Chapter 14:
Sorting and searching

Chapter Goals
• To study several sorting and searching algorithms
• To appreciate that algorithms for the same task can differ widely in performance
• To learn how to estimate and compare the performance of algorithms
• To learn how to measure the running time of a program

Selection Sort
• Sorts an array by repeatedly finding the smallest element of the unsorted tail region and moving it to the front
• Slow when run on large data sets
• Example: sorting an array of integers

<table>
<thead>
<tr>
<th>12</th>
<th>5</th>
<th>17</th>
<th>9</th>
<th>11</th>
</tr>
</thead>
</table>

Sorting an Array of Integers
• Find the smallest and swap it with the first element

<table>
<thead>
<tr>
<th>5</th>
<th>9</th>
<th>17</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
</table>

• Find the next smallest. It is already in the correct place

<table>
<thead>
<tr>
<th>5</th>
<th>9</th>
<th>17</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
</table>

• Find the next smallest and swap it with first element of unsorted portion

<table>
<thead>
<tr>
<th>5</th>
<th>9</th>
<th>11</th>
<th>17</th>
<th>12</th>
</tr>
</thead>
</table>

• Repeat

<table>
<thead>
<tr>
<th>5</th>
<th>9</th>
<th>11</th>
<th>12</th>
<th>17</th>
</tr>
</thead>
</table>

• When the unsorted portion is of length 1, we are done

| 5 | 9 | 11 | 12 | 17 |
This class sorts an array, using the selection sort algorithm.

public class SelectionSorter
{
    /**
     * Constructs a selection sorter.
     * @param anArray the array to sort
     */
    public SelectionSorter(int[] anArray)
    {
        a = anArray;
    }

    /**
     * Sorts the array managed by this selection sorter.
     */
    public void sort()
    {
        for (int i = 0; i < a.length - 1; i++)
        {
            int minPos = minimumPosition(i);
            swap(minPos, i);
        }
    }

    /**
     * Finds the smallest element in a tail range of the array.
     * @param from the first position in a to compare
     * @return the position of the smallest element in the range a[from] . . . a[a.length - 1]
     */
    private int minimumPosition(int from)
    {
        int minPos = from;
        for (int i = from + 1; i < a.length; i++)
            if (a[i] < a[minPos]) minPos = i;
        return minPos;
    }

    /**
     * Swaps two entries of the array.
     * @param i the first position to swap
     * @param j the second position to swap
     */
    private void swap(int i, int j)
    {
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }

    private int[] a;
}

import java.util.Arrays;

/**
 * This program demonstrates the selection sort algorithm by sorting an array that is filled with random numbers.
 */
public class SelectionSortDemo
{
    public static void main(String[] args)
    {
        int[] a = ArrayUtil.randomIntArray(20, 100);
        System.out.println(Arrays.toString(a));

        SelectionSorter sorter = new SelectionSorter(a);
        sorter.sort();

        System.out.println(Arrays.toString(a));
    }
}
Self Check 14.1

Why do we need the temp variable in the swap method? What would happen if you simply assigned \( a[i] \) to \( a[j] \) and \( a[j] \) to \( a[i] \)?

**Answer:** Dropping the temp variable would not work. Then \( a[i] \) and \( a[j] \) would end up being the same value.

Profiling the Selection Sort Algorithm

- We want to measure the time the algorithm takes to execute
  - *Exclude the time the program takes to load*
  - *Exclude output time*
- Create a **StopWatch** class to measure execution time of an algorithm
  - *It can start, stop and give elapsed time*
  - *Use System.currentTimeMillis method*
- Create a **StopWatch** object
  - *Start the stopwatch just before the sort*
  - *Stop the stopwatch just after the sort*
  - *Read the elapsed time*
```java
ch14/selsort/StopWatch.java (cont.)

```java
23:     isRunning = true;
24:     startTime = System.currentTimeMillis();
25: }
26: 27: /**
28:     Stops the stopwatch. Time stops accumulating and is
29:     added to the elapsed time.
30: */
31: public void stop()
32: {
33:     if (!isRunning) return;
34:     long endTime = System.currentTimeMillis();
35:     elapsedTime = elapsedTime + endTime - startTime;
36: }
37: 38: /**
39:     Returns the total elapsed time.
40: */
41: public long getElapsedTime()
42: {
43:     if (isRunning) return;
44:     long endTime = System.currentTimeMillis();
45:     return elapsedTime + endTime - startTime;
46: }
47: 48: /**
49:     Stops the watch and resets the elapsed time to 0.
50: */
51: public void reset()
52: {
53:     elapsedTime = 0;
54:     isRunning = false;
55: }
56: 57: private long elapsedTime;
58: 59: private long startTime;
60: 61: private boolean isRunning;
62: }
63: ch14/selsort/SelectionSortTimer.java  (cont.)
64: ```
```java
ch14/selsort/SelectionSortTimer.java

```java
import java.util.Scanner;

```java
01: public class SelectionSortTimer
02: {
03:     public static void main(String[] args)
04:     {
05:         Scanner in = new Scanner(System.in);
06:         System.out.print("Enter array size: ");
07:         int n = in.nextInt();
08:         int[] a = ArrayUtil.randomIntArray(n, 100);
09:         SelectionSorter sorter = new SelectionSorter(a);
10:         StopWatch timer = new StopWatch();
11:         timer.start();
12:         sorter.sort();
13:         timer.stop();
14:         System.out.println("Elapsed time: 
15:             " + timer.getElapsedTime() + " milliseconds");
16:     }
17: }
18: ```
```java
```
Selection Sort on Various Size Arrays

<table>
<thead>
<tr>
<th>n</th>
<th>Milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>786</td>
</tr>
<tr>
<td>20,000</td>
<td>2,148</td>
</tr>
<tr>
<td>30,000</td>
<td>4,796</td>
</tr>
<tr>
<td>40,000</td>
<td>9,192</td>
</tr>
<tr>
<td>50,000</td>
<td>13,321</td>
</tr>
<tr>
<td>60,000</td>
<td>19,299</td>
</tr>
</tbody>
</table>

* Obtained with a Pentium processor, 2 GHz, Java 6, Linux

• Doubling the size of the array nearly quadruples the time needed to sort it

Self Check 14.3

Approximately how many seconds would it take to sort a data set of 80,000 values?

**Answer:** Four times as long as 40,000 values, or about 36 seconds.
Self Check 14.4

Look at the graph in Figure 1. What mathematical shape does it resemble?

**Answer:** A parabola.

---

Public void sort()
{
    for (int i = 0; i < a.length - 1; i++)
    {
        int minPos = minimumPosition(i);
        swap(minPos, i);
    }
}

Private int minimumPosition(int from)
{
    int minPos = from;
    for (int i = from + 1; i < a.length; i++)
    if (a[i] < a[minPos]) minPos = i;
    return minPos;
}

Private void swap(int i, int j)
{
    int temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}

---

The most important parts of SelectionSorter

---

Analyzing the Performance of the Selection Sort Algorithm

- How do we analyze performance? Count the number of operations executed? Counting machine instructions is somewhat awkward. For this program we will count the number of times the elements of the array are visited (accessed or modified).
- In an array of size n, count how many times an array element is visited:
  - To find the smallest, visit 2 *(n – 1) elements. Then another 3 visits for the swap
  - To find the next smallest, visit 2 *(n - 2) elements + 3 visits for the swap
  - The last time, visit 2 * 1 elements + 3 visits for the swap

---

Analyzing the Performance of the Selection Sort Algorithm

- The number of visits:

  \[
  2 (n - 1) + 3 + (n - 2) + 3 + (n - 3) + 3 + \ldots + 2 + 3
  \]

  \[
  = 2 ((n - 1) + (n - 2) + \ldots + 2) + (n - 1) * 3
  \]

  We can use the following identity: 1 + 2 + \ldots + (n - 1) + n = n (n + 1) / 2
  But first we have to get it in the right form:

  \[
  1 + 2 + \ldots + (n - 2) + (n - 1) = n (n - 1) / 2
  \]

  \[
  = 2 ((n - 1) + (n - 2) + \ldots + 2 + 1 - 1) + (n - 1) * 3
  \]

  \[
  = 2 n (n - 1) / 2 - 2 + (n - 1) * 3
  \]

  This can be simplified to \( n^2 + 2n - 5 \)

  When n is large \( 2n - 5 \) is small compared to \( n^2 \) so we will ignore these terms.
Analyzing the Performance of the Selection Sort Algorithm

- The number of visits for large values of n is approximately \( n^2 \)
- Let us compare the number of array visits for two different array sizes. What is the ratio of the number of visits for \( n = 2000 \) versus \( n = 1000 \)?
  \[
  \frac{2000^2}{1000^2} = 4
  \]
  This is in rough agreement with our experimental results.
- We say selection sort is of the order \( n^2 \).

Self Check 14.5

If you increase the size of a data set tenfold, how much longer does it take to sort it with the selection sort algorithm?

**Answer:** It takes about 100 times longer.

Insertion Sort

- Assume initial sequence \( a[0] \ldots a[k] \) is sorted (\( k = 0 \)):
  \[
  11 \ 9 \ 16 \ 5 \ 7
  \]
- Add \( a[1] \); element needs to be inserted before 11
  \[
  9 \ 11 \ 16 \ 5 \ 7
  \]
- Add \( a[2] \)
  \[
  9 \ 11 \ 16 \ 5 \ 7
  \]
- Add \( a[3] \)
  \[
  5 \ 9 \ 11 \ 16 \ 7
  \]
- Finally, add \( a[4] \)
  \[
  5 \ 9 \ 11 \ 16 \ 7
  \]

ch14/insertionsort/InsertionSorter.java

```java
public class InsertionSorter {
    public InsertionSorter(int[] anArray) {
        a = anArray;
    }

    public void sort() {
        for (int i = 1; i < a.length; i++) {
            int next = a[i];
            // Move all larger elements up
            int j = i;
            while (j > 0 && a[j - 1] > next) {
                a[j] = a[j - 1];
                j--;
            }
            // Insert the element
            a[j] = next;
        }
    }

    private int[] a;
}
```
Sorting in a Java Program

- The `Arrays` class implements a sorting method that is applicable to arrays of primitive type
- e.g. To sort an array of integers
  ```java
  int[] a = . . . ;
  Arrays.sort(a);
  ```
- That `sort` method uses the Quicksort algorithm (see Advanced Topic 14.3)

Searching

- Linear search: also called sequential search
- Encountered earlier as one of the simple array algorithms in chapter 7
- Examines all values in an array until it finds a match or reaches the end
- Number of visits for a linear search of an array of \( n \) elements:
  - The average search visits \( n/2 \) elements
  - The maximum visits is \( n \)

```
ch14/linsearch/LinearSearcher.java
01: /**
02: * A class for executing linear searches through an array.
03: */
04: public class LinearSearcher
05: {
06:     /**
07:      * Constructs the LinearSearcher.
08:      * @param anArray an array of integers
09:      */
10:     public LinearSearcher(int[] anArray)
11:     {
12:         a = anArray;
13:     }
14: }
15: /**
16:  * Finds a value in an array, using the linear search algorithm.
17:  * @param v the value to search
18:  * @return the index at which the value occurs, or -1
19:  * if it does not occur in the array
20:  */
21: * Continued
```

```
ch14/linsearch/LinearSearcher.java (cont.)
22: public int search(int v)
23: {
24:     for (int i = 0; i < a.length; i++)
25:     {
26:         if (a[i] == v)
27:             return i;
28:     }
29:     return -1;
30: }
31: private int[] a;
32: * Continued
```
Typical Output:

[46, 99, 45, 57, 64, 95, 81, 69, 11, 97, 6, 85, 61, 88, 29, 65, 83, 88, 45, 88]
Enter number to search for, -1 to quit: 11
Found in position 8

Self Check 14.11
Suppose you need to look through 1,000,000 records to find a telephone number. How many records do you expect to search before finding the number?

Answer: On average, you'd make 500,000 comparisons.

Self Check 14.12
Why can't you use a "for each" loop for (int element : a) in the search method?

Answer: The search method returns the index at which the match occurs, not the data stored at that location.

Binary Search

- Binary search is a faster form of search that is possible only if the array to search has already been sorted
- Locates a value in a sorted array by,
  - Determining whether the value occurs in the first or second half of the array
  - Then repeating the search in one of the halves
**Binary Search**

- To search for 15:

  \[ \begin{array}{c}
  0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
  1 & 5 & 8 & 9 & 12 & 17 & 20 & 32 \\
  \end{array} \]

- 15 ≠ 17: we don’t have a match

---

**Searching a Sorted Array in a Program**

- The **Arrays** class contains a static **binarySearch** method

  - The method returns either
    - The index of the element, if element is found
    - Or \(k - 1\) where \(k\) is the position before which the element should be inserted

  ```java
  int[] a = { 1, 4, 9 }; int v = 7; int pos = Arrays.binarySearch(a, v); // Returns -3; v should be inserted before position 2
  ```
Self Check 14.14

Why is it useful that the `Arrays.binarySearch` method indicates the position where a missing element should be inserted?

**Answer:** Then you know where to insert it so that the array stays sorted, and you can keep using binary search.

Self Check 14.15

Why does `Arrays.binarySearch` return `-k - 1` and not `-k` to indicate that a value is not present and should be inserted before position `k`?

**Answer:** Otherwise, you would not know whether a value is present when the method returns 0.