Searching and Sorting Arrays

CS 1: Problem Solving & Program Design Using C++

Objectives

- Search far and wide for different searching algorithms, including:
  - Linear search
  - Binary search
- Sort and sift through different sorting algorithms, including:
  - Bubble sort
  - Selection sort

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 elements 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 searchList(int list[], int numElems, int value)
{
   int index = 0;      // Used as a subscript to search array
   int position = -1;  // To record position of search value
   bool found = false; // Flag to indicate if value was found

   while (index < numElems && !found)
   {
      if (list[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
- 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
- If the middle element is the correct value, done
- Otherwise, go to the half of the array that may contain the correct value
- Continue 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 Algorithm

Set first index to 0.

Set last index 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].

Binary Search Algorithm (2)

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.

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
```

Binary Search Function (2)

```
    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

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

- 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

Bubble Sort Example: First Pass

- Array numlist3 contains:

Compare values 23 and 5 – not in correct order, so exchange them

17	23	5	11

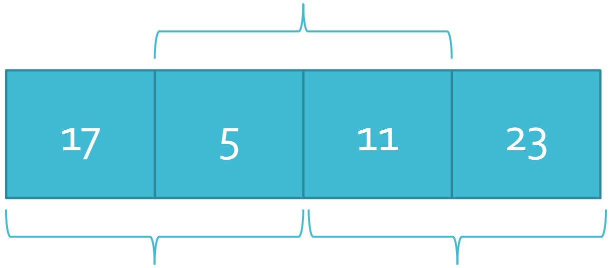
Compare values 17 and 23 – in correct order, so no exchange

Compare values 23 and 11 – not in correct order, so exchange them

Bubble Sort Example: Second Pass

- After first pass, array numlist3 contains:

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



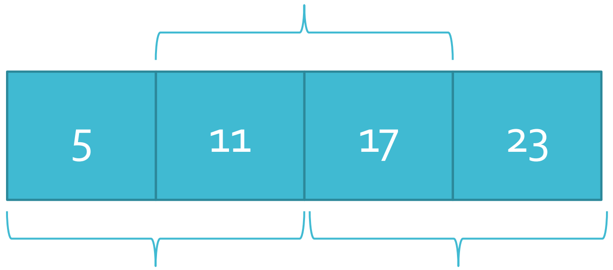
Compare values 17 and 5 – not in correct order, so exchange them

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

Bubble Sort Example: Third Pass

- After second pass, array numlist3 contains:

Compare values 11 and 17 – in correct order, so no exchange



Compare values 5 and 11 – in correct order, so no exchange

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

No exchanges needed, so array is in order

Bubble Sort Function

```
void sortArray (int array[], int size)
{
        bool swap;
        int temp;

        do
        {
                swap = false;
                for (int count = 0; count < size - 1; count++)
                {
                        if (array [count] > array [count + 1])
                        {
                                temp = array [count];
                                array [count] = array [count + 1];
                                array [count + 1] = temp;
                                swap = true;
                        }
                }
        } while (swap);
}
```

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

- Smallest element is 2; exchange 2 with element in 1st position in array:

2	11	29	3

Selection Sort Example (2)

- Next smallest element is 3; exchange 3 with element in 2nd position in array:

2	3	29	11

- Next smallest element is 11; exchange 11 with element in 3rd position in array:

2	3	11	29

Selection Sort Function

```
void selectionSort(int array[], int size)
{
    int startScan, minIndex, minValue;

    for (startScan = 0; startScan < size - 1; startScan++)
    {
        minIndex = startScan;
        minValue = array[startScan];
        for(int index = startScan + 1; index < size; index++)
        {
            if (array[index] < minValue)
            {
                minValue = array[index];
                minIndex = index;
            }
        }
        array[minIndex] = array[startScan];
        array[startScan] = minValue;
    }
}
```

Selection Sort Tradeoffs

- Benefit:
  - More efficient than Bubble Sort, since fewer exchanges
- Disadvantage:
  - May not be as easy as Bubble Sort to understand

Summary

- Looked at the following searches
  - Linear search
  - Binary search
- Sorted through the following sorts
  - Bubble sort
  - Selection sort