a little tedious but there's a reason, so bear with it. For each value in the numbered list to the left, try to find it. Type it into the top text area and click the Search button. Then write down how many tries it required to find it.

   5 _____
   16 _____
   19 _____
   25 _____
   37 _____
   44 _____
   56 _____
   62 _____
   79 _____
   81 _____
   99 _____
   100 _____
   105 _____
   117 _____
   200 _____
   300 _____

3) What is the greatest number of tries needed? _____

   What is the least number needed? _____

   What number of tries appears most frequently in your result list? _____

4) There are 16 values in the list that is searched. Suppose you had 32 values. What do you guess would be the most frequent number of searches? _____

   What if your list had 64 values? _____