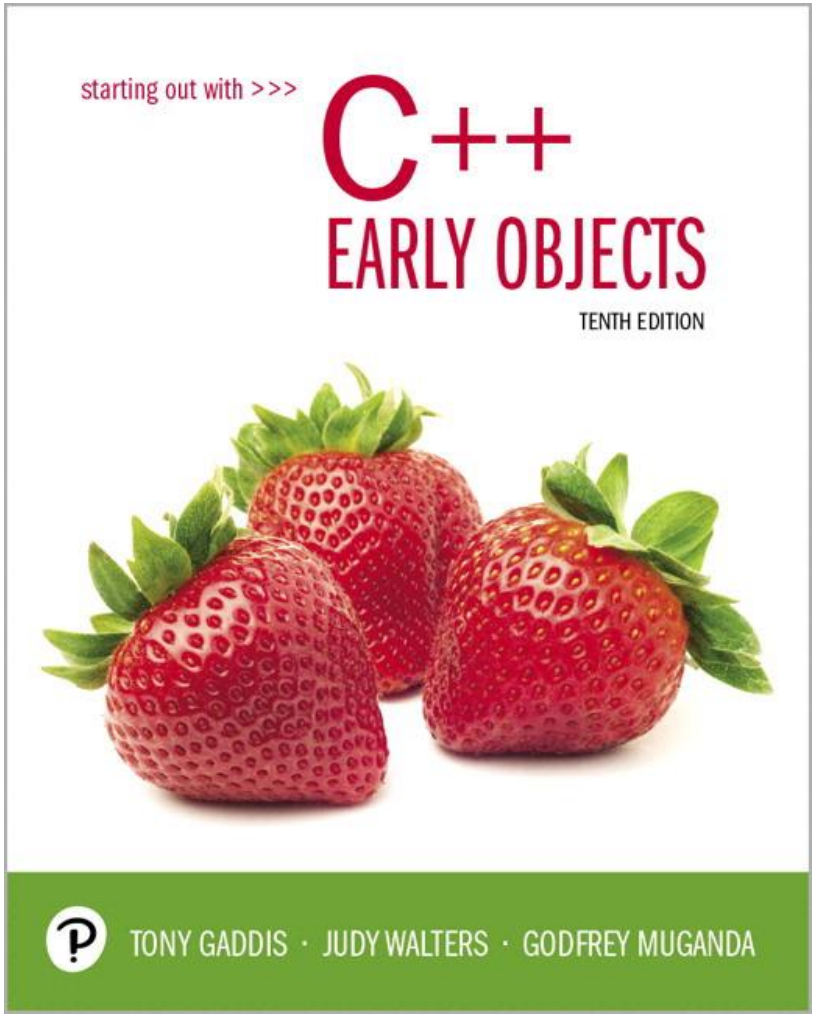
Starting Out with C++ Early Objects

Tenth Edition



Chapter 9

Searching, Sorting, and Algorithm Analysis

Topics

9.1 Introduction to Search Algorithms

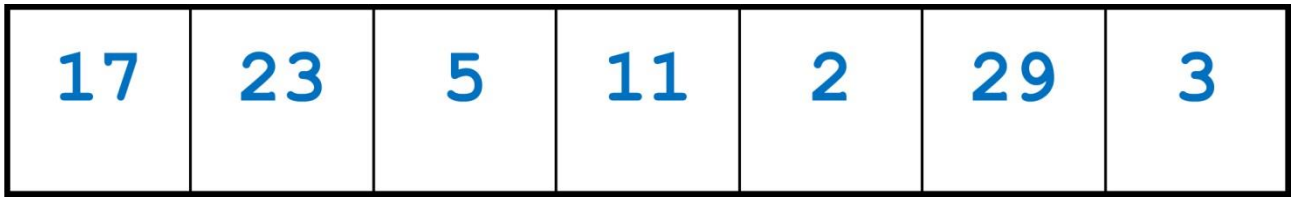
- Search: to locate a specific item in a list (array, vector, etc.) of information
- Two algorithms (methods) considered here:
  - Linear search (also called sequential search)
  - Binary search

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 and to implement
  - Elements in array can be in any order
- Disadvantage
  - Inefficient (slow): for array of N elements, it examines $N/2$ elements on average for a value that is found in the array, N elements for a value that is not in the array

Hands-On Pg. 614

- Listing 9-1

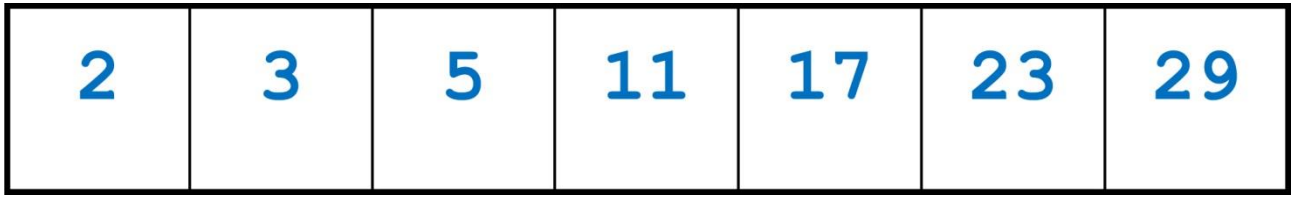
Binary Search Algorithm

Binary search requires that the array is in order.

1. Examine the value of the middle element
2. If the middle element has the desired value, done. Otherwise, determine which half of the array may have the desired value. Continue working with this portion of the array.
3. Repeat steps 1 and 2 until either the element with the desired 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,`** then stops

Binary Search Tradeoffs

- Benefit
  - It is much more efficient than linear search. For an array of N elements, the number of comparisons is the smallest integer x such that $2^x > N$. When N = 20,000, x is 15.
- However,
  - It requires that the array elements be in order

Hands-On Pg. 618

- Listing 9-2

9.2 Searching an Array of Objects

- Search algorithms are not limited to arrays of integers
- When searching an array of objects or structures, the value being searched for is a member of an object or structure, not the entire object or structure
- Member in object/structure: key field
- Value used in search: search key

Hands-On Pg. 620

- Listing 9-3

9.3 Introduction to Sorting Algorithms

- Sort: arrange values into an order
  - Alphabetical
  - Ascending (smallest to largest) numeric
  - Descending (largest to smallest) 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 the end of the array.
3. Pass through the array again, repeating the process and exchanging as necessary.
4. Repeat until a pass is made with no exchanges.

Bubble Sort Example 1 of 3

Array **`numlist3`** contains

Compare values 17 and 23. They are in order, so no exchange is needed.

Compare values 23 and 5. Exchange them.

Compare values 23 and 11. Exchange them.

This is the end of the first pass of sorting using Bubble Sort.

Bubble Sort Example 2 of 3

The second pass starts with the array from pass one

17	5	11	23

Compare values 17 and 5. Exchange them.

5	17	11	23

Compare 17 and 11. Exchange them.

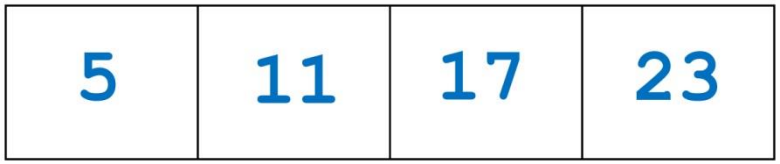
5	11	17	23

Compare 17 and 23. No exchange is needed.

This is the end of the second pass.

Bubble Sort Example 3 of 3

The third pass starts with the array from pass two



Compare values 5 and 11. No exchange is needed.

Compare values 11 and 17. No exchange is needed.

Compare values 17 and 23. No exchange is needed.

Since no exchanges were made, the array is in order.

Bubble Sort Tradeoffs

- Benefit
  - It is easy to understand and to implement
- Disadvantage
  - It is inefficient due to the number of exchanges. This makes it slow for large arrays

Hands-On Pg. 627

- Listing 9-4

An Improved Bubble Sort

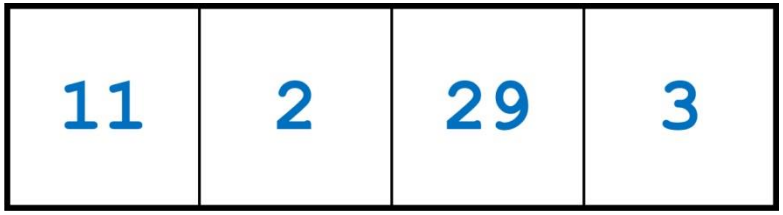
- If no exchanges occur in a pass using Bubble Sort, then the array must already be in order.
- This can be incorporated into the Bubble Sort algorithm:
  - detect if an exchange was made in the inner for loop and set a boolean flag.
  - test the flag in the outer for loop and terminate execution if no exchanges were made.

Selection Sort Algorithm

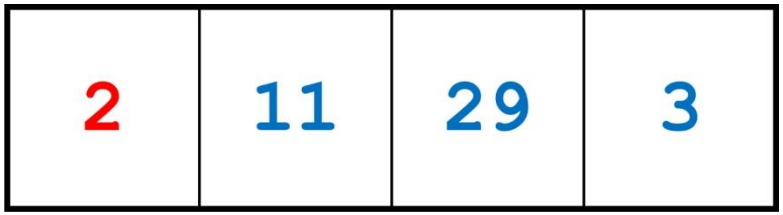
1. Locate the smallest element in the array and exchange it with the element in position 0.
2. Locate the next smallest element in the array and exchange it with element in position 1.
3. Continue until all of the elements are in order.

Selection Sort Example 1 of 2

Array **numlist** contains

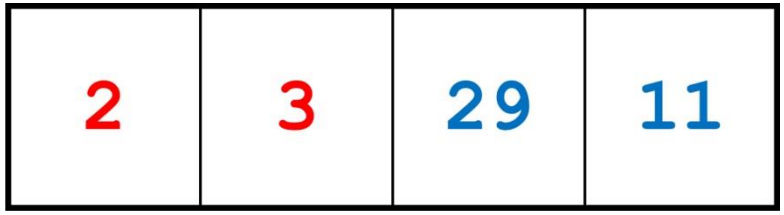


The smallest element is **2**. Exchange 2 with the element at subscript 0. The element in position 0 is now in order

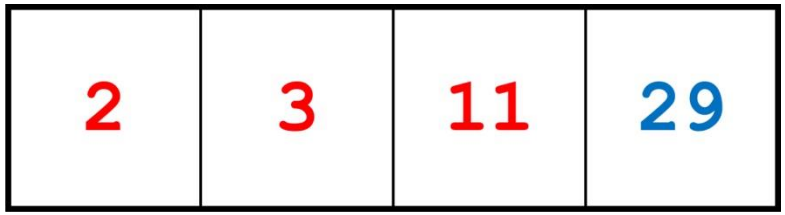


Selection Sort Example 2 of 2

The next smallest element is **3**. Exchange 3 with the element at subscript 1. The element in position 1 is now in order.



The next smallest element is **11**. Exchange 11 with the element at subscript 2.



Hands-On Pg. 632

- Listing 9-5

Sorting Considerations

- Although it does not have Bubble Sort's ability to terminate the sort when it detects that all elements are in order, Selection Sort is considered more efficient than Bubble Sort.
- This is due to the fewer number of exchanges that occur in Selection Sort.

9.4 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 the contents of array elements, entire structures or objects must be exchanged, not just the key fields in the structures or objects

Hands-On Pg. 634

- Listing 9-6

9.5 Sorting and Searching Vectors

- Sorting and searching algorithms can be applied to vectors as well as to arrays
- You need slight modifications to the sorting functions to use vector arguments
  - **`vector <type> &`** used in prototype
  - There is no need to indicate the vector size, as functions can use the vector's **`size`** member function

Hands-On Pg. 637

- Listing 9-7

9.6 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 1 of 3

- Computational Problem: a problem solved by an algorithm
- Instance of the Problem: a specific problem that is solved by the algorithm
- Size of an Instance: the amount of memory needed to hold the data for the specific problem

Analysis of Algorithms: Terminology 2 of 3

- Basic step: an 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
- Non-example of a basic step:
  - find the largest element in an array

Analysis of Algorithms: Terminology 3 of 3

- Complexity of an algorithm: the number of basic steps required to execute the algorithm for an input of size N (N = number of input values)
- Worst-case complexity of an algorithm: the number of basic steps for input of size N that requires the most work
- Average case complexity function: the complexity for typical, average inputs of size N

Complexity Example

Find the largest value in array A of size n

```
1.biggest = A[0]
2.indx = 0
3.while (indx < n) do
4.   if (A[n] > biggest)
5.   then
6.     biggest = A[n]
7.   end if
8.end while
```

Analysis:

Lines 1 and 2 execute once.

The test in line 3 executes n times.

The test in line 4 executes n times.

The assignment in line 6 executes at most n times.

Due to lines 3 and 4, the algorithm requires execution time proportional to n.

Comparison of Algorithmic Complexity

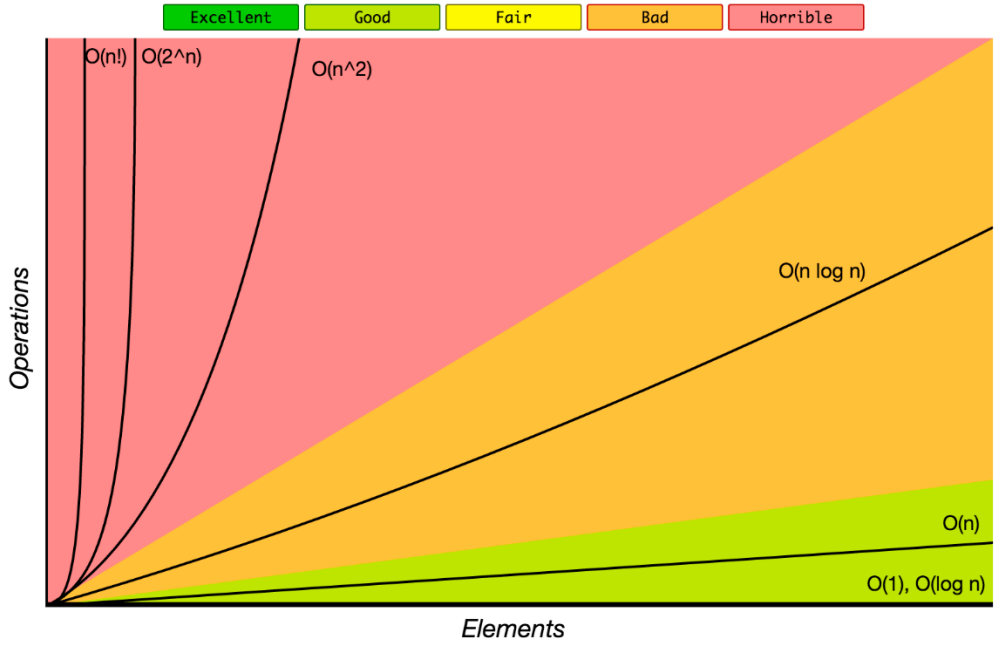
Given algorithms F and G with complexity functions $f(n)$ and $g(n)$ for input of size n

- If the ratio $\frac{f(n)}{g(n)}$ approaches a constant value as n gets large, F and G have equivalent efficiency
- If the ratio $\frac{f(n)}{g(n)}$ gets larger as n gets large, algorithm G is more efficient than algorithm F
- If the ratio $\frac{f(n)}{g(n)}$ approaches 0 as n gets large, algorithm F is more efficient than algorithm G

"Big O" Notation

- Function $f(n)$ is $O(g(n))$ ("f is big O of g") for some mathematical function $g(n)$ if the ratio $\frac{f(n)}{g(n)}$ approaches a positive constant as n gets large
- $O(g(n))$ defines a complexity class for the function $f(n)$ and for the algorithm F
- Increasing complexity classes means faster rate of growth and less efficient algorithms

Big-O Complexity Chart



Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	O(1)	O(1)	O(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Hands-On Pg. 650

- Listing 9-8

Algo. Analysis Cheat Sheet

- Canvas -> Files: *Algorithm Analysis – Quick Summary*
- Those on YouTube, post a comment on the video and I'll send it to you for free!

Learn More!

- Canvas -> Files: Analysis Of Algos.
- Those on YouTube, post a comment on the video and I'll send it to you for free!

Copyright