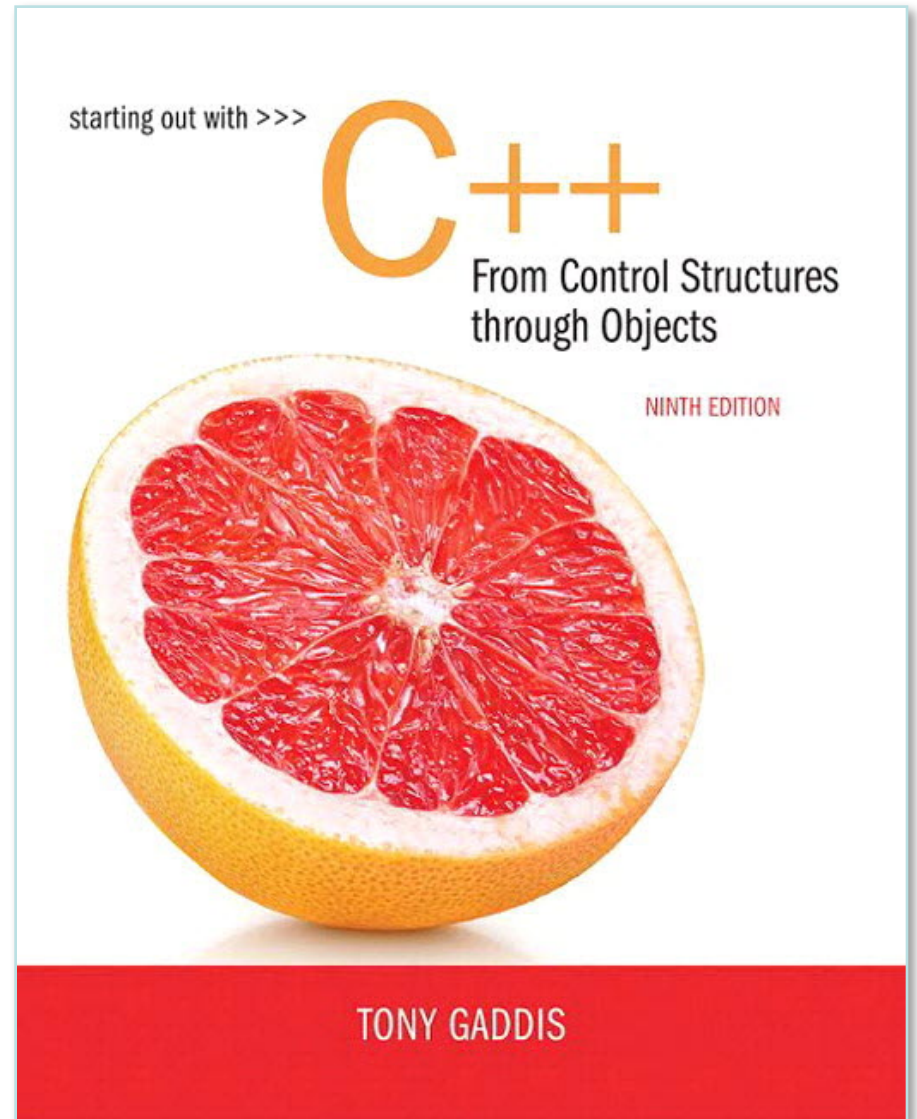
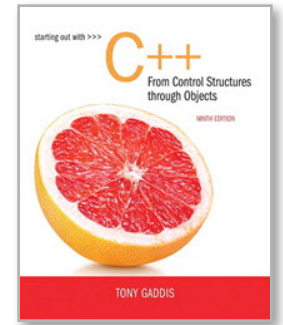


Chapter 8:

Searching and Sorting Arrays





8.1

Introduction to Search Algorithms

Introduction to Search Algorithms

- Search: locate an item in a list of information
- Two algorithms we will examine:
 - Linear search
 - Binary search

Linear Search

- Also called the sequential search
- Starting at the first element, this algorithm sequentially steps through an array examining each element until it locates the value it is searching for.

Linear Search - Example

- Array `numlist` contains:

17	23	5	11	2	29	3
----	----	---	----	---	----	---

- 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

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
```

A Linear Search Function

```
int linearSearch(int arr[], int size, int value)
{
    int index = 0;    // Used as a subscript to search the array
    int position = -1; // To record the position of search value
    bool found = false; // Flag to indicate if value was found

    while (index < size && !found)
    {
        if (arr[index] == value) // If the value is found
        {
            found = true; // Set the flag
            position = index; // Record the value's subscript
        }
        index++; // Go to the next element
    }
    return position; // Return the position, or -1
}
```

Linear Search - Tradeoffs

● Benefits:

- Easy algorithm to understand
- Array can be in any order

● Disadvantages:

- Inefficient (slow): for array of N elements, examines $N/2$ elements on average for value in array, N elements for value not in array

Binary Search

Requires array elements to be in order

1. Divides the 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:

2	3	5	11	17	23	29
---	---	---	----	----	----	----

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

Binary Search

Set first to 0

Set last to the last subscript in the array

Set found to false

Set position to -1

While found is not true and first is less than or equal to last

Set middle to the subscript half-way between array[first] and array[last].

If array[middle] equals the desired value

Set found to true

Set position to middle

Else If array[middle] is greater than the desired value

Set last to middle - 1

Else

Set first to middle + 1

End If.

End While.

Return position.

A Binary Search Function

```
int binarySearch(int array[], int size, int value)
{
    int first = 0,          // First array element
        last = size - 1,    // Last array element
        middle,             // Mid point of search
        position = -1;      // Position of search value
    bool found = false;     // Flag

    while (!found && first <= last)
    {
        middle = (first + last) / 2;    // Calculate mid point
        if (array[middle] == value)     // If value is found at mid
        {
            found = true;
            position = middle;
        }
        else if (array[middle] > value) // If value is in lower half
            last = middle - 1;
        else
            first = middle + 1;         // If value is in upper half
    }
    return position;
}
```



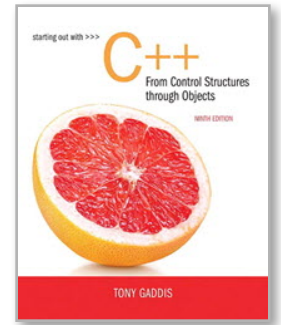
Binary Search - Tradeoffs

● Benefits:

- Much more efficient than linear search. For array of N elements, performs at most $\log_2 N$ comparisons

● Disadvantages:

- Requires that array elements be sorted



8.3

Introduction to Sorting Algorithms

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

Concept:

- Compare 1st two elements
 - If out of order, exchange them to put in order
- Move down one element, compare 2nd and 3rd elements, exchange if necessary. Continue until end of array.
- Pass through array again, exchanging as necessary
- Repeat until pass made with no exchanges

Example – First Pass

Array `numlist3` contains:

17	23	5	11
----	----	---	----

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

Example – Second Pass

After first pass, array `numlist3` contains:

17	5	11	23
----	---	----	----

compare values 17 and 5 – not in correct order, so exchange them

compare values 17 and 11 – not in correct order, so exchange them

compare values 17 and 23 – in correct order, so no exchange

Example – Third Pass

After second pass, array `numlist3` contains:

5	11	17	23
---	----	----	----

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

A Bubble Sort Function – From Program 8-4

```
37 void bubbleSort(int array[], int size)
38 {
39     int maxElement;
40     int index;
41
42     for (maxElement = size - 1; maxElement > 0; maxElement--)
43     {
44         for (index = 0; index < maxElement; index++)
45         {
46             if (array[index] > array[index + 1])
47             {
48                 swap(array[index], array[index + 1]);
49             }
50         }
51     }
52 }
53
54 //*****
55 // The swap function swaps a and b in memory.      *
56 //*****
57 void swap(int &a, int &b)
58 {
59     int temp = a;
60     a = b;
61     b = temp;
62 }
```

Bubble Sort - Tradeoffs

- Benefit:

- Easy to understand and implement

- Disadvantage:

- Inefficient: slow for large arrays

Selection Sort

- Concept for sort in ascending order:
 - Locate smallest element in array. Exchange it with element in position 0
 - Locate next smallest element in array. Exchange it with element in position 1.
 - Continue until all elements are arranged in order

Selection Sort - Example

Array `numlist` contains:

11	2	29	3
----	---	----	---

1. Smallest element is 2. Exchange 2 with element in 1st position in array:

2	11	29	3
---	----	----	---

Example (Continued)

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

2	3	29	11
---	---	----	----

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

2	3	11	29
---	---	----	----

A Selection Sort Function – From Program 8-5


```
37 void selectionSort(int array[], int size)
38 {
39     int minIndex, minValue;
40
41     for (int start = 0; start < (size - 1); start++)
42     {
43         minIndex = start;
44         minValue = array[start];
45         for (int index = start + 1; index < size; index++)
46         {
47             if (array[index] < minValue)
48             {
49                 minValue = array[index];
50                 minIndex = index;
51             }
52         }
53         swap(array[minIndex], array[start]);
54     }
55 }
```

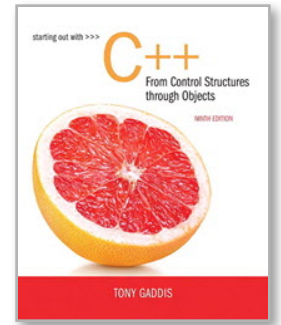
Selection Sort - Tradeoffs

Benefit:

-  More efficient than Bubble Sort, since fewer exchanges

Disadvantage:

-  May not be as easy as Bubble Sort to understand



8.5

Sorting and Searching Vectors

Sorting and Searching Vectors

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