Structural Programming and Data Structures

Winter 2000

CMPUT 102: Sorting

Dr. Osmar R. Zaïane

University of Alberta

Course Content

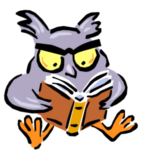
• Introduction	• Vectors
• Objects	• Testing/Debugging
• Methods	• Arrays
• Tracing Programs	• Searching
• Object State	• Files I/O
• Sharing resources	• **Sorting**
• Selection	• Inheritance
• Repetition	• Recursion

Objectives of Lecture 23

Sorting

- Introduce the problem of sorting collections;
- Learn how to sort using a bubble sort algorithm;
- Learn how to sort with the selection algorithm.

Outline of Lecture 23



- The sorting problem
- Simple methods like bubble sort
- Selection sort example
- Selection sort code
- Complexity of selection sort

The Sort Problem

- Given a container, with elements that can be compared, put it in increasing or decreasing order.

0	1	2	3	4	5	6	7	8	9
25	50	10	95	75	30	70	55	60	80

0	1	2	3	4	5	6	7	8	9
10	25	30	50	55	60	70	75	80	95

Sorting Problem (con't)

- Given a container of *n* elements A[*0..n-1*] such that any elements x and y in the container A can be compared directly, either x<y, or x=y, or x>y.
- We want to permute the elements of A so that at the end $A[0] \le A[1] \le \dots \le A[n-1]$ (monotone non-decreasing), or $A[0] \ge A[1] \ge \dots \ge A[n-1]$ (monotone decreasing)

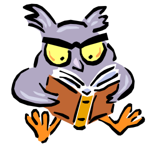
The Order of Things

- Numbers
  - -99 < -34< -6< 0 < 1 < 9 < 23 < 999
- Characters
  - A < B < C < D < E < F < …< X < Y < Z
  - a < b < c < d < e < f < …< x < y < z
  - a < z < A < Z
- Strings
  - "Abacus" < "Alpha" < "Hello" < "Memorization" < "Memorize" < "Memory" < "Zebra"

Sorting

- There is often a need to put data in order.
- Sorting is among the most basic and universal of computational problems.
- There are hundreds of algorithms and variations on algorithms.
- Variety of sorting methods: internal vs. external, sorting in place vs. sorting with auxiliary structures, etc.

Outline of Lecture 23



- The sorting problem
- Simple methods like bubble sort
- Selection sort example
- Selection sort code
- Complexity of selection sort

© Dr. Osmar R. Zaïane, 2000

One simple sorting method

Given a list: 35 | 18 | 22 | 97 | 61 | 10

Iterate over the collection and permute neighbours if necessary repeat iteration until no permutation possible.

18	35	22	97	61	10
18	22	35	97	61	10
18	22	35	97	61	10
18	22	35	61	97	10

18	22	35	61	10	97
…					
18	22	35	10	61	97
…					
10	18	22	35	61	97
…					
10	18	22	35	61	97

…

© Dr. Osmar R. Zaïane, 2000

The Bubble Sort

Given a list: 35 | 18 | 22 | 97 | 61 | 10

First pass of (compare and exchange) will send the largest to the last position.

35	18	22	97	61	10
18	35	22	97	61	10
18	22	35	97	61	10
18	22	35	97	61	10
18	22	35	61	97	10
18	22	35	61	10	97

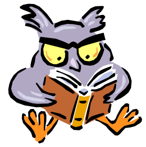
18	22	35	61	10	97
…					
18	22	35	10	61	97
18	22	35	10	61	97
…					
18	22	10	35	61	97
…					
10	18	22	35	61	97

© Dr. Osmar R. Zaïane, 2000


```
public static void bubble_sort( int data[] ) {
// Sort the given Array with selection sort method
//(Ascending order)

   int current, last;

   for ( last = data.length-1; last >=1; last--)
      for ( current = 0; current < last; current++ )
         if ( data[current] > data[current+1] )
            this.exchange( data, current, current+1 );
  }
```

© Dr. Osmar R. Zaïane, 2000

Outline of Lecture 23



- The sorting problem
- Simple methods like bubble sort
- Selection sort example
- Selection sort code
- Complexity of selection sort

Selection Sort

- Look for the smallest element and exchange it with the element whose index is 0.

0	1	2	3	4	5	6	7	8	9
25	50	10	95	75	30	70	55	60	80

0	1	2	3	4	5	6	7	8	9
10	50	25	95	75	30	70	55	60	80

Selection Sort (con't)

- Look for the smallest element whose index is greater than or equal to 1 and exchange it with the element whose index is 1.

0	1	2	3	4	5	6	7	8	9
10	50	25	95	75	30	70	55	60	80

0	1	2	3	4	5	6	7	8	9
10	25	50	95	75	30	70	55	60	80

Selection Sort (con't)

- Look for the smallest element whose index is greater than or equal to 2 and exchange it with the element whose index is 2.

0	1	2	3	4	5	6	7	8	9
10	25	50	95	75	30	70	55	60	80

0	1	2	3	4	5	6	7	8	9
10	25	30	95	75	50	70	55	60	80

Selection Sort (con't)

- Look for the smallest element whose index is greater than or equal to k and exchange it with the element whose index is k (for k = 3, 4, …, n-1)

0	1	2	3	4	5	6	7	8	9
10	25	30	95	75	50	70	55	60	80

0	1	2	3	4	5	6	7	8	9
10	25	30	50	75	95	70	55	60	80

Selection Sort (con't)

0	1	2	3	4	5	6	7	8	9
10	25	30	50	75	95	70	55	60	80

0	1	2	3	4	5	6	7	8	9
10	25	30	50	55	95	70	75	60	80

0	1	2	3	4	5	6	7	8	9
10	25	30	50	55	60	70	75	95	80

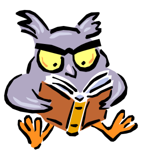
Selection Sort (con't)

0	1	2	3	4	5	6	7	8	9
10	25	30	50	55	60	70	75	95	80

0	1	2	3	4	5	6	7	8	9
10	25	30	50	55	60	70	75	95	80

0	1	2	3	4	5	6	7	8	9
10	25	30	50	55	60	70	75	80	95

Outline of Lecture 23



- The sorting problem
- Simple methods like bubble sort
- Selection sort example
- Selection sort code
- Complexity of selection sort

Selection Sort Algorithm

INPUT : data: an array of int
OUTPUT: data: sorted in ascending order

Method:

```
for ( first = 1; first < length - 1; first ++) {
    find Smallest such that data[Smallest] is the
    smallest between data[first] and data[length-1];
    permute Data[first] and Data[Smallest];
}
```


Selection Sort Code

```
private void selectionSort(int anArray[]) {
// Sort the given Array with selection sort method (Ascending order)

   int          index;
   int          smallIndex;

   for (index = 0; index < anArray.length - 1; index++) {
        smallIndex = this.getSmallest(anArray, index);
        this.exchange(anArray, index, smallIndex);
   }
}
```

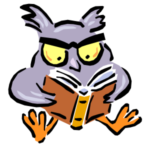

Code for method: exchange

```
private void exchange(int anArray[], int i, int j) {
// Exchange the elements of the array with
// the given two indexes.

   int          temp;

   temp = anArray[i];
   anArray[i] = anArray[j];
   anArray[j] = temp;
}
```


Code for method: getSmallest

```
private int getSmallest(int anArray[], int start) {
// Return the index of the smallest element
// of the given array whose index is greater
// than or equal to the given start index.
   int          smallestIndex;
   int          index;

   smallestIndex = start;
   for (index = start + 1; index < anArray.length; index++)
        if (anArray[index] < anArray[smallestIndex])
             smallestIndex = index;
   return smallestIndex;
}
```

Outline of Lecture 23



- The sorting problem
- Simple methods like bubble sort
- Selection sort example
- Selection sort code
- Complexity of selection sort

Complexity of Selection Sort

- How many comparison operations are required for a selection sort of an *n*-element container?
- The sort method executes **getSmallest** for the indexes: *0*, *1*, … *n-2*.
- Each time **getSmallest** is executed for an index, it does: (*n* - index) comparisons.
- The total number of comparisons is:

$$(n-0) + (n-1) + \ldots + (n-(n-2)) = (1 + 2 + \ldots + n) - 1 =$$

$$\frac{n(n+1)}{2} - 1 \approx n^2 \text{ for large } n.$$

$O(n^2)$ ➔ Quadratic time complexity