10.4 Selection Sort

In using an insertion sort, values are constantly being moved. A sorting technique that reduces the amount of data movement is the selection sort. Here, we again perform a number of passes through the data and, after each pass, we again place one more item into an ordered sequence. Now, however, once an item has been placed in its position in the ordered sequence, it is never moved again.

We can begin a selection sort by scanning all the values, finding the largest one, and placing it at the top of the list, exchanging it with the item that was originally in this position.

Example 1

```
 To begin sorting the data
7 1 9 3 5 4
we find the largest element, 9, and swap it with 4, the element at the
right
end of the list. The result, after the first pass of a selection sort is
7 1 4 3 5 9
    '-----/
```

On the second pass, all the items except the last are examined to see which of these is the largest and this item is then placed at the right end of this sublist. This pattern continues on subsequent passes; on each one, the largest value among the unsorted items is placed at the top of the sublist.

Example 2

Using the set of data shown in Example 1, the table shows the results of successive passes of selection sort. The vertical bars indicate the division points between the unsorted items and the sorted items.

After Pass

1	7	1	4	3	5	**| 9**
2	5	1	4	3	**| 7**	9
3	3	1	4	**| 5**	7	9
4	3	1	**| 4**	5	7	9
5	**| 1**	3	4	5	7	9

Notice that only five passes are required to sort six items. Once all but one of the items are placed in their correct positions, the remaining item must also be in its correct position.

To code this algorithm in Java, we note that, for an array called `list`, we must successively find the largest item in sublists of sizes `list.length`, `list.length-1`, ... , `2`. Since the upper bound of an array has index that is one less than the size of the array, the loop that controls the passes of the sort will have the form

```
for (int top = list.length - 1;top> 0; top--)
   // locate largest item and then
   // swap it with item at list[top]
```

We have already seen how to find the largest element in an array and how to swap elements in an array. The next example shows the selection sort that results when we put these pieces together.

Example 3

The method selectSort uses a selection sort to arrange an array of **double** values in ascending order.

```
public static void selectSort (double[] list)
{
   for (int top = list.length - 1; top> 0; top--)
   {
      int largeLoc = 0;
      for (int i = 1; i <= top; i++)
         if (list[i] > list[largeLoc])
            largeLoc = i;

      double temp = list [top] ;
      list[top] = list [largeLoc] ;
      list[largeLoc] = temp;
   }
}
```

Exercise 10.4

1. If a selection sort were to be used to sort the data shown below in alphabetical order, show the data after each pass of the sort.

   Renee Brien Vincent Doris Scarlett

```
public class Question1 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		String [] names =
{"Renee","Brien","Vincent","Doris","Scarlett"};
		selectSort(names);
		for (int x = 0; x < names.length; x++)
			System.out.println(names[x]);


	}

	public static void selectSort (String [] list)
	{
		for (int top = list.length -1; top >=0; top --)
```

```
		{
			int largeLoc = 0;
			for (int x = 0; x <= top; x++)
			{
				if (list[x].charAt(0) > list[top].charAt(0))
					largeLoc = x;
				String temp = list[largeLoc];
				list[largeLoc] = list[top];
				list[top] = temp;
			}
		}
	}

}
```

2. In the `selectSort` method shown in Example 3, what would happen if the expression `list [i] > list [largeLoc]` were to be changed to `list[i] < list[largeLoc]`?

   The array would be sorted in a descending order instead of descending.

```
public class Question2 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		double [] list = {7,1,4,3,5,9};
		selectSort(list);
		for (int x = 0; x < list.length; x++)
			System.out.print(list[x] + "\t");

	}

	public static void selectSort (double[] list)
	{
	   for (int top = list.length - 1; top> 0; top--)
	   {
	      int largeLoc = 0;
	      for (int i = 1; i <= top; i++)
	         if (list[i] < list[largeLoc])
	            largeLoc = i;

	      double temp = list [top] ;
	      list[top] = list [largeLoc] ;
	      list[largeLoc] = temp;
	   }
	}
}
```

3. In our version of selection sort, if the largest item is already at location `top` in the list, then the method still swaps that value with itself, even though that is not necessary.

a. How could the method be changed to avoid this unnecessary swapping?

You could add an `if` statement such as `if (list[largeLoc] != list[top])`. This statement checks if largest location is not already at the location where it should belong after being sorted. Hence, if the element at index `largeLoc` is already at the right index, the unnecessary swapping will not take place.

```
public class Question3 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		double [] list = {10,7,1,4,3,5,9};
		selectSort(list);
		for (int x = 0; x < list.length; x++)
			System.out.print(list[x] + "\t");


	}
	public static void selectSort (double[] list)
	{
	   for (int top = list.length - 1; top> 0; top--)
	   {
	      int largeLoc = 0;
	      for (int i = 1; i <= top; i++)
	         if (list[i] > list[largeLoc])
	            largeLoc = i;

	      if (list[largeLoc] != list[top]) {
	         double temp = list [top] ;
	             list[top] = list [largeLoc] ;
	             list[largeLoc] = temp;
	      }
	   }
	}


}
```

b. Why might it be better to leave the method as it is in the text?

It might be better to leave the method as it is in text in order to avoid any complicated errors.

4. On each pass of our version of selection sort, the *largest* value among the remaining unsorted items was placed in its correct position. An alternate form of the algorithm uses each pass to place the *smallest* value among the remaining unsorted values in its correct position.
   a. Given the set of data.

8 9 6 1 2 4

show the data as they would appear after each pass of a selection sort using this algorithm.

```
public class Question4 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		double [] list = {8,9,6,1,2,4};
		selectSort(list);
		for (int x = 0; x < list.length; x++)
			System.out.print(list[x] + "\t");
	}

	public static void selectSort (double[] list) {
		for (int top = 0; top <= list.length - 1; top++)
		{
			int smallLoc = list.length-1;
			for (int i = list.length - 1; i >= top; i--) {
				if (list[i] < list[smallLoc])
					smallLoc = i;
			}

			double temp = list[top];
			list[top] = list[smallLoc];
			list[smallLoc] = temp;
		}
	}
}
```

b. Write a Java method that implements this algorithm to sort an array of `int` values.

```
public static void selectSort (int [] list) {
		for (int top = 0; top <= list.length - 1; top++)
		{
			int smallLoc = list.length-1;
			for (int i = list.length - 1; i >= top; i--) {
				if (list[i] < list[smallLoc])
					smallLoc = i;
			}

			int temp = list[top];
			list[top] = list[smallLoc];
			list[smallLoc] = temp;
		}
```

```
}
```

5. Sometimes we are only interested in knowing the values that would occupy one end of the list if the list were sorted. As an example, we may want to know the scores of only the top ten competitors in a contest. Modify the selection sort of Example 3 so that, instead of sorting the entire array, it puts the *k* largest values in order in the last *k* positions in the array. The value of *k* should be a parameter of the method.

```
public class Question5 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		double [] list = {8,9,6,1,2,4};
		selectSort(list,2);
	}

	public static void selectSort (double[] list, int value) {
		for (int top = 0; top <= list.length - 1; top++)
		{
			int smallLoc = list.length-1;
			for (int i = list.length - 1; i >= top; i--) {
				if (list[i] < list[smallLoc])
					smallLoc = i;
			}

			double temp = list[top];
			list[top] = list[smallLoc];
			list[smallLoc] = temp;
		}

		if (value <= list.length-1)
			for (int x = list.length - 1 - value; x < list.length-
1; x++)
				System.out.print(list[x] + "\t");

		else
			for (int x = 0; x < list.length-1; x++)
				System.out.print(list[x] + "\t");

	}

}
```