ENGINEERING PROGRAMMING I

Chapter 09

2017

Topics

- 9.1 Introduction to Search Algorithms
- 9.2 Searching an Array of Objects
- 9.3 Introduction to Sorting Algorithms
- 9.4 Sorting an Array of Objects
- 9.5 Sorting and Searching Vectors
- 9.6 Introduction to Analysis of Algorithms

Introduction to Search Algorithms

- Search: locate an item in a list (array, vector, etc.) of information
- Two algorithms (methods) considered here:
  - Linear search
  - Binary search

Linear Search Algorithm

```
Set found to false
  Set position to –1
  Set index to 0
While index < number of elts and found is false
    If list [index] is equal to search value
                    found = true
        position = index
    End If
    Add 1 to index
End While
Return position
```

Linear Search Example

- Array **numlist** contains

- Searching for the the value **11**, linear search examines **17**, **23**, **5**, and **11**
- Searching for the the value **7**, linear search examines **17**, **23**, **5**, **11**, **2**, **29**, and **3**

Linear Search Tradeoffs

- Benefits
  - Easy algorithm to understand
  - Array can be in any order
- Disadvantage
  - Inefficient (slow): for array of N elements, examines N/2 elements on average for value that is found in the array, N elements for value that is not in the array

Binary Search Algorithm

1. Divide a sorted array into three sections:
   - middle element
   - elements on one side of the middle element
   - elements on the other side of the middle element

2. If the middle element is the correct value, done. Otherwise, go to step 1, using only the half of the array that may contain the correct value.

3. Continue steps 1 and 2 until either the value is found or there are no more elements to examine.

Binary Search Example

- Array **numlist2** contains

- Searching for the the value **11**, binary search examines **11** and stops
- Searching for the the value **7**, binary search examines **11**, **3**, **5**, and stops

Binary Search Tradeoffs

- Benefit
  - Much more efficient than linear search
    (For array of N elements, performs at most $log_2 N$ comparisons)
- Disadvantage
  - Requires that array elements be sorted

Introduction to Sorting Algorithms

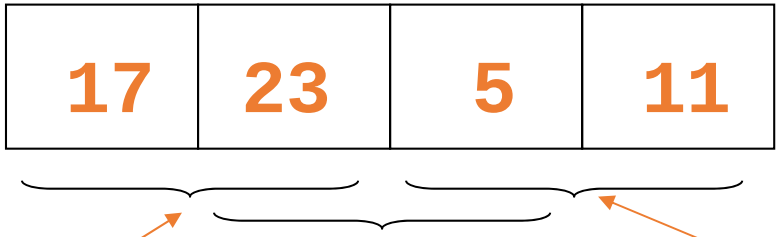
- Sort: arrange values into an order
  - Alphabetical
  - Ascending numeric
  - Descending numeric
- Two algorithms considered here
  - Bubble sort
  - Selection sort

Bubble Sort Algorithm

1. Compare 1st two elements and exchange them if they are out of order.
2. Move down one element and compare 2nd and 3rd elements. Exchange if necessary. Continue until end of array.
3. Pass through array again, repeating process and exchanging as necessary.
4. Repeat until a pass is made with no exchanges.

Bubble Sort Example

- Array **numlist3** contains



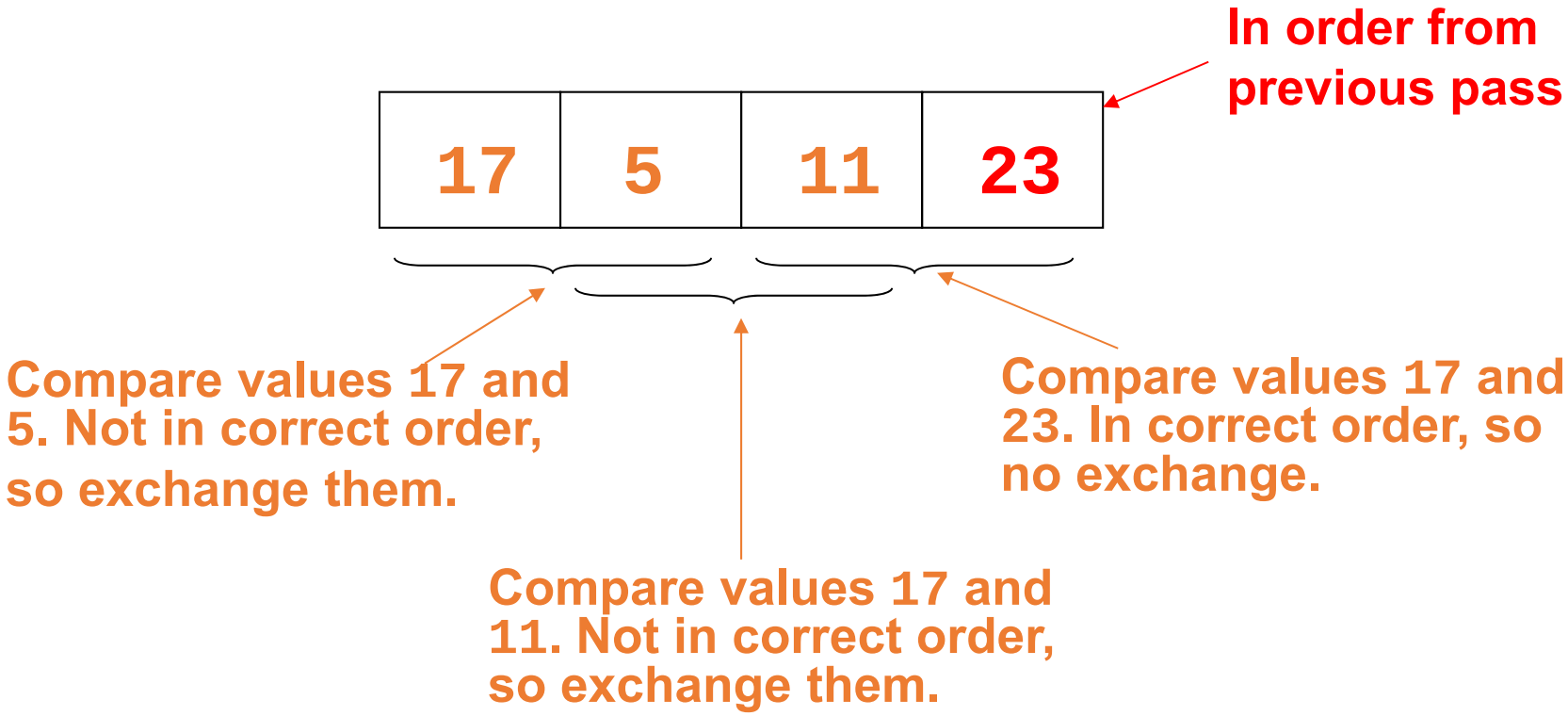
Compare values 17 and 23. In correct order, so no exchange.

Compare values 23 and 5. Not in correct order, so exchange them.

Compare values 23 and 11. Not in correct order, so exchange them.

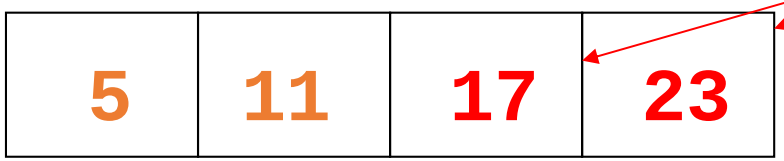
Bubble Sort Example (continued)

- After first pass, array **numlist3** contains



Bubble Sort Example (continued)

- After second pass, array **numlist3** contains



In order from previous passes

Compare values 5 and 11. In correct order, so no exchange.

Compare values 11 and 17. In correct order, so no exchange.

Compare values 17 and 23. In correct order, so no exchange.

No exchanges, so array is in order

Bubble Sort Tradeoffs

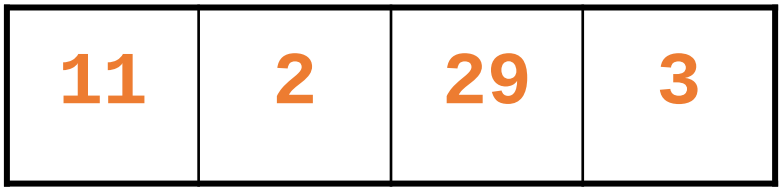
- Benefit
  - Easy to understand and implement
- Disadvantage
  - Inefficiency makes it slow for large arrays

Selection Sort Algorithm

1. Locate smallest element in array and exchange it with element in position 0.
2. Locate next smallest element in array and exchange it with element in position 1.
3. Continue until all elements are in order.

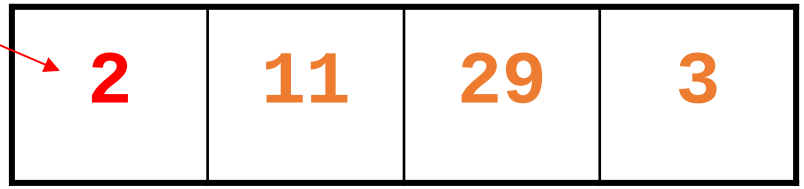
Selection Sort Example

Array **numlist** contains



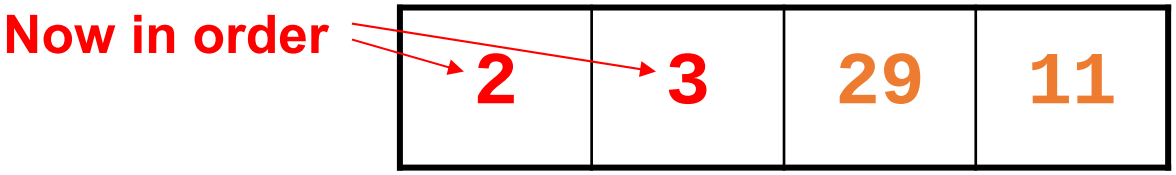
Smallest element is **2**. Exchange 2 with element in 1st array position (*i.e.* element 0).

Now in order

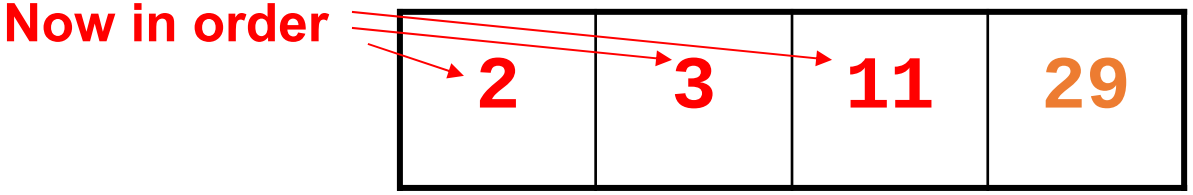


Selection Sort – Example (continued)

Next smallest element is **3**. Exchange **3** with element in 2nd array position.



Next smallest element is **11**. Exchange **11** with element in 3rd array position.



Selection Sort Tradeoffs

- Benefit
  - More efficient than Bubble Sort, due to fewer exchanges
- Disadvantage
  - Considered harder than Bubble Sort to understand

Sorting an Array of Objects

- As with searching, arrays to be sorted can contain objects or structures
- The key field determines how the structures or objects will be ordered
- When exchanging contents of array elements, entire structures or objects must be exchanged, not just the key fields in the structures or objects

Sorting and Searching Vectors

- Sorting and searching algorithms can be applied to vectors as well as to arrays
- Need slight modifications to functions to use vector arguments
  - **`vector <type> &`** used in prototype
  - No need to indicate vector size as functions can use **`size`** member function to calculate

Introduction to Analysis of Algorithms

- Given two algorithms to solve a problem, what makes one better than the other?
- Efficiency of an algorithm is measured by
  - space (computer memory used)
  - time (how long to execute the algorithm)
- Analysis of algorithms is a more effective way to find efficiency than by using empirical data

Analysis of Algorithms: Terminology

- Computational Problem: problem solved by an algorithm
- Basic step: operation in the algorithm that executes in a constant amount of time
- Examples of basic steps:
  - exchange the contents of two variables
  - compare two values