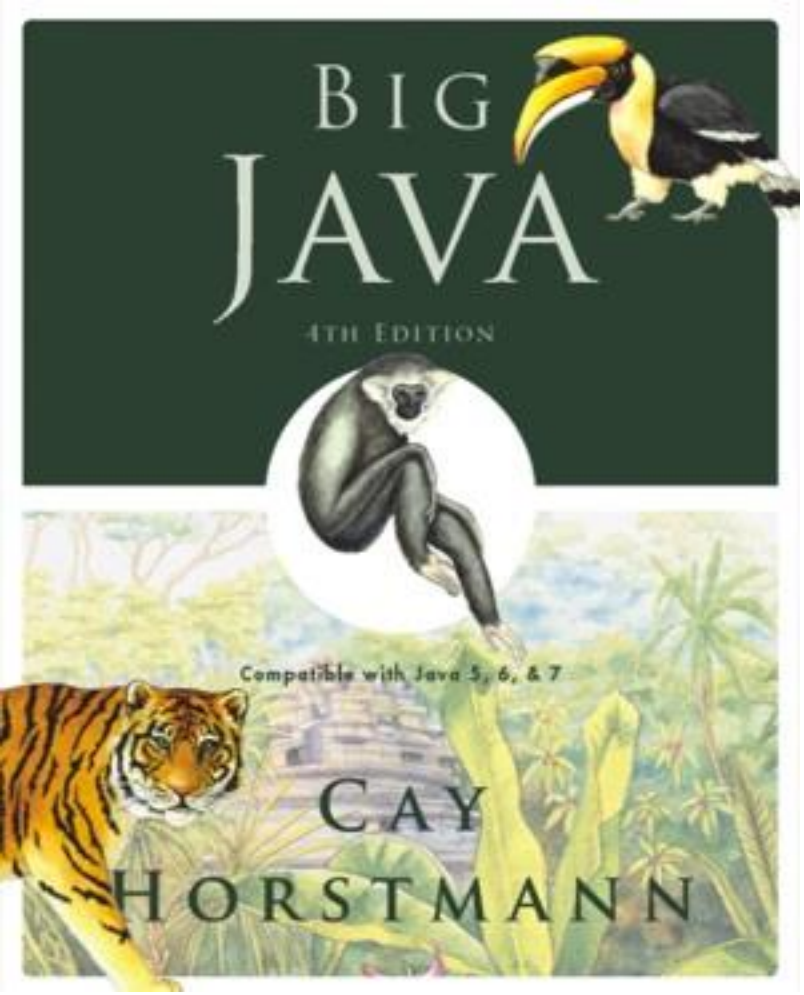



الفصل ١٤ - الفرز والبحث

# 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 understand the big-Oh notation
- To learn how to estimate and compare the performance of algorithms
- To learn how to measure the running time of a program

- لدراسة العديد من الفرز وخوارزميات البحث
- أن نقدر أن خوارزميات لنفس المهمة يمكن تختلف على نطاق واسع في الأداء
- لفهم أوه الكبيرة التدوين
- لمعرفة كيفية تقدير ومقارنة أداء الخوارزميات
- لمعرفة كيفية قياس وقت تشغيل برنامج

# Selection Sort

اختيار ترتيب

- 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

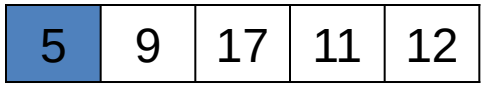
| 11 | 9 | 17 | 5 | 12 |
|---|---|---|---|---|

- يفرز مجموعة من خلال إيجاد مرارا أصغر عنصر؟ منطقة الذيل التي لم يتم فرزها ونقلها إلى الأمام
- بطء عند تشغيل على مجموعات البيانات الكبيرة
- مثال: فرز مجموعة من الأعداد الصحيحة

# Sorting an Array of Integers

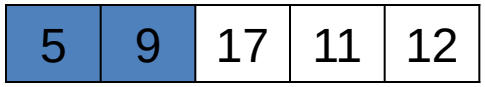
فرز صفيف من الأعداد الصحيحة

- Find the smallest and swap it with the first element




- العثور على أصغر ومبادلة مع العنصر الأول؟

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




- العثور على أصغر المقبل. هو بالفعل في المكان الصحيح

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


- العثور على أصغر المقبل ومبادلة مع العنصر الأول من الجزء فرزها

- Repeat


- كرر؟

- عندما الجزء فرزها هو من طول ١، نحن القيام به

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

# ch14/selsort/SelectionSorter.java

```
1  /**
2     This class sorts an array, using the selection sort
3     algorithm
4  */
5  public class SelectionSorter
6  {
7     private int[] a;
8
9     /**
10       Constructs a selection sorter.
11       @param anArray the array to sort
12    */
13    public SelectionSorter(int[] anArray)
14    {
15       a = anArray;
16    }
17
```

*Continued*

# ch14/selsort/SelectionSorter.java (cont.)

```
18      /**
19         Sorts the array managed by this selection sorter.
20      */
21      public void sort()
22      {
23         for (int i = 0; i < a.length - 1; i++)
24         {
25            int minPos = minimumPosition(i);
26            swap(minPos, i);
27         }
28      }
29
```

*Continued*

# ch14/selsort/SelectionSorter.java (cont.)

```
30      /**
31         Finds the smallest element in a tail range of the array.
32         @param from the first position in a to compare
33         @return the position of the smallest element in the
34         range a[from] . . . a[a.length - 1]
35      */
36      private int minimumPosition(int from)
37      {
38         int minPos = from;
39         for (int i = from + 1; i < a.length; i++)
40            if (a[i] < a[minPos]) minPos = i;
41         return minPos;
42      }
43
44      /**
45         Swaps two entries of the array.
46         @param i the first position to swap
47         @param j the second position to swap
48      */
49      private void swap(int i, int j)
50      {
51         int temp = a[i];
52         a[i] = a[j];
53         a[j] = temp;
54      }
55   }
```

# ch14/selsort/SelectionSortDemo.java

```
 1  import java.util.Arrays;
 2
 3  /**
 4     This program demonstrates the selection sort algorithm by
 5     sorting an array that is filled with random numbers.
 6  */
 7  public class SelectionSortDemo
 8  {
 9     public static void main(String[] args)
10     {
11        int[] a = ArrayUtil.randomIntArray(20, 100);
12        System.out.println(Arrays.toString(a));
13
14        SelectionSorter sorter = new SelectionSorter(a);
15        sorter.sort();
16
17        System.out.println(Arrays.toString(a));
18     }
19  }
20
21
```

# ch14/selsort/ArrayUtil.java

```
1  import java.util.Random;
2
3  /**
4     This class contains utility methods for array manipulation.
5  */
6  public class ArrayUtil
7  {
8     private static Random generator = new Random();
9
10    /**
11       Creates an array filled with random values.
12       @param length the length of the array
13       @param n the number of possible random values
14       @return an array filled with length numbers between
15       0 and n - 1
16    */
17    public static int[] randomIntArray(int length, int n)
18    {
19       int[] a = new int[length];
20       for (int i = 0; i < a.length; i++)
21          a[i] = generator.nextInt(n);
22
23       return a;
24    }
25 }
```

# ch14/selsort/ArrayUtil.java (cont.)

## Typical Program Run:

```
[65, 46, 14, 52, 38, 2, 96, 39, 14, 33, 13, 4, 24, 99, 89, 77, 73, 87, 36, 81]
[2, 4, 13, 14, 14, 24, 33, 36, 38, 39, 46, 52, 65, 73, 77, 81, 87, 89, 96, 99]
```



*Big Java* by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

# 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.

لماذا نحتاج إلى متغير مؤقت في أسلوب المبادلة؟ ماذا سيحدث لو كنت ببساطة تعيين `a[i]` to `a[j]` and `a[j]` to `a[i]`?

الإجابة: إسقاط متغير درجة الحرارة لن يجدي نفعا. ثم [ a[i]and a[j]أن ينتهي الأمر به إلى نفس القيمة.

# Self Check 14.2

What steps does the selection sort algorithm go through to sort the sequence `6 5 4 3 2 1`?

**Answer:**

| 1 | 5 | 4 | 3 | 2 | 6 |
|---|---|---|---|---|---|

| 1 | 2 | 4 | 3 | 5 | 6 |
|---|---|---|---|---|---|

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|

فما هي الخطوات التي خوارزمية الفرز اختيار تذهب من خلال لفرز تسلسل ٦ ٥ ٤ ٣ ٢ ١؟

# 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*

- نحن نريد لقياس الوقت المستغرق لتنفيذ الخوارزمية
- استبعاد الوقت ويأخذ البرنامج لتحميل
- استبعاد وقت الإخراج
- إنشاء فئة ساعة توقيت لقياس وقت تنفيذ خوارزمية
- ويمكن أن تبدأ، والتوقف عن إعطاء الوقت المنقضي
- طريقة استخدام System.currentTimeMillis
- إنشاء كائن ساعة توقيت
- بدء تشغيل ساعة التوقيت فقط قبل الفرز
- وقف ساعة توقيت فقط بعد الفرز
- قراءة الوقت المنقضي

## ch14/selsort/StopWatch.java

```
1  /**
2     A stopwatch accumulates time when it is running. You can
3     repeatedly start and stop the stopwatch. You can use a
4     stopwatch to measure the running time of a program.
5  */
6  public class StopWatch
7  {
8     private long elapsedTime;
9     private long startTime;
10    private boolean isRunning;
11
12    /**
13       Constructs a stopwatch that is in the stopped state
14       and has no time accumulated.
15    */
16    public StopWatch()
17    {
18       reset();
19    }
20
```

***Continued***

# ch14/selsort/StopWatch.java (cont.)

```
21      /**
22         Starts the stopwatch. Time starts accumulating now.
23      */
24      public void start()
25      {
26         if (isRunning) return;
27         isRunning = true;
28         startTime = System.currentTimeMillis();
29      }
30
31      /**
32         Stops the stopwatch. Time stops accumulating and is
33         is added to the elapsed time.
34      */
35      public void stop()
36      {
37         if (!isRunning) return;
38         isRunning = false;
39         long endTime = System.currentTimeMillis();
40         elapsedTime = elapsedTime + endTime - startTime;
41      }
42
```

*Continued*

# ch14/selsort/StopWatch.java (cont.)

```
43      /**
44         Returns the total elapsed time.
45         @return the total elapsed time
46      */
47      public long getElapsedTime()
48      {
49         if (isRunning)
50         {
51            long endTime = System.currentTimeMillis();
52            return elapsedTime + endTime - startTime;
53         }
54         else
55            return elapsedTime;
56      }
57
58      /**
59         Stops the watch and resets the elapsed time to 0.
60      */
61      public void reset()
62      {
63         elapsedTime = 0;
64         isRunning = false;
65      }
66   }
```

# ch14/selsort/SelectionSortTimer.java

```
 1  import java.util.Scanner;
 2
 3  /**
 4     This program measures how long it takes to sort an
 5     array of a user-specified size with the selection
 6     sort algorithm.
 7  */
 8  public class SelectionSortTimer
 9  {
10     public static void main(String[] args)
11     {
12        Scanner in = new Scanner(System.in);
13        System.out.print("Enter array size: ");
14        int n = in.nextInt();
15
16        // Construct random array
17
18        int[] a = ArrayUtil.randomIntArray(n, 100);
19        SelectionSorter sorter = new SelectionSorter(a);
20
```

# ch14/selsort/SelectionSortTimer.java (cont.)

```
21          // Use stopwatch to time selection sort
22
23          StopWatch timer = new StopWatch();
24
25          timer.start();
26          sorter.sort();
27          timer.stop();
28
29          System.out.println("Elapsed time: "
30                + timer.getElapsedTime() + " milliseconds");
31       }
32    }
33
34
```

## Program Run:

```
Enter array size: 100000
Elapsed time: 27880 milliseconds
```

# Selection Sort on Various Size Arrays*

اختيار فرز مختلف صالحة الحجم *

| n | Milliseconds |
|---|---|
| 10,000 | 786 |
| 20,000 | 2,148 |
| 30,000 | 4,796 |
| 40,000 | 9,192 |
| 50,000 | 13,321 |
| 60,000 | 19,299 |

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

* حصل مع معالج بنتيوم، ٢ غيغاهرتز، جافا ٦، لينكس

# Selection Sort on Various Size Arrays

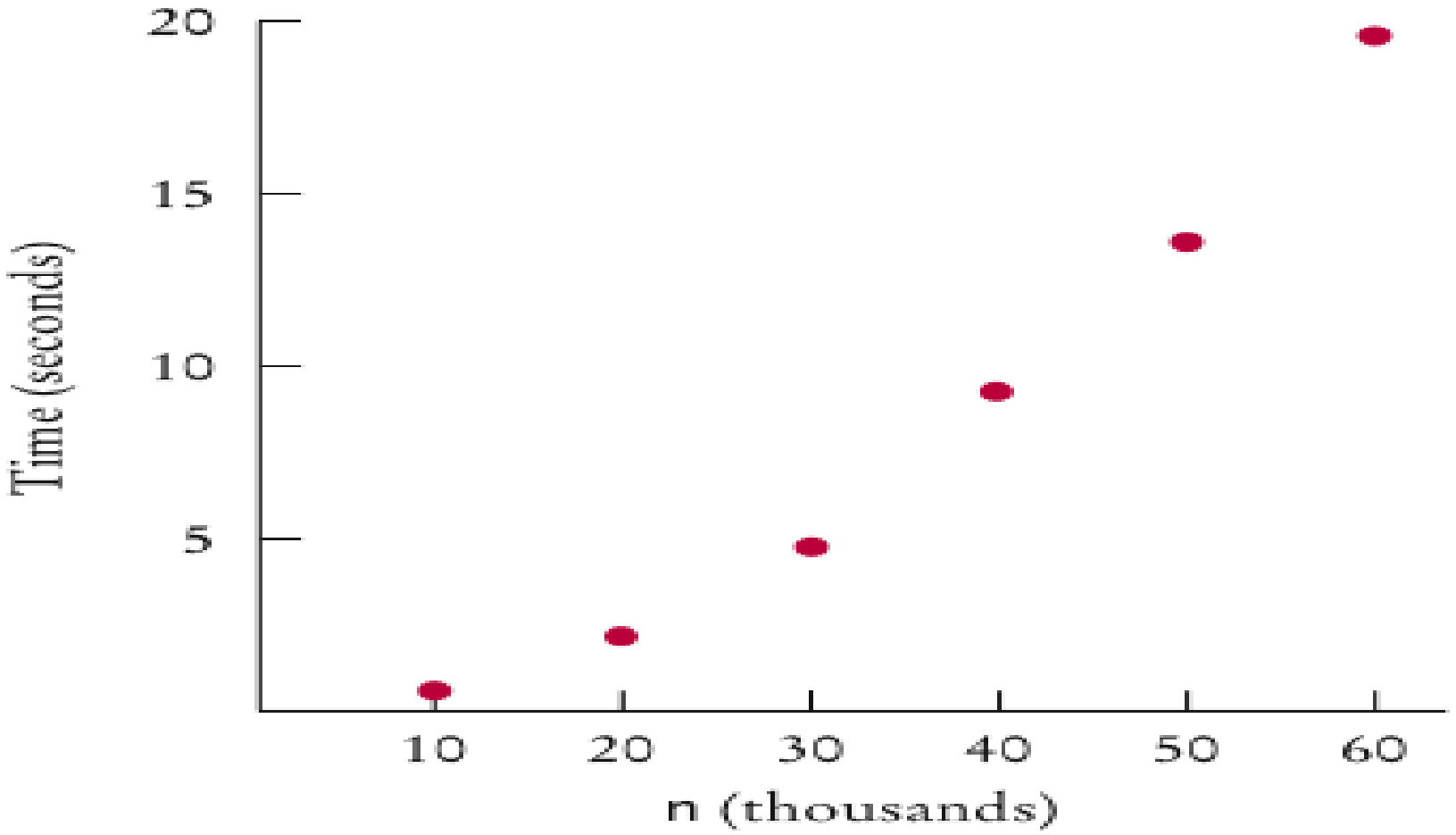



**Figure 1** Time Taken by Selection Sort

# Selection Sort on Various Size Arrays

- Doubling the size of the array more than doubles the time needed to sort it
  - مضاعفة حجم المصفوفة أكثر من يضاعف من الوقت اللازم لترتيب هذا الامر

## 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.

- تقريبا كم ثانية سوف يستغرق لفرز مجموعة بيانات من ٨٠٠٠٠ القيم؟
- الإجابة: أربع مرات طالما ٤٠٠٠٠ القيم، أو حوالي ٣٦ ثانية.

# Self Check 14.4

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

**Answer:** A parabola.

- نظرة على الرسم البياني في الشكل ١. ما شكل رياضي أنها لا تشبه؟
- الجواب: القطع المكافئ

# Analyzing the Performance of the Selection Sort Algorithm

تحليل أداء اختيار خوارزمية ترتيب

- In an array of size *n*, count how many times an array element is visited
  - *To find the smallest, visit n elements + 2 visits for the swap*
  - *To find the next smallest, visit (n - 1) elements + 2 visits for the swap*
  - *The last term is 2 elements visited to find the smallest + 2 visits for the swap*

- في مجموعة من حجم ن، عد كم مرة زار عنصر صفيف
- العثور على أصغر، العناصر زيارة ن + ٢ مرة للمبادلة
- العثور على أصغر المقبلة، زيارة (ن - ١) عناصر + ٢ مرة للمبادلة
- في الموسم الماضي هو ٢ عناصر زار العثور على أصغر + ٢ مرة للمبادلة

# Analyzing the Performance of the Selection Sort Algorithm

- The number of visits:
  - *n + 2 + (n - 1) + 2 + (n - 2) + 2 + ...+ 2 + 2*
  - *This can be simplified to $n^2$ /2 + 5n/2 - 3*
  - *5n/2 - 3 is small compared to $n^2$ /2 — so let's ignore it*
  - *Also ignore the 1/2 — it cancels out when comparing ratios*

- عدد الزيارات:
- ن + ٢ + (ن - ١) + ٢ + (ن - ٢) + ٢ + ... + ٢ + ٢
- وهذا يمكن أن تكون مبسطة إلى N2 / 2 + 5N / 2-3
- 5N / 2-3صغيرة مقارنة - N2 / 2لذلك دعونا تجاهله
- كما تجاهل ٠٢/٠١ - فيه تفويت عند مقارنة نسب

# Analyzing the Performance of the Selection Sort Algorithm

- The number of visits is of the order $n^2$
- Using big-Oh notation: The number of visits is $O(n^2)$
- Multiplying the number of elements in an array by **2** multiplies the processing time by **4**
- Big-Oh notation "$f(n) = O(g(n))$" expresses that $f$ grows no faster than $g$
- To convert to big-Oh notation: Locate fastest-growing term, and ignore constant coefficient

- عدد الزيارات من أجل $n^2$
- استخدام التدوين يا كبير: عدد الزيارات $O(n^2)$
  - ضرب عدد من العناصر في مجموعة بنسبة ٢ يضاعف وقت المعالجة بنسبة ٤
  - كبير يا-التدوين "$f(n) = O(g(n))$" يعبر عن ذلك $f$ ينمو بسرعة لا تزيد $g$
  - تحويل إلى كبير أوه-تدوين: حدد المدى الأسرع نموا، وتجاهل معامل ثابت

# 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.

- إذا قمت بزيادة حجم مجموعة البيانات عشرة أضعاف، كم من الوقت يستغرق لترتيب هذا الامر مع الخوارزمية اختيار نوع؟
- الجواب: يستغرق حوالي ١٠٠ أوقات أطول.

## Self Check 14.6

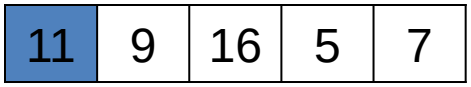
How large does $n$ need to be so that $n^2/2$ is bigger than $5n/2 - 3$?

**Answer:** If $n$ is 4, then $n^2/2$ is 8 and $5n/2 - 3$ is 7.

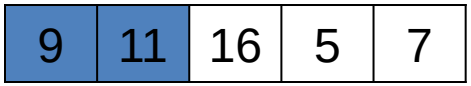
# Insertion Sort

إدراج ترتيب

- Assume initial sequence `a[0] ... a[k]` is sorted `(k = 0)`:




- Add `a[1]`; element needs to be inserted before `11`




- Add `a[2]`


- Add `a[3]`


- Finally, add `a[4]`

# ch14/insertionsort/InsertionSorter.java

```
1  /**
2     This class sorts an array, using the insertion sort
3     algorithm
4  */
5  public class InsertionSorter
6  {
7     private int[] a;
8
9     /**
10       Constructs an insertion sorter.
11       @param anArray the array to sort
12    */
13    public InsertionSorter(int[] anArray)
14    {
15       a = anArray;
16    }
17
```

*Continued*

# ch14/insertionsort/InsertionSorter.java (cont.)

```
18      /**
19         Sorts the array managed by this insertion sorter
20      */
21      public void sort()
22      {
23         for (int i = 1; i < a.length; i++)
24         {
25            int next = a[i];
26            // Move all larger elements up
27            int j = i;
28            while (j > 0 && a[j - 1] > next)
29            {
30               a[j] = a[j - 1];
31               j--;
32            }
33            // Insert the element
34            a[j] = next;
35         }
36      }
37   }
```

# Merge Sort

دمج ترتيب

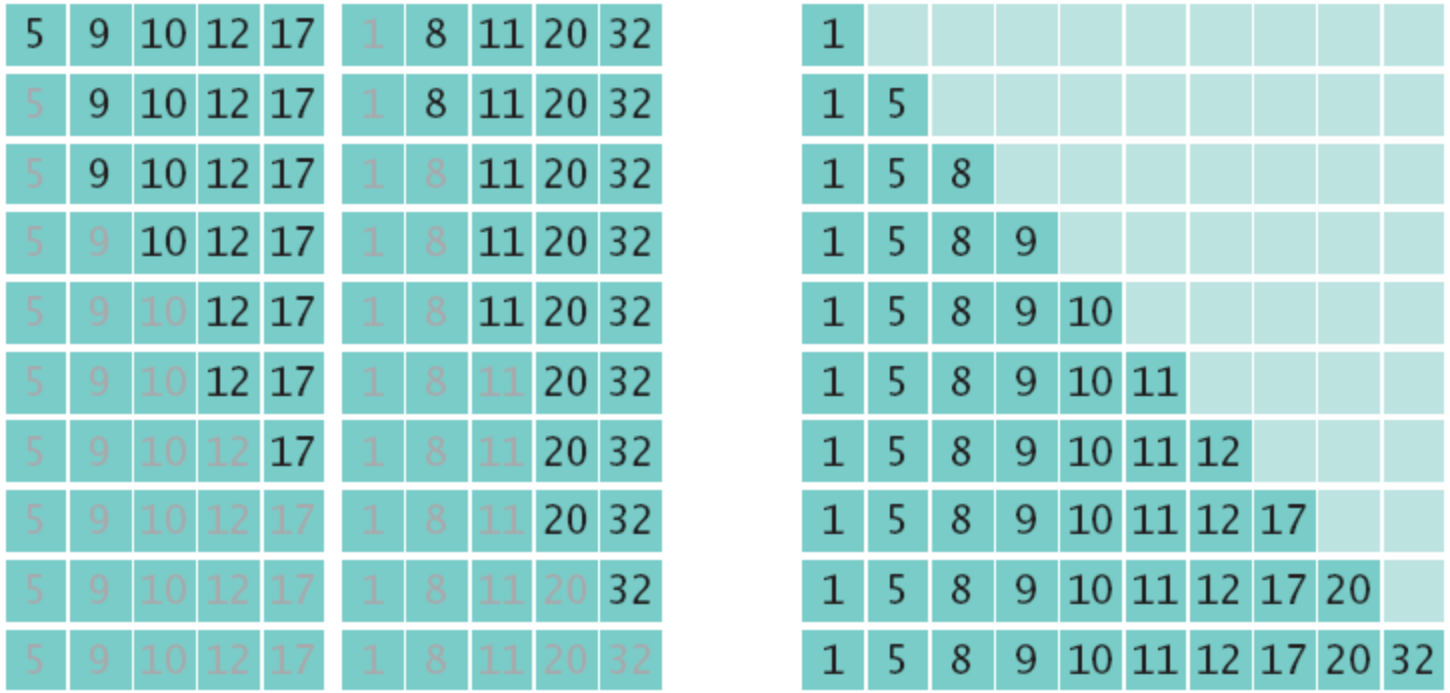
- Sorts an array by
  - *Cutting the array in half*
  - *Recursively sorting each half*
  - *Merging the sorted halves*
- Dramatically faster than the selection sort

- يفرز مجموعة من قبل
- قطع مجموعة في النصف
- فرز متكرر كل شوط
- دمج شطري فرز
- بشكل كبير أسرع من نوع الاختيار

# Merge Sort Example

- Divide an array in half and sort each half تقسيم صفيف في نصف وفرز كل شوط
- دمج اثنين من صفائف فرزها إلى مجموعة فرزها واحدة


- Merge the two sorted arrays into a single sorted array



# Merge Sort

```
public void sort()
{
   if (a.length <= 1) return;
   int [] first = new int[a.length / 2];
   int[] second = new int[a.length - first.length];
   // Copy the first half of a into first, the second half
   // into second
   . . .
   MergeSorter firstSorter = new MergeSorter(first);
   MergeSorter secondSorter = new MergeSorter(second);
   firstSorter.sort();
   secondSorter.sort();
   merge(first, second);
}
```

# ch14/mergesort/MergeSorter.java

```
1  /**
2     This class sorts an array, using the merge sort algorithm.
3  */
4  public class MergeSorter
5  {
6     private int[] a;
7
8     /**
9        Constructs a merge sorter.
10       @param anArray the array to sort
11    */
12    public MergeSorter(int[] anArray)
13    {
14       a = anArray;
15    }
16
```

***Continued***

# ch14/mergesort/MergeSorter.java (cont.)

```
17      /**
18         Sorts the array managed by this merge sorter.
19      */
20      public void sort()
21      {
22         if (a.length <= 1) return;
23         int[] first = new int[a.length / 2];
24         int[] second = new int[a.length - first.length];
25         // Copy the first half of a into first, the second half into second
26         for (int i = 0; i < first.length; i++) { first[i] =
a[i]; }
27         for (int i = 0; i < second.length; i++)
28         {
29            second[i] = a[first.length + i];
30         }
31         MergeSorter firstSorter = new MergeSorter(first);
32         MergeSorter secondSorter = new MergeSorter(second);
33         firstSorter.sort();
34         secondSorter.sort();
35         merge(first, second);
36      }
37
```

*Continued*

# ch14/mergesort/MergeSorter.java (cont.)

```
38      /**
39         Merges two sorted arrays into the array managed by this merge sorter.
40         @param first the first sorted array
41         @param second the second sorted array
42      */
43      private void merge(int[] first, int[] second)
44      {
45         int iFirst = 0; // Next element to consider in the first array
46         int iSecond = 0; // Next element to consider in the second array
47         int j = 0; // Next open position in a
48
49         // As long as neither iFirst nor iSecond is past the end, move
50         // the smaller element into a
51         while (iFirst < first.length && iSecond < second.length)
52         {
53            if (first[iFirst] < second[iSecond])
54            {
55               a[j] = first[iFirst];
56               iFirst++;
57            }
58            else
59            {
60               a[j] = second[iSecond];
61               iSecond++;
62            }
63            j++;
64         }
65
```

***Continued***

# ch14/mergesort/MergeSorter.java (cont.)

```
66          // Note that only one of the two loops below copies entries
67          // Copy any remaining entries of the first array
68          while (iFirst < first.length)
69          {
70             a[j] = first[iFirst];
71             iFirst++; j++;
72          }
73          // Copy any remaining entries of the second half
74          while (iSecond < second.length)
75          {
76             a[j] = second[iSecond];
77             iSecond++; j++;
78          }
79       }
80    }
```

# ch14/mergesort/MergeSortDemo.java

```
1  import java.util.Arrays;
2
3  /**
4     This program demonstrates the merge sort algorithm by
5     sorting an array that is filled with random numbers.
6  */
7  public class MergeSortDemo
8  {
9     public static void main(String[] args)
10    {
11       int[] a = ArrayUtil.randomIntArray(20, 100);
12       System.out.println(Arrays.toString(a));
13
14       MergeSorter sorter = new MergeSorter(a);
15       sorter.sort();
16       System.out.println(Arrays.toString(a));
17    }
18 }
19
```

***Continued***

# ch14/mergesort/MergeSortDemo.java (cont.)

## Typical Program Run:

```
[8, 81, 48, 53, 46, 70, 98, 42, 27, 76, 33, 24, 2, 76, 62, 89, 90, 5, 13, 21]
[2, 5, 8, 13, 21, 24, 27, 33, 42, 46, 48, 53, 62, 70, 76, 76, 81, 89, 90, 98]
```

# Self Check 14.7

Why does only one of the two `while` loops at the end of the `merge` method do any work?

**Answer:** When the preceding `while` loop ends, the loop condition must be `false`, that is,

`iFirst >= first.length` or `iSecond >= second.length`
(De Morgan's Law).

- لماذا واحد فقط من اثنين في `while` حلقات في نهاية طريقة دمج merge القيام بأي عمل؟
- الجواب: عندما السابقة `while` تنتهي الحلقة، يجب أن يكون شرط حلقة `false`، وهذا هو،
- iFirst> = first.length أوiSecond> = second.length
- )القانون دي مورغان).

# Self Check 14.8

Manually run the merge sort algorithm on the array 8 7 6 5 4 3 2 1.

**Answer:**

First sort 8 7 6 5.
Recursively, first sort 8 7.
Recursively, first sort 8. It's sorted.
Sort 7. It's sorted.
Merge them: 7 8.
Do the same with 6 5 to get 5 6.
Merge them to 5 6 7 8.
Do the same with 4 3 2 1: Sort 4 3 by sorting 4 and 3 and merging them to 3 4.
Sort 2 1 by sorting 2 and 1 and merging them to 1 2.
Merge 3 4 and 1 2 to 1 2 3 4.
Finally, merge 5 6 7 8 and 1 2 3 4 to 1 2 3 4 5 6 7 8.

- تشغيل يدويا الخوارزمية دمج النوع على مجموعة ٨ ٧ ٦ ٥ ٤ ٣ ٢ ١.

- النوع الأول ٨ ٧ ٦ ٥.
- بشكل متكرر، فرز أول ٨ ٧.
- بشكل متكرر، وهو أول نوع ٨. انها مرتبة و.
- ترتيب ٧. انها مرتبة و.
- دمجها: ٧ ٨.
- تفعل الشيء نفسه مع ٦ ٥ للحصول على ٥ ٦.
- دمجها ل٥ ٦ ٧ ٨.
- تفعل الشيء نفسه مع ٤ ٣ ٢ ١: فرز ٤ ٣ عن طريق الفرز ٤ و ٣ ودمجها ل٣ ٤.
- ترتيب ٢ ١ عن طريق الفرز ٢ و ١ و دمجها إلى ١ ٢.
- دمج ٣ ٤ و ١ فبراير - ١ فبراير ٣ ٤.
- وأخيرا، دمج ٥ ٦ ٧ ٨ و ١ ٢ ٣ أبريل - ١ فبراير ٣ ٤ ٥ ٦ ٧ ٨

# Analyzing the Merge Sort Algorithm

تحليل دمج خوارزمية ترتيب

| *n* | Merge Sort (milliseconds) | Selection Sort (milliseconds) |
|---|---|---|
| 10,000 | 40 | 786 |
| 20,000 | 73 | 2,148 |
| 30,000 | 134 | 4,796 |
| 40,000 | 170 | 9,192 |
| 50,000 | 192 | 13,321 |
| 60,000 | 205 | 19,299 |

# Merge Sort Timing vs. Selection Sort

دمج ترتیب توقیت مقابل اختیار ترتیب

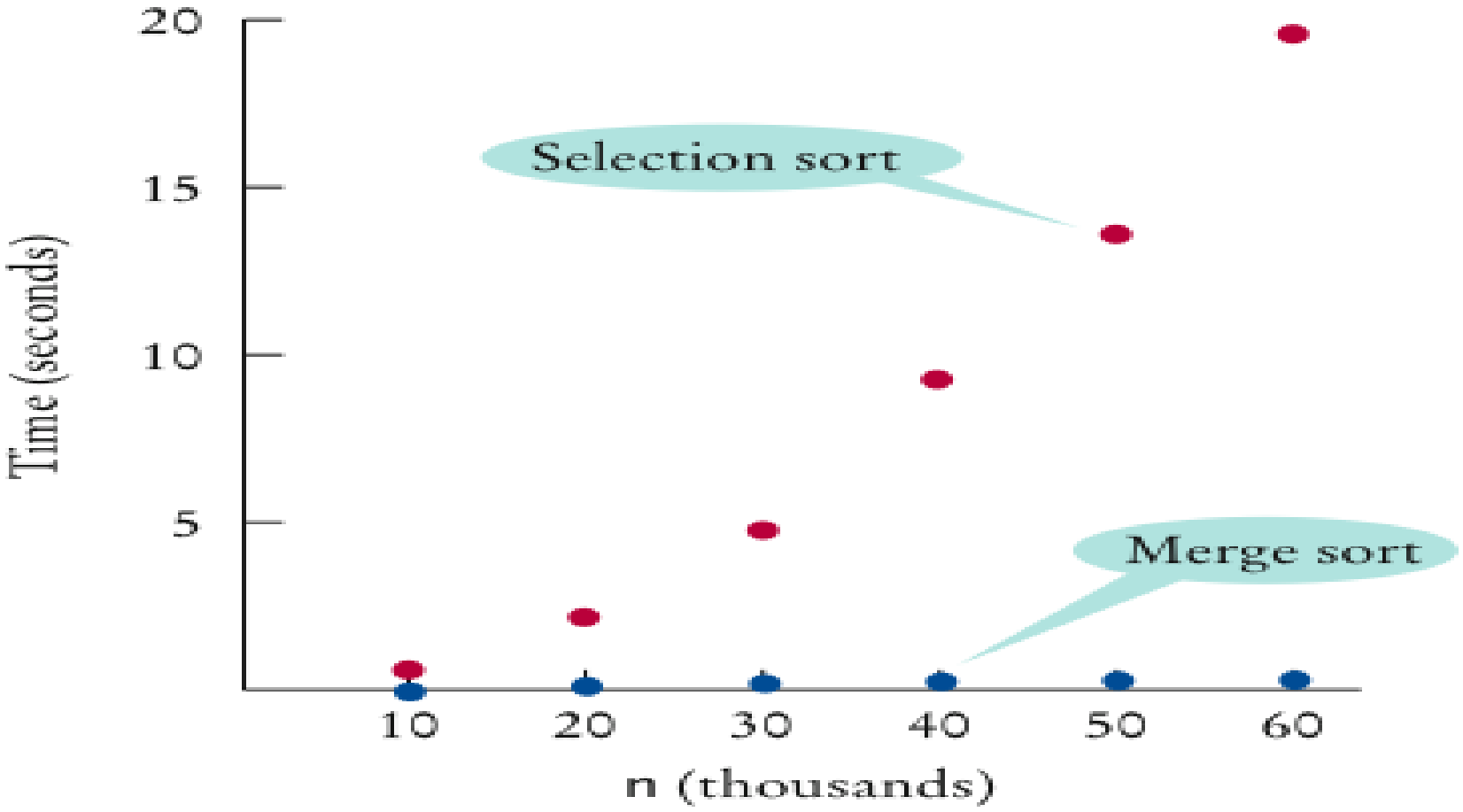



**Figure 2**
Merge Sort Timing versus Selection Sort

# Analyzing the Merge Sort Algorithm

- In an array of size *n*, count how many times an array element is visited
- Assume *n* is a power of 2: $n = 2^m$
- Calculate the number of visits to create the two sub-arrays and then merge the two sorted arrays
  - *3 visits to merge each element or 3n visits*
  - *2n visits to create the two sub-arrays*
  - *total of 5n visits*

- في مجموعة من حجم ن، عد كم مرة زار عنصر صفيف
- نفترض ن هي قوة ٢: N = M٢
- حساب عدد الزيارات لإنشاء صفائف الفرعية اثنين ثم دمج اثنين من صفائف فرزها
- ٣ زيارات لدمج كل عنصر أو مرة N٣
- مرة ٢ Nلإنشاء صفائف الفرعية اثنين
- مجموع الزيارات N٥

# Analyzing the Merge Sort Algorithm

# تحليل دمج خوارزمية ترتيب

- Let $T(n)$ denote the number of visits to sort an array of $n$ elements then
  - $T(n) = T(n/2) + T(n/2) + 5n$ or
  - $T(n) = 2T(n/2) + 5n$
- The visits for an array of size $n/2$ is:
  - $T(n/2) = 2T(n/4) + 5n/2$
  - So $T(n) = 2 \times 2T(n/4) + 5n + 5n$
- The visits for an array of size $n/4$ is:
  - $T(n/4) = 2T(n/8) + 5n/4$
  - So $T(n) = 2 \times 2 \times 2T(n/8) + 5n + 5n + 5n$

- السماح $T(n)$ للدلالة على عدد من الزيارات لفرز مجموعة من العناصر ن ثم
- زيارات لمجموعة من حجم ن / ٢ هو:
- زيارات لمجموعة من حجم ن / ٤ هو:

# Analyzing Merge Sort Algorithm

- Repeating the process *k* times:
  - $T(n) = 2^k T(n/2^k) + 5nk$
  - Since $n = 2^m$, when $k=m$: $T(n) = 2^m T(n/2^m) + 5nm$
  - $T(n) = nT(1) + 5nm$
  - $T(n) = n + 5n\log_2(n)$

- تكرار مرات عملية ك:

# Analyzing Merge Sort Algorithm

- To establish growth order
  - *Drop the lower-order term n*
  - *Drop the constant factor 5*
  - *Drop the base of the logarithm since all logarithms are related by a constant factor*
  - *We are left with n log(n)*
- Using big-Oh notation: Number of visits is O(*n*log(*n*))

# Merge Sort Vs Selection Sort

دمج ترتيب مقابل تحديد ترتيب

- Selection sort is an $O(n^2)$ algorithm
- Merge sort is an $O(n\log(n))$ algorithm
- The $n\log(n)$ function grows much more slowly than $n^2$

- اختيار النوع هو O (N2) خوارزمية
- دمج النوع هو $O(n\log(n))$ خوارزمية
- ينمو (ن) وظيفة $n\log(n)$ ببطء أكثر بكثير من N2

# Self Check 14.9

Given the timing data for the merge sort algorithm in the table at the beginning of this section, how long would it take to sort an array of 100,000 values?

**Answer:** Approximately 100,000 × log(100,000) / 50,000 × log(50,000) = 2 × 5 / 4.7 = 2.13 times the time required for 50,000 values. That's 2.13 × 97 milliseconds or approximately 207 milliseconds.

وبالنظر إلى البيانات توقيت لخوارزمية الفرز الدمج في الجدول في بداية هذا القسم، كم من الوقت سيستغرق لفرز مجموعة من ١٠٠،٠٠٠ القيم؟
الجواب: حوالي ١٠٠،٠٠٠ × سجل (١٠٠،٠٠٠) / ٥٠،٠٠٠ × سجل (٥٠،٠٠٠) = ٢ × ٥ / ٤.٧ = ٢.١٣
أضعاف الوقت اللازم ل٥٠،٠٠٠ القيم. هذا هو ٢.١٣ × ٩٧ مللي ثانية أو ما يقرب من ٢٠٧ ميلي ثانية.

# Self Check 14.10

If you double the size of an array, how much longer will the merge sort algorithm take to sort the new array?

**Answer:** $(2n \log(2n)/n \log(n)) = 2(1+ \log(2)/\log(n))$. For $n > 2$, that is a value < 3.

- إذا كنت مضاعفة حجم صفيف، كم من الوقت سوف خوارزمية الفرز دمج تأخذ لفرز مجموعة جديدة؟

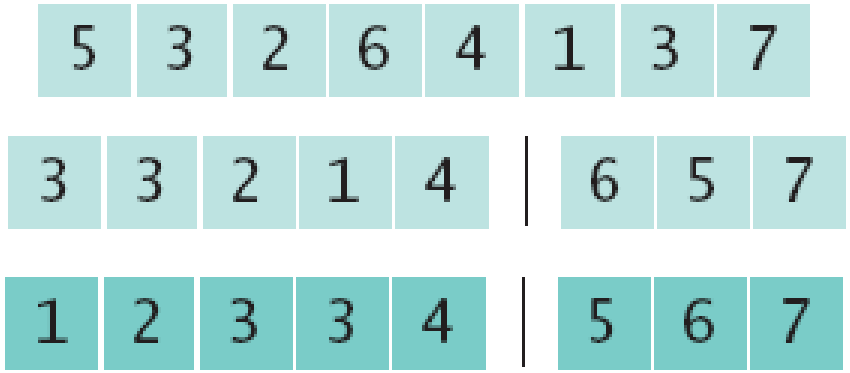
**Answer:** $(2n \log(2n)/n \log(n)) = 2(1+ \log(2)/\log(n))$. )
For $n > 2$, that is a value < 3.

# The Quicksort Algorithm

وفرز سريع خوارزمية

- Divide and conquer
  - *1. Partition the range*
  - *2. Sort each partition*

- فرق تسد
- تقسيم مجموعة
- فرز كل قسم

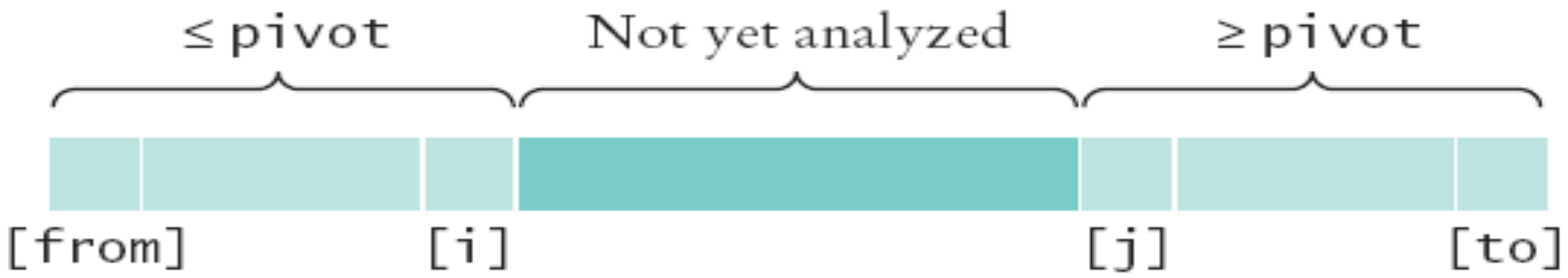


# The Quicksort Algorithm

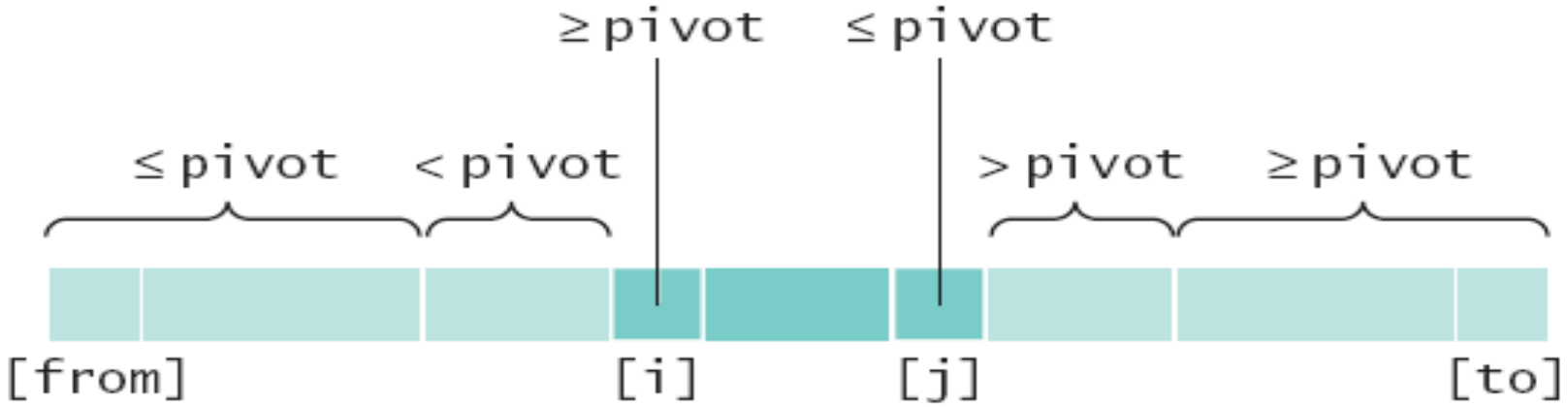
وفرز سريع خوارزمية

```
public void sort(int from, int to)
{
   if (from >= to)
   return; int p =
   partition(from, to);
   sort(from, p);
   sort(p + 1, to);
}
```

# The Quicksort Algorithm




Partitioning a Range




Extending the Partitions

# The Quicksort Algorithm

```
private int partition(int from, int to)
{
   int pivot = a[from];
   int i = from - 1;
   int j = to + 1;
   while (i < j)
   {
      i++;
      while (a[i] < pivot) i++;
      j--;
      while (a[j] > pivot) j--;
      if (i < j) swap(i, j);
   }
   return j;
}
```

# The First Programmer

Babbage's Difference Engine

# Searching

البحث

- **Linear search:** also called **sequential search**
- 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*
- A linear search locates a value in an array in O(*n*) steps

- البحث الخطي: وتسمى أيضا البحث متسلسل
- يدرس كافة القيم في صفيف حتى يجدها تطابق أو يصل إلى نهاية
- عدد الزيارات لبحث خطية من مجموعة من العناصر ن:
- متوسط زيارات البحث ن / ٢ العناصر
- الحد الأقصى مرة هو ن
- بحث خطي يحدد موقع قيمة في صفيف في O(*n*) خطوات

## ch14/linsearch/LinearSearcher.java

```
1  /**
2     A class for executing linear searches through an array.
3  */
4  public class LinearSearcher
5  {
6     private int[] a;
7
8     /**
9        Constructs the LinearSearcher.
10       @param anArray an array of integers
11    */
12    public LinearSearcher(int[] anArray)
13    {
14       a = anArray;
15    }
16
```

*Continued*

# ch14/linsearch/LinearSearcher.java (cont.)

```
17      /**
18         Finds a value in an array, using the linear search
19         algorithm.
20         @param v the value to search
21         @return the index at which the value occurs, or -1
22         if it does not occur in the array
23      */
24      public int search(int v)
25      {
26         for (int i = 0; i < a.length; i++)
27         {
28            if (a[i] == v)
29               return i;
30         }
31         return -1;
32      }
33   }
```

# ch14/linsearch/LinearSearchDemo.java

```
 1  import java.util.Arrays;
 2  import java.util.Scanner;
 3
 4  /**
 5     This program demonstrates the linear search algorithm.
 6  */
 7  public class LinearSearchDemo
 8  {
 9     public static void main(String[] args)
10     {
11        int[] a = ArrayUtil.randomIntArray(20, 100);
12        System.out.println(Arrays.toString(a));
13        LinearSearcher searcher = new
LinearSearcher(a);
14
15        Scanner in = new Scanner(System.in);
16
```

***Continued***

# ch14/linsearch/LinearSearchDemo.java (cont.)

```
17          boolean done = false;
18          while (!done)
19          {
20             System.out.print("Enter number to search for, -1 to quit:
");
21             int n = in.nextInt();
22             if (n == -1)
23                done = true;
24             else
25             {
26                int pos = searcher.search(n);
27                System.out.println("Found in position " + pos);
28             }
29          }
30       }
31    }
```

## Typical Program Run:

```
[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.

- لنفترض أنك بحاجة الى ان ننظر من خلال ١،٠٠٠،٠٠٠ سجلات للعثور على رقم الهاتف. عدد السجلات تتوقعون للبحث قبل العثور على العدد؟
- الإجابة: في المتوسط، وكنت جعل ٥٠٠،٠٠٠ المقارنات.

# 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.

- لماذا لا يمكنك استخدام "كل" حلقة عن `(int element : a)` في طريقة البحث؟
- الإجابة: أسلوب البحث بإرجاع المؤشر الذي يحدث في المباراة، وليس البيانات المخزنة في ذلك الموقع.

# Binary Search

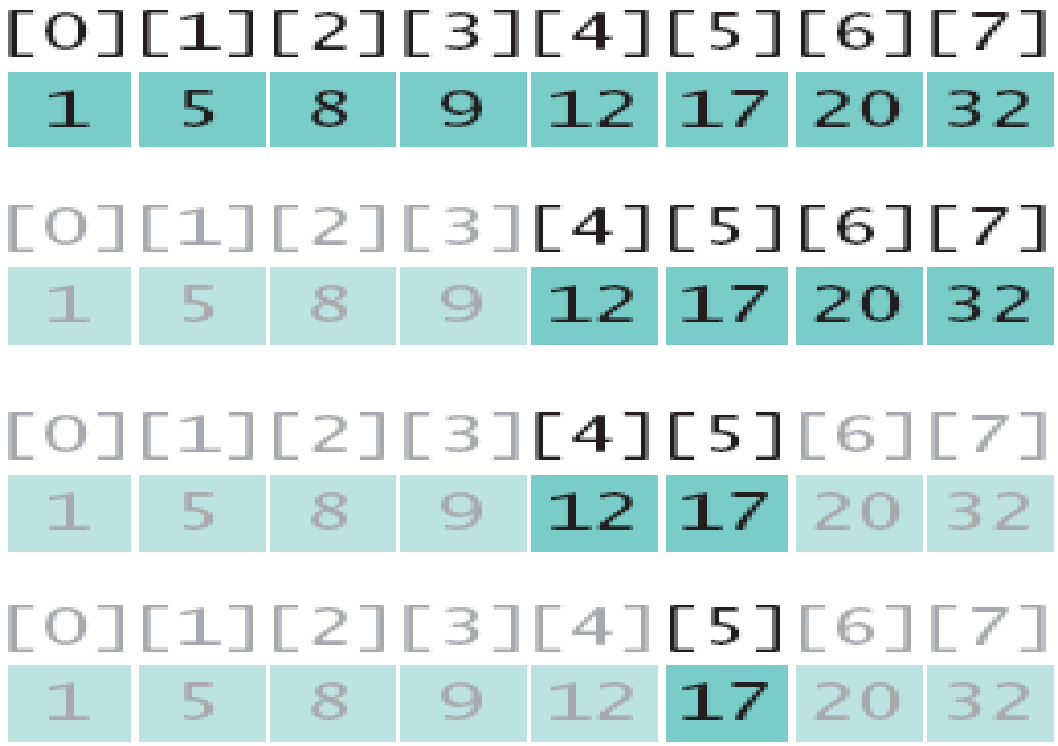


- Locates a value in a sorted array by
  - *Determining whether the value occurs in the first or second half*
  - *Then repeating the search in one of the halves*

- يقع قيمة في مجموعة وفرزها من قبل
- تحديد ما إذا كانت قيمة يحدث في النصف الأول أو الثاني
- ثم تكرار البحث في أحد نصفي

# Binary Search

- To search 15:




- 15 ≠ 17: We don't have a match

# ch14/binsearch/BinarySearcher.java

```
1  /**
2     A class for executing binary searches through an array.
3  */
4  public class BinarySearcher
5  {
6     private int[] a;
7
8     /**
9        Constructs a BinarySearcher.
10       @param anArray a sorted array of integers
11    */
12    public BinarySearcher(int[] anArray)
13    {
14       a = anArray;
15    }
16
```

*Continued*

# ch14/binsearch/BinarySearcher.java (cont.)

```
17      /**
18         Finds a value in a sorted array, using the binary
19         search algorithm.
20         @param v the value to search
21         @return the index at which the value occurs, or -1
22         if it does not occur in the array
23      */
24      public int search(int v)
25      {
26         int low = 0;
27         int high = a.length - 1;
28         while (low <= high)
29         {
30            int mid = (low + high) / 2;
31            int diff = a[mid] - v;
32
33            if (diff == 0) // a[mid] == v
34               return mid;
35            else if (diff < 0) // a[mid] < v
36               low = mid + 1;
37            else
38               high = mid - 1;
39         }
40         return -1;
41      }
42   }
```

# Binary Search

- Count the number of visits to search a sorted array of size $n$
  - *We visit one element (the middle element) then search either the left or right subarray*
  - *Thus: $T(n) = T(n/2) + 1$*
- If $n$ is $n/2$, then $T(n/2) = T(n/4) + 1$
- Substituting into the original equation: $T(n) = T(n/4) + 2$
- This generalizes to: $T(n) = T(n/2^k) + k$

- حساب عدد الزيارات للبحث عن مجموعة مرتبة من حجم ن
- نزور عنصر واحد (العنصر المتوسط) ثم ابحث إما إلى اليسار أو اليمين subarray
  - وبالتالي $T(n) = T(n/2) + 1$ :
- إذا كان $n$ is $n/2$, then $T(n/2) = T(n/4) + 1$
- استبدال في المعادلة الأصلية $T(n) = T(n/4) + 2$
- هذا يعمم ل $T(n) = T(n/2^k) + k$

# Binary Search

- Assume $n$ is a power of 2, $n = 2m$ where $m = \log_2(n)$
- Then: $T(n) = 1 + \log_2(n)$
- Binary search is an $O(\log(n))$ algorithm

# Searching a Sorted Array in a Program

# بالبحث في فرز مصفوفة في برنامج

- 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:*

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

- تحتوي الطبقة صالحة طريقة binarySearchثابت
- الأسلوب بإرجاع إما
- مؤشر العنصر، إذا تم العثور على العنصر
- أو كيلو - ١ حيث kهو موقف قبل الذي ينبغي إدراج العنصر:

# Self Check 14.13

Suppose you need to look through a sorted array with 1,000,000 elements to find a value. Using the binary search algorithm, how many records do you expect to search before finding the value?

**Answer:** You would search about 20. (The binary log of 1,024 is 10.)

- لنفترض أنك بحاجة الى ان ننظر من خلال مجموعة فرزها مع ١،٠٠٠،٠٠٠ العناصر لإيجاد قيمة. باستخدام خوارزمية البحث الثنائية، عدد السجلات تتوقعون للبحث قبل العثور على القيمة؟
- الجواب: عليك أن تبحث عن ٢٠. (سجل ثنائي من ١٠٢٤ هو ١٠.)

# 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.

- لماذا هو مفيد أن طريقة Arrays.binarySearch يشير إلى الموضع الذي ينبغي إدراج عنصر في عداد المفقودين؟
- الجواب: ثم أنت تعرف من أين أدخله بحيث يبقى فرز مجموعة، ويمكنك الاستمرار في استخدام البحث الثنائية.

# 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.

- لماذا Arrays.binarySearch العودة كيلو - ١ وليس كيلو تشير إلى أن القيمة غير موجودة، وينبغي أن تدرج قبل وضع ك؟
- الإجابة: خلاف ذلك، وكنت لا أعرف ما إذا كانت قيمة موجودة عند إرجاع الأسلوب ٠.

# Sorting Real Data

فرز بيانات حقيقية

- The `Arrays` class contains static `sort` methods
- To sort an array of integers:

```
int[] a = ... ;
Arrays.sort(a);
```

- That `sort` method uses the Quicksort algorithm (see Special Topic 14.3)

- فئة صالحة تحتوي على أساليب الفرز ثابتة
- لفرز مجموعة من الأعداد الصحيحة:
- يستخدم هذا الأسلوب نوع خوارزمية فرز سريع (انظر موضوع خاص ١٤.٣)

# Sorting Real Data

- `Arrays.sort` sorts objects of classes that implement `Comparable` interface:

```
public interface Comparable
{
   int compareTo(Object otherObject);
}
```

- The call `a.compareTo(b)` returns
  - *A negative number if `a` should come before `b`*
  - *0 if `a` and `b` are the same*
  - *A positive number otherwise*

- Arrays.sortيفرز الأجسام من الفئات التي تقوم بتنفيذ واجهة قابلة للمقارنة:

- استدعاء call `a.compareTo(b)` يعود
- عدد السلبي إذا كان aيجب أن تأتي قبل b
- ٠ إذاaو bهي نفسها
- الرقم الإيجابي على خلاف ذلك

# Sorting Real Data

- Several classes in Java (e.g. `String` and `Date`) implement `Comparable`
- You can implement `Comparable` interface for your own classes:

```
public class Coin implements Comparable
{
   ...
   public int compareTo(Object otherObject)
   {
      Coin other = (Coin)otherObject;
      if (value < other.value) return -1;
      if (value == other.value) return 0;
      return 1;
   }
   ...
}
```

# `compareTo` Method

- The implementation must define a total ordering relationship
  - *Antisymmetric*

    يجب أن تحدد تنفيذ علاقة يأمر الكلية

    *If* `a.compareTo(b)` $\leq 0$, *then* `b.compareTo(a)` $\geq 0$
  - *Reflexive*

    `a.compareTo(a)` $= 0$
  - *Transitive*

    *If* `a.compareTo(b)` $\leq 0$ *and* `b.compareTo(c)` $\leq 0$, *then* `a.compareTo(c)` $\leq 0$

# Sorting Real Data

فرز بيانات حقيقية

- Once your class implements `Comparable`, simply use the `Arrays.sort` method:

```
Coin[] coins = new Coin[n];
// Add coins
...
Arrays.sort(coins);
```

- If the objects are stored in an `ArrayList`, use

```
Collections.sort:
ArrayList<Coin> coins = new ArrayList<Coin>();
// Add coins
...
Collections.sort(coins);
```

- `Collections.sort` uses the merge sort algorithm

# Self Check 14.16

Why can't the `Arrays.sort` method sort an array of `Rectangle` objects?

**Answer:** The `Rectangle` class does not implement the `Comparable` interface.

- لماذا لا يمكن للطريقة Arrays.sort نوع صفيف من الكائنات `Rectangle`؟
- الإجابة: فئة `Rectangle` لا تنفذ واجهة قابلة `Comparable`.

# Self Check 14.17

What steps would you need to take to sort an array of `BankAccount` objects by increasing balance?

**Answer:** The `BankAccount` class needs to implement the `Comparable` interface. Its `compareTo` method must compare the bank balances.

- ما هي الخطوات التي تحتاج إلى اتخاذها لفرز مجموعة من الكائنات BankAccount عن طريق زيادة الرصيد؟
- الجواب: تحتاج الطبقة BankAccount لتنفيذ واجهة قابلة للمقارنة. منهجه compareTo يجب مقارنة الأرصدة لدى البنوك.