Chapter 23
Searching and Sorting

CS1: Java Programming
Colorado State University

Original slides by Daniel Liang
Modified slides by Chris Wilcox

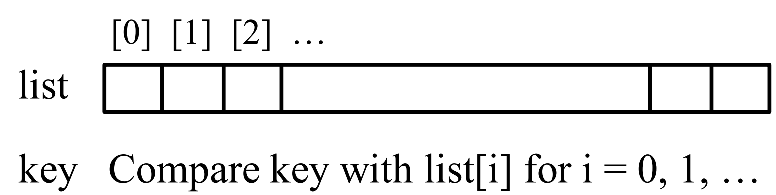
Objectives

- 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.

animation

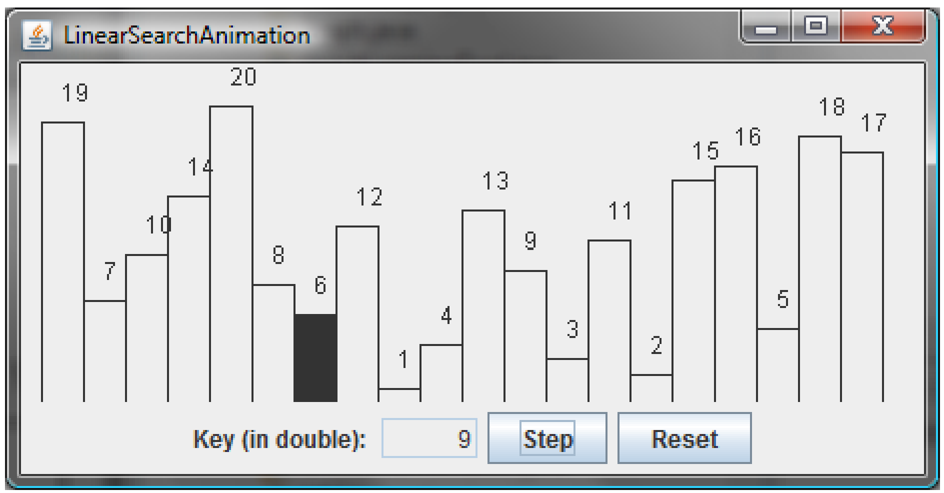
Linear Search Animation

Key	List							
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8

animation

Linear Search Animation

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



From Idea to Solution

```
/** 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.

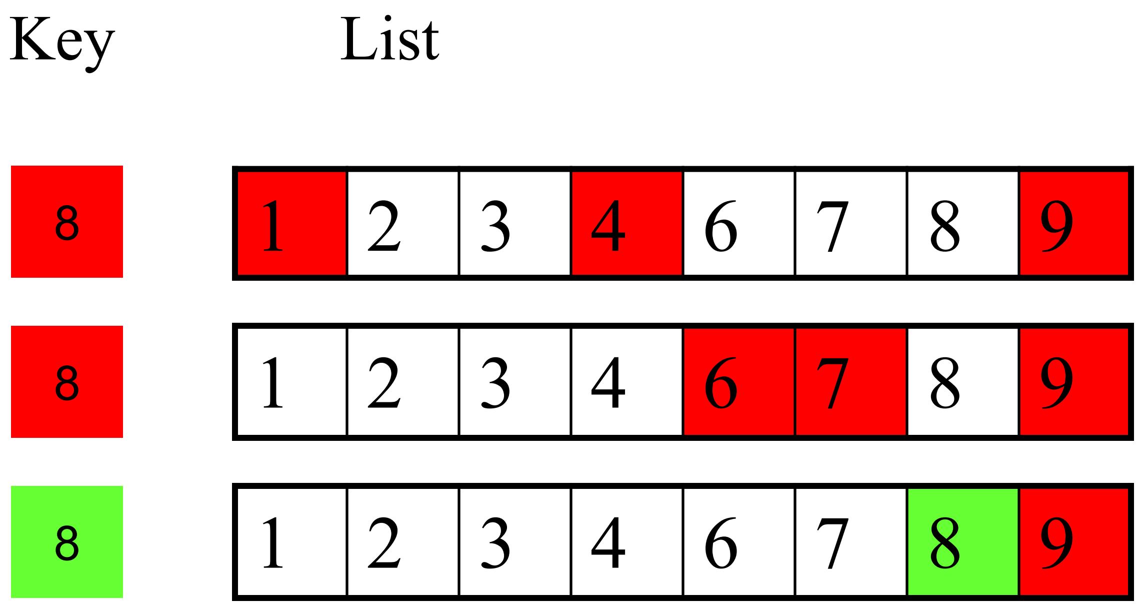
Binary Search, cont.

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.

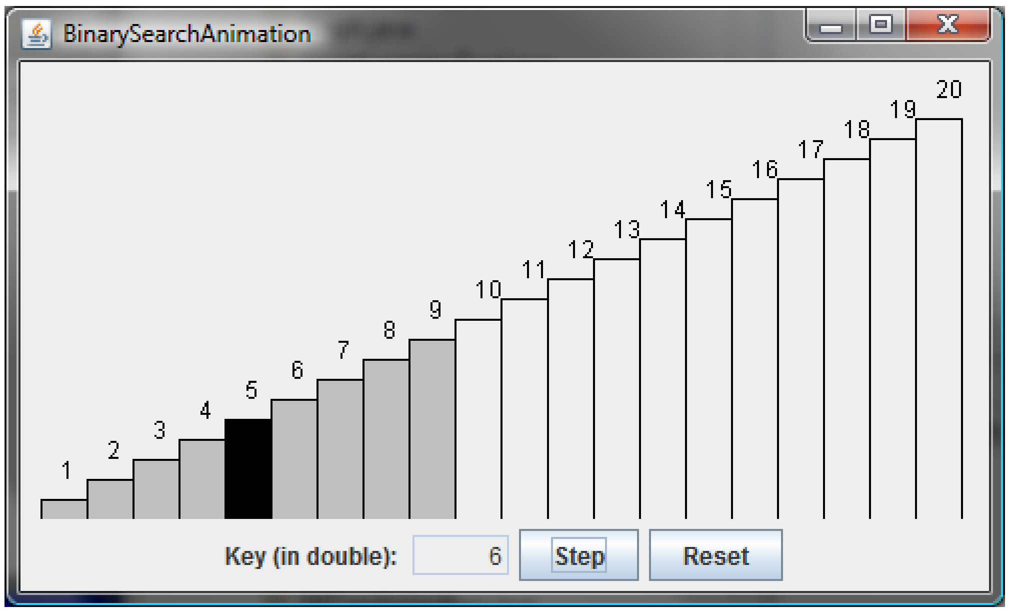
animation

Binary Search

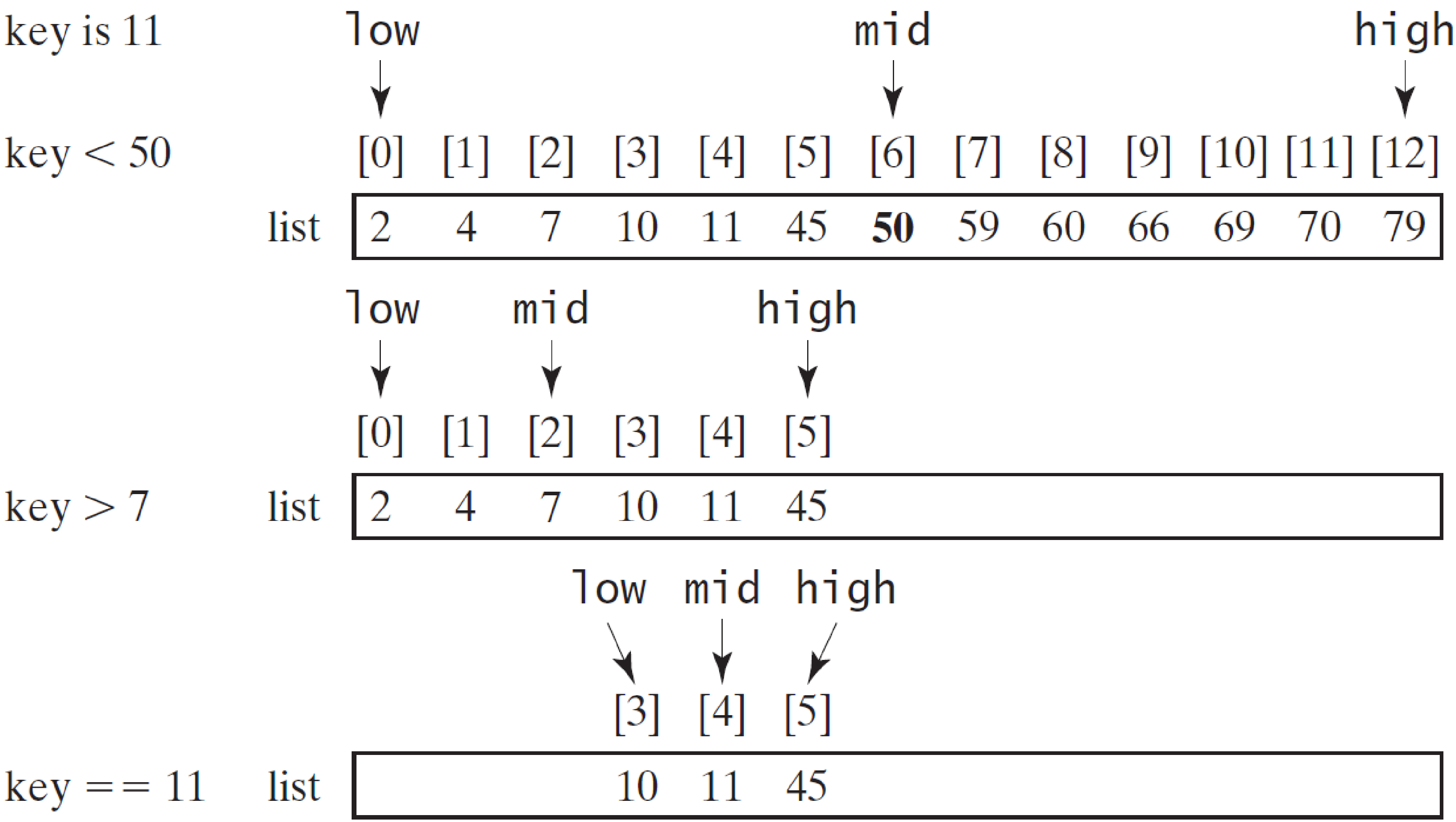


Binary Search Animation

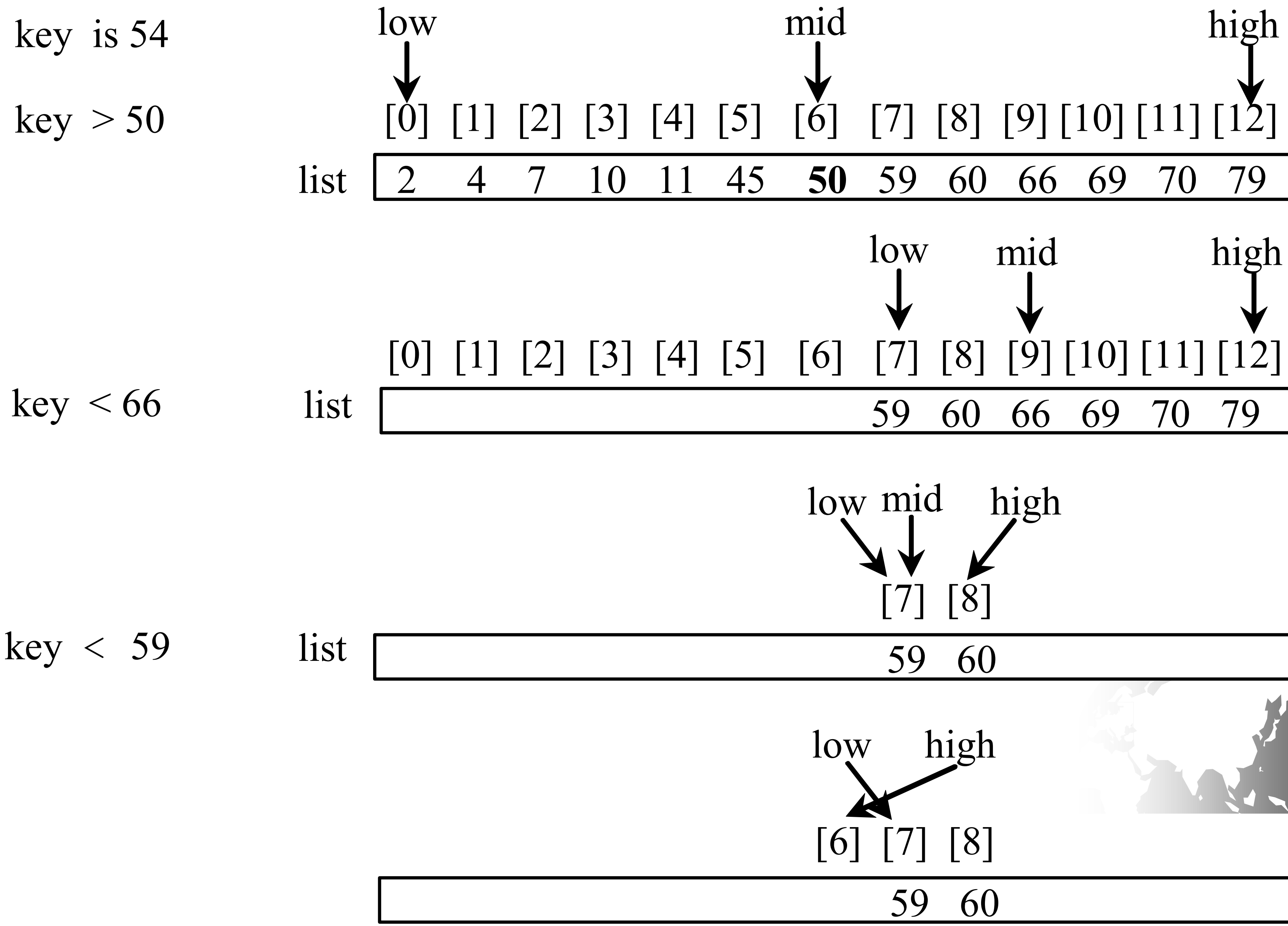
http://www.cs.armstrong.edu/liang/animation/web/BinarySearch.html



Binary Search, cont.



Binary Search, cont.



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.

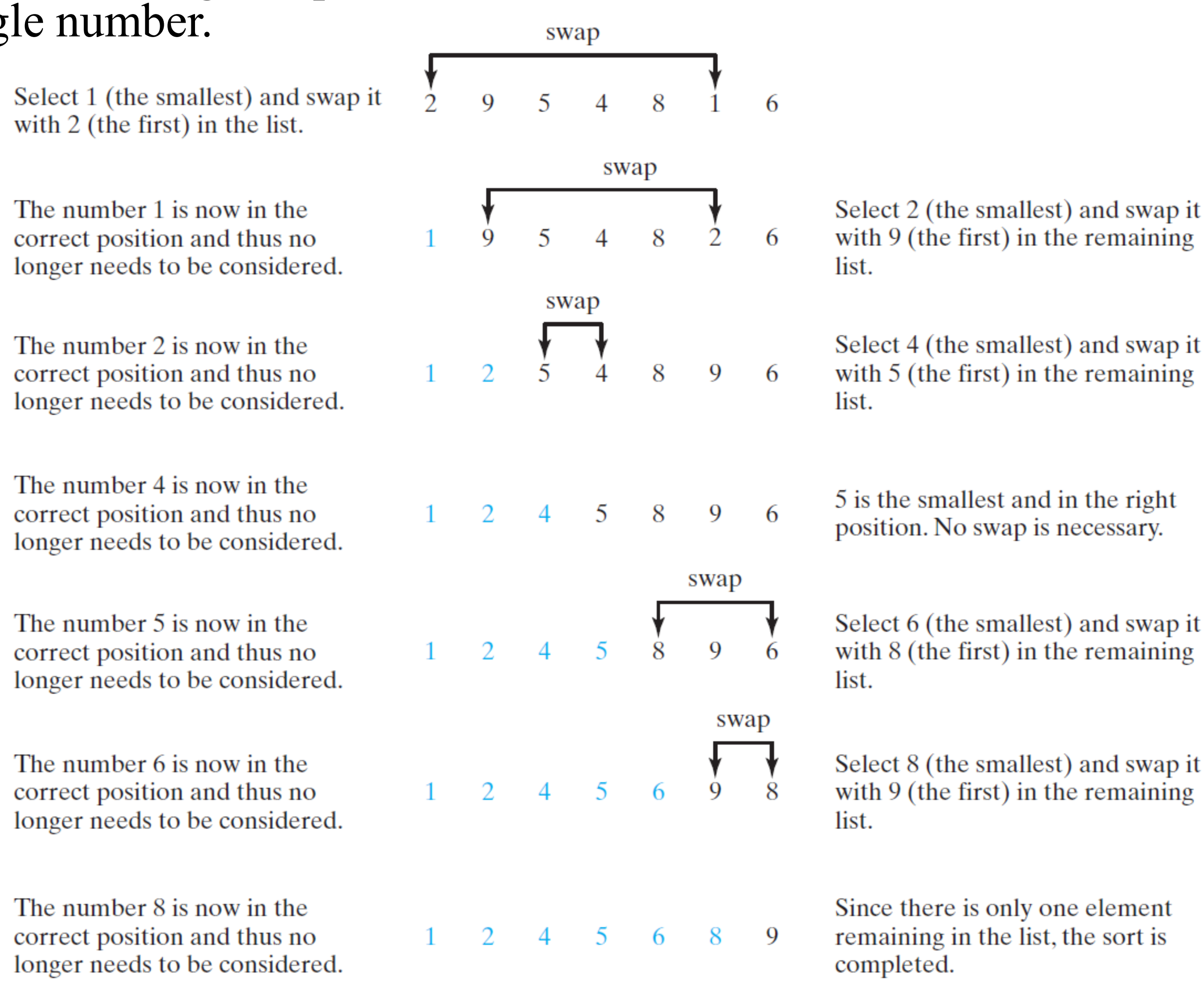
What data to sort?

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

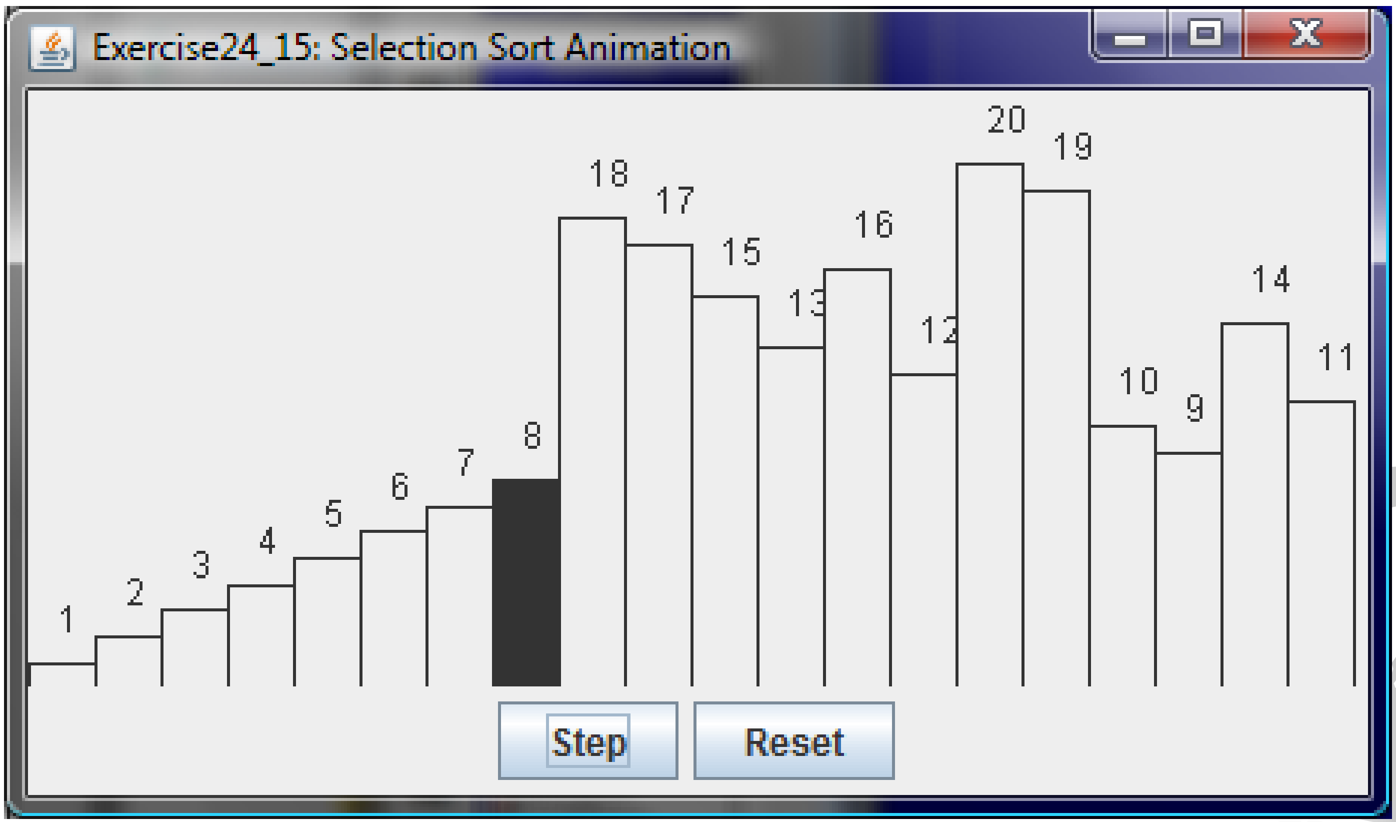
Selection sort finds the smallest number in the list and places it first. It then finds the smallest number remaining and places it second, and so on until the list contains only a single number.



animation

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[0] list[1] list[2] list[3] ...           list[10]

list[0] list[1] list[2] list[3] ...           list[10]

list[0] list[1] list[2] list[3] ...           list[10]

list[0] list[1] list[2] list[3] ...           list[10]

list[0] list[1] list[2] list[3] ...           list[10]

                                ...

list[0] list[1] list[2] list[3] ...           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]
}
```

Expand

```
  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)

Insertion Sort

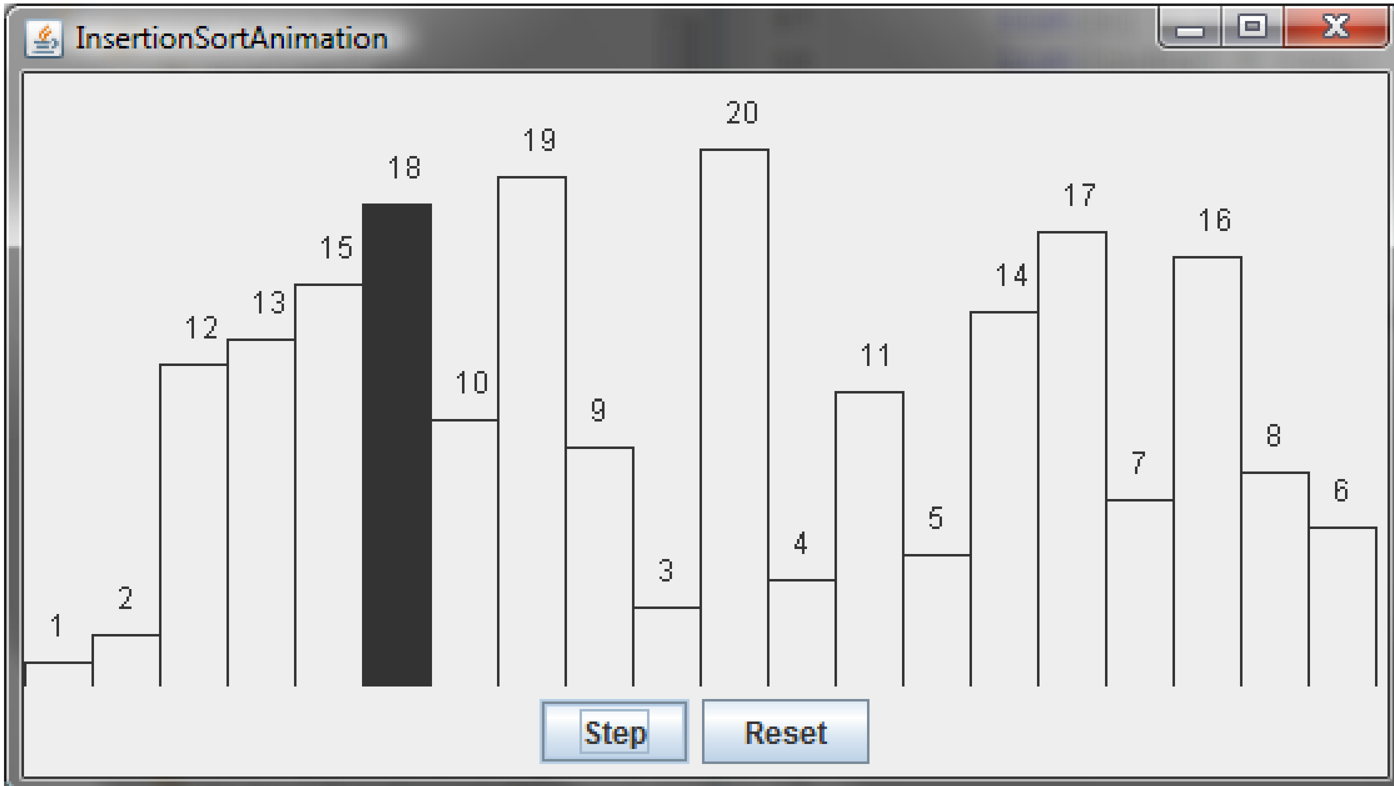
int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted

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.

Step 1: Initially, the sorted sublist contains the first element in the list. Insert 9 into the sublist.	2 9 5 4 8 1 6
Step2: The sorted sublist is {2, 9}. Insert 5 into the sublist.	2 9 → 5 4 8 1 6
Step 3: The sorted sublist is {2, 5, 9}. Insert 4 into the sublist.	2 5 → 9 → 4 8 1 6
Step 4: The sorted sublist is {2, 4, 5, 9}. Insert 8 into the sublist.	2 4 5 9 → 8 1 6
Step 5: The sorted sublist is {2, 4, 5, 8, 9}. Insert 1 into the sublist.	2 → 4 → 5 → 8 → 9 → 1 6
Step 6: The sorted sublist is {1, 2, 4, 5, 8, 9}. Insert 6 into the sublist.	1 2 4 5 8 → 9 → 6
Step 7: The entire list is now sorted.	1 2 4 5 6 8 9

Insertion Sort Animation

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



animation

Insertion Sort

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

2	9	5	4	8	1	6

2	9	5	4	8	1	6

2	5	9	4	8	1	6

2	4	5	9	8	1	6

2	4	5	8	9	1	6

1	2	4	5	8	9	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.

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	
list	2	5	9	4				Step 1: Save 4 to a temporary variable currentElement
list	2	5		9				Step 2: Move list[2] to list[3]
list	2		5	9				Step 3: Move list[1] to list[2]
list	2	4	5	9				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]

list[0] list[1] list[2] list[3]

list[0] list[1] list[2] list[3] ...
```

From Idea to Solution

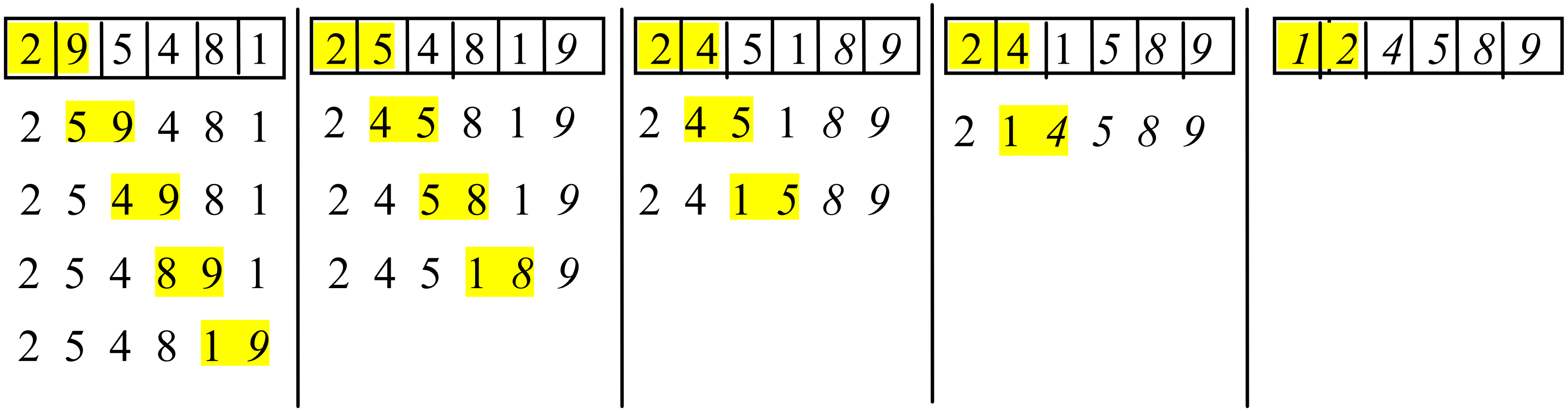
```
for (int i = 1; i < list.length; i++) {
  insert list[i] into a sorted sublist list[0..i-1] so that
  list[0..i] is sorted
}
```

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

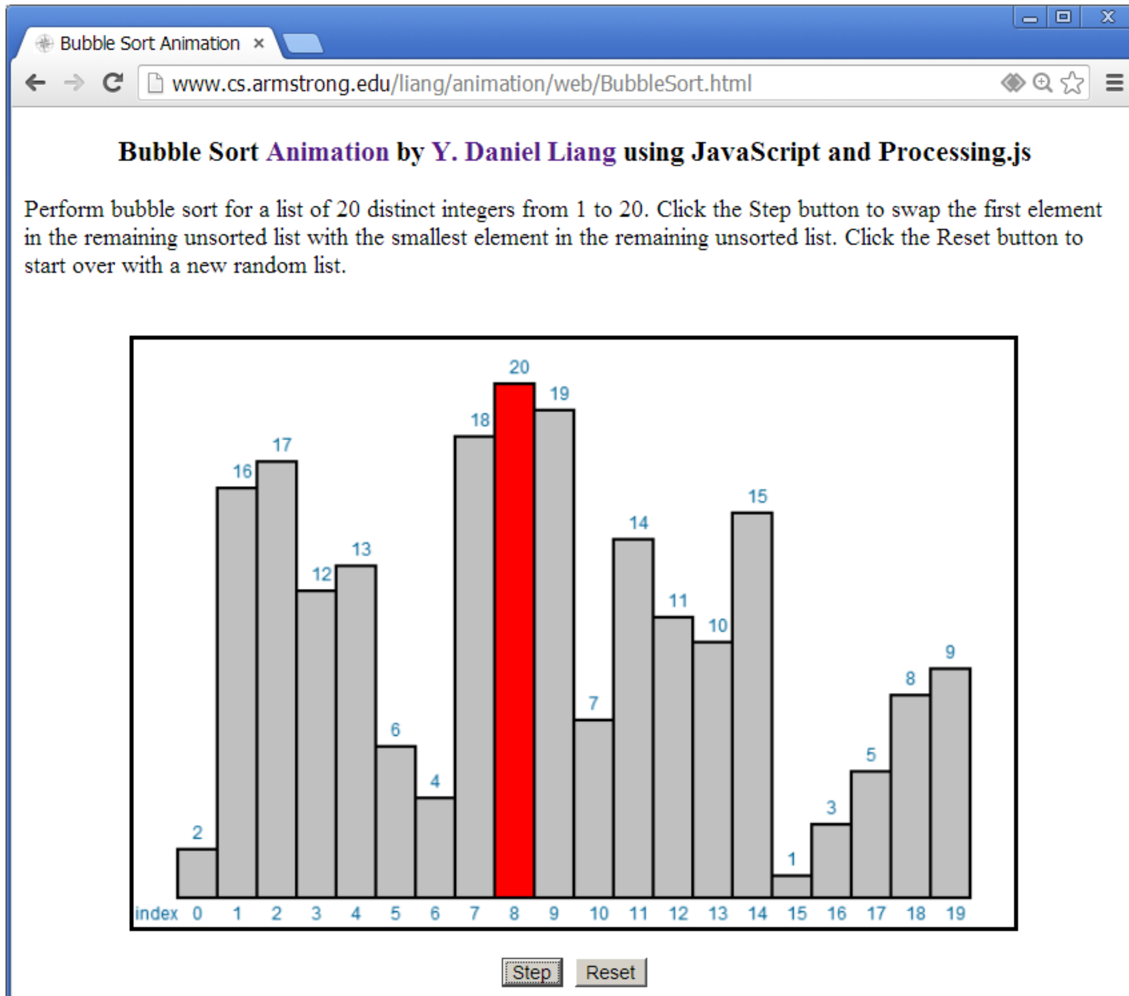
Bubble sort time: $O(n^2)$

$$(n-1)+(n-2)+...+2+1=\frac{n^2}{2}-\frac{n}{2}$$

BubbleSort Run

Bubble Sort Animation

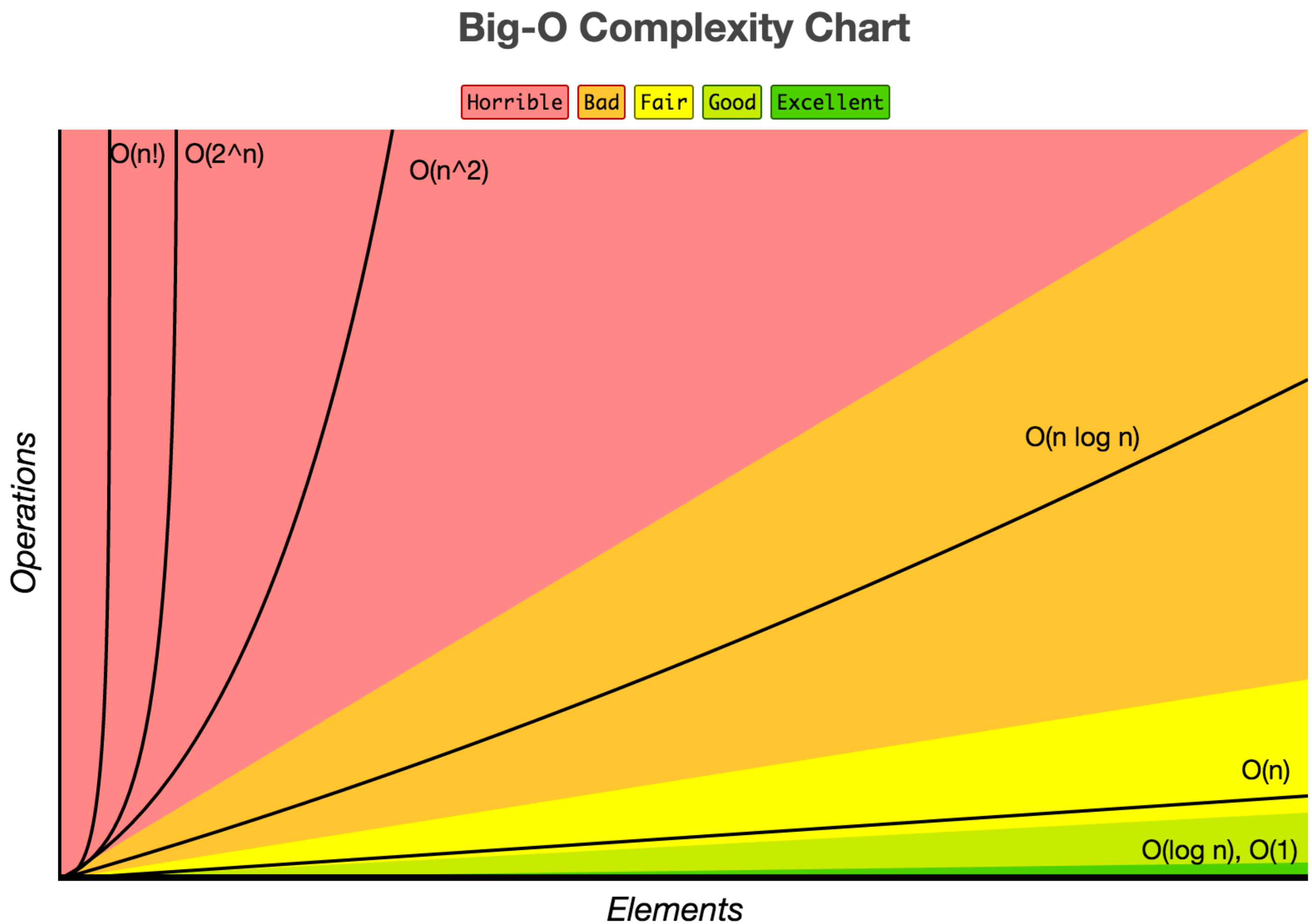
http://www.cs.armstrong.edu/liang/animation/web/BubbleSort.html



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



http://bigocheatsheet.com/

Complexity Examples

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

http://bigocheatsheet.com/

Why does it matter?

Algorithm	10	20	50	100	1,000	10,000	100,000
O(1)	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s
O(log(n))	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s
O(n)	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s
O(n*log(n))	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s
$O(n^2)$	<1 s	<1 s	<1 s	<1 s	<1 s	2 s	3 m
$O(n^3)$	<1 s	<1 s	<1 s	<1 s	20 s	6 h	232 d
$O(2^n)$	<1 s	<1 s	260 d	∞	∞	∞	∞
O(n!)	<1 s	∞	∞	∞	∞	∞	∞
$O(n^n)$	3 m	∞	∞	∞	∞	∞	∞