Chapter 23
Searching and Sorting

CS1: Java Programming
Colorado State University

Original slides by Daniel Liang
Modified slides by Chris Wilcox
Objectives

- To study and analyze time complexity of various sorting algorithms (§§23.2–23.7).
- To design, implement, and analyze insertion sort (§23.2).
- To design, implement, and analyze bubble sort (§23.3).
- To design, implement, and analyze merge sort (§23.4).
Searching Arrays

Searching is the process of looking for a specific element in an array; for example, discovering whether a certain score is included in a list of scores. Searching is a common task in computer programming. There are many algorithms and data structures devoted to searching. In this section, two commonly used approaches are discussed, linear search and binary search.

public class LinearSearch {
    /** The method for finding a key in the list */
    public static int linearSearch(int[] list, int key) {
        for (int i = 0; i < list.length; i++)
            if (key == list[i])
                return i;
        return -1;
    }
}
Linear Search

The linear search approach compares the key element, _key_, _sequentially_ with each element in the array _list_. The method continues to do so until the key matches an element in the list or the list is exhausted without a match being found. If a match is made, the linear search returns the index of the element in the array that matches the key. If no match is found, the search returns -1.
## Linear Search Animation

<table>
<thead>
<tr>
<th>Key</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
</tbody>
</table>

Key: 3

List: 6 4 1 9 7 3 2 8
Linear Search Animation

http://www.cs.armstrong.edu/liang/animation/web/LinearSearch.html
/** The method for finding a key in the list */
public static int linearSearch(int[] list, int key) {
    for (int i = 0; i < list.length; i++)
        if (key == list[i])
            return i;
    return -1;
}

Trace the method

int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
int i = linearSearch(list, 4); // returns 1
int j = linearSearch(list, -4); // returns -1
int k = linearSearch(list, -3); // returns 5
Binary Search

For binary search to work, the elements in the array must already be ordered. Without loss of generality, assume that the array is in ascending order.

e.g., 2 4 7 10 11 45 50 59 60 66 69 70 79

The binary search first compares the key with the element in the middle of the array.
Binary Search, cont.

Consider the following three cases:

- If the key is less than the middle element, you only need to search the key in the first half of the array.
- If the key is equal to the middle element, the search ends with a match.
- If the key is greater than the middle element, you only need to search the key in the second half of the array.
# Binary Search

<table>
<thead>
<tr>
<th>Key</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td><img src="" alt="List" /></td>
</tr>
<tr>
<td>8</td>
<td><img src="" alt="List" /></td>
</tr>
<tr>
<td>8</td>
<td><img src="" alt="List" /></td>
</tr>
</tbody>
</table>
Binary Search Animation

Binary Search, cont.

key is 11

key < 50

key > 7

key == 11
Binary Search, cont.

key is 54

key > 50

key < 66

key < 59
Binary Search, cont.

The binarySearch method returns the index of the element in the list that matches the search key if it is contained in the list. Otherwise, it returns -insertion point - 1.

The insertion point is the point at which the key would be inserted into the list.
Sorting Arrays

Sorting, like searching, is also a common task in computer programming. Many different algorithms have been developed for sorting. This section introduces a simple, intuitive sorting algorithms: selection sort.
Why study sorting?

Sorting is a classic subject in computer science. There are three reasons for studying sorting algorithms.

– First, sorting algorithms illustrate many creative approaches to problem solving and these approaches can be applied to solve other problems.

– Second, sorting algorithms are good for practicing fundamental programming techniques using selection statements, loops, methods, and arrays.

– Third, sorting algorithms are excellent examples to demonstrate algorithm performance.
What data to sort?

The data to be sorted might be integers, doubles, characters, or objects. §7.8, “Sorting Arrays,” presented selection sort and insertion sort for numeric values. The selection sort algorithm was extended to sort an array of objects in §11.5.7, “Example: Sorting an Array of Objects.” The Java API contains several overloaded sort methods for sorting primitive type values and objects in the java.util.Arrays and java.util.Collections class. For simplicity, this section assumes:

- data to be sorted are integers,
- data are sorted in ascending order, and
- data are stored in an array. The programs can be easily modified to sort other types of data, to sort in descending order, or to sort data in an ArrayList or a LinkedList.
Selection Sort

Selection sort finds the smallest number in the list and places it first. It then finds the smallest number remaining and places it second, and so on until the list contains only a single number.

Select 1 (the smallest) and swap it with 2 (the first) in the list.

Select 2 (the smallest) and swap it with 9 (the first) in the remaining list.

Select 4 (the smallest) and swap it with 5 (the first) in the remaining list.

Select 6 (the smallest) and swap it with 8 (the first) in the remaining list.

Select 8 (the smallest) and swap it with 9 (the first) in the remaining list.

Since there is only one element remaining in the list, the sort is completed.
Selection Sort Animation

http://www.cs.armstrong.edu/liang/animation/web/SelectionSort.html
From Idea to Solution

for (int i = 0; i < list.length; i++) {
    select the smallest element in list[i..listSize-1];
    swap the smallest with list[i], if necessary;
    // list[i] is in its correct position.
    // The next iteration apply on list[i+1..listSize-1]
}

... list[10]
...
for (int i = 0; i < listSize; i++) {
    swap the smallest with list[i], if necessary;
    // list[i] is in its correct position.
    // The next iteration apply on list[i..listSize-1]
}

Expand

double currentMin = list[i];

for (int j = i+1; j < list.length; j++) {
    if (currentMin > list[j]) {
        currentMin = list[j];
    }
}
}
for (int i = 0; i < listSize; i++) {

swap the smallest with list[i], if necessary;
// list[i] is in its correct position.
// The next iteration apply on list[i..listSize-1]
}

Expand

double currentMin = list[i];
int currentMinIndex = i;
for (int j = i; j < list.length; j++) {
    if (currentMin > list[j]) {
        currentMin = list[j];
        currentMinIndex = j;
    }
}

for (int i = 0; i < listSize; i++) {
    select the smallest element in list[i..listSize-1];
    // list[i] is in its correct position.
    // The next iteration apply on list[i..listSize-1]
}

if (currentMinIndex != i) {
    list[currentMinIndex] = list[i];
    list[i] = currentMin;
}
Wrap it in a Method

/** The method for sorting the numbers */

public static void selectionSort(double[] list) {
    for (int i = 0; i < list.length; i++) {
        // Find the minimum in the list[i..list.length-1]
        double currentMin = list[i];
        int currentMinIndex = i;
        for (int j = i + 1; j < list.length; j++) {
            if (currentMin > list[j]) {
                currentMin = list[j];
                currentMinIndex = j;
            }
        }
        // Swap list[i] with list[currentMinIndex] if necessary;
        if (currentMinIndex != i) {
            list[currentMinIndex] = list[i];
            list[i] = currentMin;
        }
    }
}

Invoke it

selectionSort(yourList)
The insertion sort algorithm sorts a list of values by repeatedly inserting an unsorted element into a sorted sublist until the whole list is sorted.
Insertion Sort Animation

http://www.cs.armstrong.edu/liang/animation/web/InsertionSort.html
Insertion Sort

int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted

2 9 5 4 8 1 6
2 5 9 4 8 1 6
2 4 5 8 9 1 6
1 2 4 5 6 8 9
How to Insert?

The insertion sort algorithm sorts a list of values by repeatedly inserting an unsorted element into a sorted sublist until the whole list is sorted.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>9</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

list

**Step 1:** Save 4 to a temporary variable `currentElement`

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

list

**Step 2:** Move `list[2]` to `list[3]`

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>9</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

list

**Step 3:** Move `list[1]` to `list[2]`

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

list

**Step 4:** Assign `currentElement` to `list[1]`
From Idea to Solution

for (int i = 1; i < list.length; i++) {
    insert list[i] into a sorted sublist list[0..i-1] so that list[0..i] is sorted
}

list[0]

list[0] list[1]

list[0] list[1] list[2]


From Idea to Solution

```java
for (int i = 1; i < list.length; i++) {
    insert list[i] into a sorted sublist list[0..i-1] so that list[0..i] is sorted
}
```

Expand

double currentElement = list[i];
int k;
for (k = i - 1; k >= 0 && list[k] > currentElement; k--) {
    list[k + 1] = list[k];
}
// Insert the current element into list[k + 1]
list[k + 1] = currentElement;
```

InsertSort  Run
Bubble Sort

(a) 1st pass
(b) 2nd pass
(c) 3rd pass
(d) 4th pass
(e) 5th pass

Bubble sort time: $O(n^2)$

$$(n - 1) + (n - 2) + \ldots + 2 + 1 = \frac{n^2}{2} - \frac{n}{2}$$
Bubble Sort Animation

http://www.cs.armstrong.edu/liang/animation/web/BubbleSort.html

Bubble Sort Animation by Y. Daniel Liang using JavaScript and Processing.js

Perform bubble sort for a list of 20 distinct integers from 1 to 20. Click the Step button to swap the first element in the remaining unsorted list with the smallest element in the remaining unsorted list. Click the Reset button to start over with a new random list.
Computational Complexity (Big O)

- $T(n) = O(1)$  // constant time
- $T(n) = O(\log n)$  // logarithmic
- $T(n) = O(n)$  // linear
- $T(n) = O(n\log n)$  // linearithmic
- $T(n) = O(n^2)$  // quadratic
- $T(n) = O(n^3)$  // cubic
Complexity Examples

Big-O Complexity Chart

http://bigocheatsheet.com/
# Complexity Examples

## Array Sorting Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time Complexity</th>
<th>Space Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Average</td>
</tr>
<tr>
<td>Quicksort</td>
<td>$\Omega(n \log(n))$</td>
<td>$\Theta(n \log(n))$</td>
</tr>
<tr>
<td>Mergesort</td>
<td>$\Omega(n \log(n))$</td>
<td>$\Theta(n \log(n))$</td>
</tr>
<tr>
<td>Timsort</td>
<td>$\Omega(n)$</td>
<td>$\Theta(n \log(n))$</td>
</tr>
<tr>
<td>Heapsort</td>
<td>$\Omega(n \log(n))$</td>
<td>$\Theta(n \log(n))$</td>
</tr>
<tr>
<td>Bubble Sort</td>
<td>$\Omega(n)$</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>$\Omega(n)$</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>$\Omega(n^2)$</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>Tree Sort</td>
<td>$\Omega(n \log(n))$</td>
<td>$\Theta(n \log(n))$</td>
</tr>
<tr>
<td>Shell Sort</td>
<td>$\Omega(n \log(n))$</td>
<td>$\Theta(n(\log(n))^2)$</td>
</tr>
<tr>
<td>Bucket Sort</td>
<td>$\Omega(n+k)$</td>
<td>$\Theta(n+k)$</td>
</tr>
<tr>
<td>Radix Sort</td>
<td>$\Omega(nk)$</td>
<td>$\Theta(nk)$</td>
</tr>
<tr>
<td>Counting Sort</td>
<td>$\Omega(n+k)$</td>
<td>$\Theta(n+k)$</td>
</tr>
<tr>
<td>Cubesort</td>
<td>$\Omega(n)$</td>
<td>$\Theta(n \log(n))$</td>
</tr>
</tbody>
</table>

http://bigocheatsheet.com/
Why does it matter?

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>10</th>
<th>20</th>
<th>50</th>
<th>100</th>
<th>1,000</th>
<th>10,000</th>
<th>100,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>O(1)</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
</tr>
<tr>
<td>O(log(n))</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
</tr>
<tr>
<td>O(n)</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
</tr>
<tr>
<td>O(n*log(n))</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
</tr>
<tr>
<td>O(n²)</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>2 s</td>
<td>3 m</td>
</tr>
<tr>
<td>O(n³)</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>20 s</td>
<td>6 h</td>
<td>232 d</td>
</tr>
<tr>
<td>O(2^n)</td>
<td>&lt;1 s</td>
<td>&lt;1 s</td>
<td>260 d</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>O(n!)</td>
<td>&lt;1 s</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>O(n^n)</td>
<td>3 m</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>