7.4 Sorting and Searching

- Sorting a list of elements is another very common problem (along with searching a list)
  - » sort numbers in ascending order
  - » sort numbers in descending order
  - » sort strings in alphabetic order
  - » etc.

- There are many ways to sort a list, just as there are many ways to search a list

- ***Selection sort***
  - » one of the easiest
  - » not the most efficient, but easy to understand and program

Selection Sort Algorithm for an Array of Integers

To sort an array on integers in ascending order:

- search the array <u>for the smallest number</u> and record its index
- **swap (interchange)** the smallest number with the first element of the array
  - » the sorted part of the array is now the first element
  - » the unsorted part of the array is the remaining elements
- search <u>the remaining unsorted part</u> of the array for the next smallest element and record that element's index
- swap the next smallest element with the second element of the array
- repeat the search and swap until all elements have been placed
  - » each iteration of the search/swap process increases the length of the sorted part of the array by one, and reduces the unsorted part of the array by one

Selection Sort: Diagram of an Example

Key:

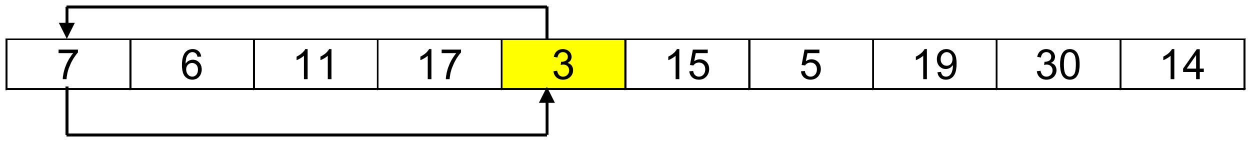
🟨 smallest remaining value

⬜ sorted elements

Problem: sort this 10-element array of integers in ascending order:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
7	6	11	17	3	15	5	19	30	14

<u>1st iteration</u>: smallest value is 3, its index is 4, swap a[0] with a[4]



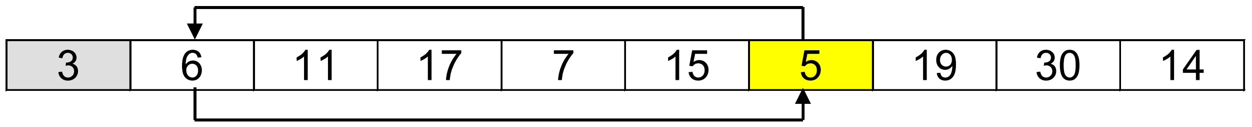
before:

7	6	11	17	🟨 3	15	5	19	30	14

after:

⬜ 3	6	11	17	7	15	5	19	30	14

<u>2nd iteration</u>: smallest value in remaining list is 5, its index is 6, swap a[1] with a[6]



⬜ 3	6	11	17	7	15	🟨 5	19	30	14

⬜ 3	⬜ 5	11	17	7	15	6	19	30	14

<u>Etc.</u> - only nine iterations are required since the last one will put the last *two* entries in place by swapping them if necessary.

Selection Sort Code

```
/**************************************************
*Precondition:
*Every indexed variable of the array a has a value.
*Action: Sorts the array a so that
*a[0] <= a[1] <= ... <= a[a.length - 1].
**************************************************/
public static void selectionSort(int[] a)
{
   int index, indexOfNextSmallest;
   for (index = 0; index < a.length - 1; index++)
   {//Place the correct value in a[index]:
       indexOfNextSmallest = getIndexOfSmallest(index, a);
       interchange(index,indexOfNextSmallest, a);
       //a[0] <= a[1] <=...<= a[index] and these are
       //the smallest of the original array elements.
       //The remaining positions contain the rest of
       //the original array elements.
   }
}
```

Example: Selection Sort

- The `SelectionSort` program in the text shows a class for sorting an array of `int`s in ascending order
- Notice the precondition: every indexed variable has a value
- Also notice that the array may have duplicate values and the class handles them in a reasonable way - they are put in sequential positions
- Finally, notice that the problem was broken down into smaller tasks, such as "find the index of the smallest value" and "interchange two elements"
  - » these subtasks are written as separate methods and are because they are helper methods (users are not expected to call them directly)

Listing 7.10 Selection Sort Class - SelectionSort.java

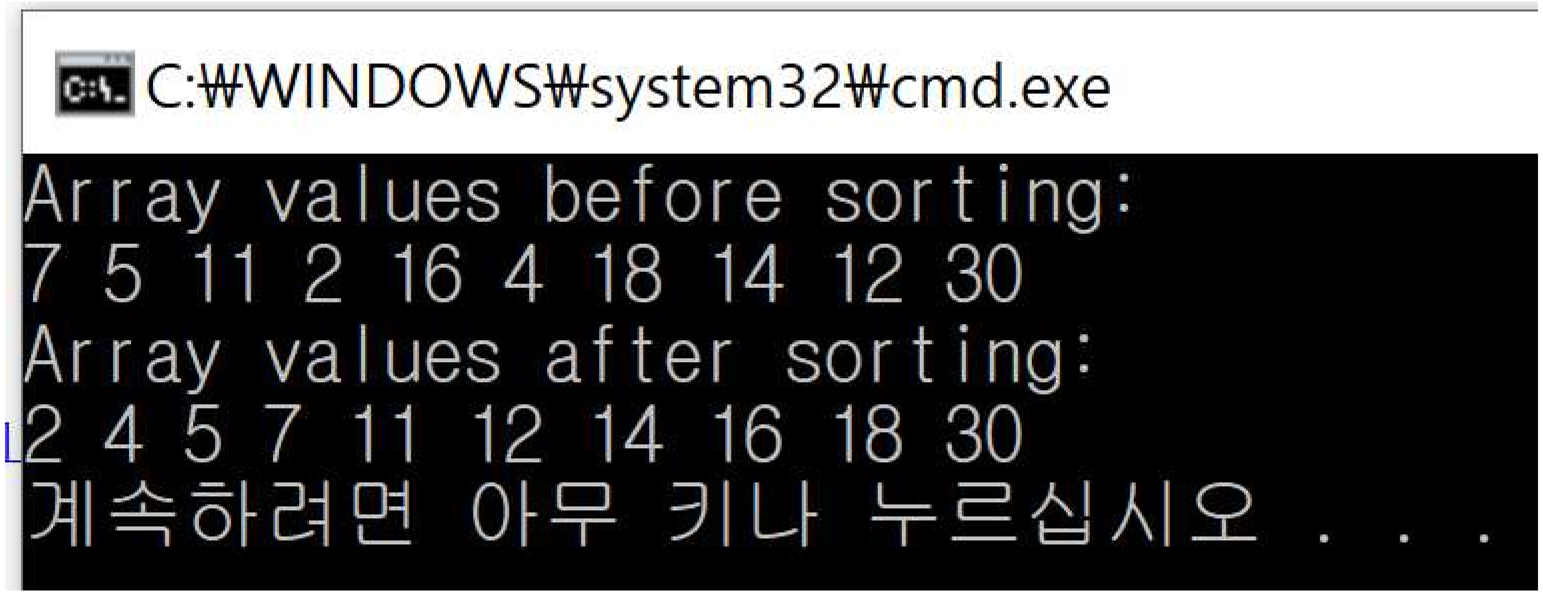
```
/**
 Class for sorting an array of base type int from smallest to largest.
*/
public class ArraySorter
{
    /**
     Precondition: Every element in anArray has a value.
     Action: Sorts the array into ascending order.
          */
    public static void selectionSort(int[] anArray)
    {
        for (int index = 0; index < anArray.length - 1; index++)
        {   // Place the correct value in anArray[index]
            int indexOfNextSmallest = getIndexOfSmallest(index, anArray);
            interchange(index, indexOfNextSmallest, anArray);
            //Assertion:anArray[0] <= anArray[1] <=...<= anArray[index]
            //and these are the smallest of the original array elements.
            //The remaining positions contain the rest of the original
            //array elements.
        }
    }
/**
  Returns the index of the smallest value in the portion of the
  array that begins at the element whose index is startIndex and
  ends at the last element.
 */
 private static int getIndexOfSmallest(int startIndex, int[] a)
 {
     int min = a[startIndex];
     int indexOfMin = startIndex;
     for (int index =                ; index <          ; index++)
     {
         if (a[index] < min)
         {
             min = a[index];
             indexOfMin = index;
             // Assertion: min is smallest of a[startIndex] through a[index]
         }
                 }
     return indexOfMin;
 }
 /**
  Precondition: i and j are valid indices for the array a.
  Postcondition: Values of a[i] and a[j] have been interchanged.
 */
 private static void interchange(int i, int j, int[] a)
 {
     int temp = a[i];
     a[i] = a[j];
     a[j] = temp; //original value of a[i]
 }
}
```

Listing 7.11 Demonstration of the SelectionSort Class - SelectionSortDemo.java

```
public class SelectionSortDemo
{
    public static void main(String[] args)
    {
        int[] b = {7, 5, 11, 2, 16, 4, 18, 14, 12, 30};

             display (b, "before");
             ArraySorter.[redacted];
             display (b, "after");
    }

     public static void display(int[] array, String when)
             {
        System.out.println("Array values " + when + " sorting:");
        for (int i = 0; i < array.length; i++)
            System.out.print(array[i] + " ");
        System.out.println( );
             }
}
```



```
Array values before sorting:
7 5 11 2 16 4 18 14 12 30
Array values after sorting:
2 4 5 7 11 12 14 16 18 30
계속하려면 아무 키나 누르십시오 . . .
```

끝