19.6 Selection Sort

Selection sort is a simple, but inefficient, sorting algorithm. If you’re sorting in increasing order, its first iteration selects the *smallest* element in the array and swaps it with the first element. The second iteration selects the *second-smallest* item (which is the smallest item of the remaining elements) and swaps it with the second element. The algorithm continues until the last iteration selects the *second-largest* element and swaps it with the second-to-last index, leaving the largest element in the last index. After the *i*th iteration, the smallest *i* items of the array will be sorted into increasing order in the first *i* elements of the array.

As an example, consider the array

```
34  56  4  10  77  51  93  30  5  52
```

A program that implements selection sort first determines the smallest element (4) of this array, which is contained in index 2. The program swaps 4 with 34, resulting in

```
4  56  34  10  77  51  93  30  5  52
```

The program then determines the smallest value of the remaining elements (all elements except 4), which is 5, contained in index 8. The program swaps 5 with 56, resulting in

```
4  5  34  10  77  51  93  30  56  52
```

On the third iteration, the program determines the next smallest value (10) and swaps it with 34.

```
4  5  10  34  77  51  93  30  56  52
```

The process continues until the array is fully sorted.

```
4  5  10  30  34  51  52  56  77  93
```

After the first iteration, the smallest element is in the first position. After the second iteration, the two smallest elements are in order in the first two positions. After the third iteration, the three smallest elements are in order in the first three positions.

19.6.1 Selection Sort Implementation

Class `SelectionSortTest` (Fig. 19.4) contains:

- `static` method `selectionSort` to sort an `int` array using the selection sort algorithm,
- `static` method `swap` to swap the values of two array elements,
- `static` method `printPass` to display the array contents after each pass, and
- `main` to test method `selectionSort`.

8

In method `main`, line 61 uses Java SE 8 streams to create an array of `int`s named `data` and populate it with random `int`s in the range `10`–`99`. Line 64 calls method `selectionSort` to sort the array's elements into ascending order.

```
 1  // Fig. 19.4: SelectionSortTest.java
 2  // Sorting an array with selection sort.
 3  import java.security.SecureRandom;
 4  import java.util.Arrays;
 5
 6  public class SelectionSortTest {
 7     // sort array using selection sort
 8     public static void selectionSort(int[] data)
 9        // loop over data.length -1 elements
10        for (int i = 0; i < data.length - 1; i++)
11           int smallest = i; // first index of rem
12
13           // loop to find index of smallest eleme
14           for (int index = i + 1; index < data.le
15              if (data[index] < data[smallest]) {
16                 smallest = index;
17              }
18           }
19
20           swap(data, i, smallest); // swap smalle
21           printPass(data, i + 1, smallest); // ou
```

```
      }
   }

   // helper method to swap values in two elemen
   private static void swap(int[] data, int firs
      int temporary = data[first]; // store firs
      data[first] = data[second]; // replace fir
      data[second] = temporary; // put temporary
   }

   // print a pass of the algorithm
   private static void printPass(int[] data, int
      System.out.printf("after pass %2d: ", pass

      // output elements till selected item
      for (int i = 0; i < index; i++) {
         System.out.printf("%d  ", data[i]);
      }

      System.out.printf("%d* ", data[index]); //

      // finish outputting array
      for (int i = index + 1; i < data.length; i
         System.out.printf("%d  ", data[i]);
      }

      System.out.printf("%n  "); // for alignmen

      // indicate amount of array that’s sorted
      for (int j = 0; j < pass; j++) {
         System.out.print("--  ");
      }
      System.out.println();
   }

   public static void main(String[] args) {
      SecureRandom generator = new SecureRandom(

      // create unordered array of 10 random int
      int[] data = generator.ints(10, 10, 91).to
```

```
        System.out.printf("Unsorted array: %s%n%n"
          selectionSort(data); // sort array
        System.out.printf("%nSorted array: %s%n",
    }
 }
```

```
Unsorted array: [40, 60, 59, 46, 98, 82, 23, 51, 31,
after pass  1: 23  60  59  46  98  82  40* 51  31  36
                               --
after pass  2: 23  31  59  46  98  82  40  51  60* 36
                             --  --
after pass  3: 23  31  36  46  98  82  40  51  60  59
                           --  --  --
```

```
after pass  4: 23  31  36  40  98  82  46* 51  60  59
                         --  --  --  --
after pass  5: 23  31  36  40  46  82  98* 51  60  59
                       --  --  --  --  --
after pass  6: 23  31  36  40  46  51  98  82* 60  59
                     --  --  --  --  --  --
after pass  7: 23  31  36  40  46  51  59  82  60  98
                   --  --  --  --  --  --  --
after pass  8: 23  31  36  40  46  51  59  60  82* 98
                 --  --  --  --  --  --  --  --
after pass  9: 23  31  36  40  46  51  59  60  82* 98
               --  --  --  --  --  --  --  --  --
Sorted array: [23, 31, 36, 40, 46, 51, 59, 60, 82, 98
```

Fig. 19.4

Sorting an array with selection sort.

Methods `selectionSort` and `swap`

Lines 8–23 declare the `selectionSort` method. Lines 10–22 loop `data.length - 1` times. Line 11 declares and initializes (to the current index `i`) the variable `smallest`, which stores the index of the smallest element in the remaining array. Lines 14–18 loop over the remaining elements in the array. For each of these elements, line 15 compares its value to the value of the smallest element. If the current element is smaller than the smallest element, line 16 assigns the current element's index to `smallest`. When this loop finishes, `smallest` will contain the index of the smallest element in the remaining array. Line 20 calls method `swap` (lines 26–30) to place the smallest remaining element in the next ordered spot in the array.

Methods `printPass`

Method `printPass` uses dashes (lines 51–53) to indicate the array's sorted portion after each pass. An asterisk is placed next to the position of the element that was swapped with the smallest element on that pass. On each pass, the element next to the asterisk (specified at line 41) and the element above the rightmost set of dashes were swapped.

19.6.2 Efficiency of the Selection Sort

The selection sort algorithm runs in $O(n^2)$ time. Method `selectionSort` uses nested `for` loops. The outer one (lines 10–22) iterates over the first $n-1$ elements in the array, swapping the smallest remaining item into its sorted position. The inner loop (lines 14–18) iterates over each item in the remaining array, searching for the smallest element. This loop executes $n-1$ times during the first iteration of the outer loop, $n-2$ times during the second iteration, then $n-3,\ \cdots,\ 3,\ 2,\ 1$. This inner loop will iterate a total of $n(n-1)/2$ or $(n^2-n)/2$. In Big O notation, smaller terms drop out and constants are ignored, leaving a Big O of $O(n^2)$.