INTRODUCTION TO SEARCH ALGORITHMS

• Search: locate an item in a list of information

• Two algorithms we will examine:
  • Linear search
  • Binary search
INTRODUCTION TO SEARCH ALGORITHMS

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.
- Array numlist contains:

| 17 | 23 | 5  | 11 | 2  | 29 | 3  |

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

INTRODUCTION TO SEARCH ALGORITHMS

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
INTRODUCTION TO SEARCH ALGORITHMS

A LINEAR SEARCH FUNCTION

```c
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
}
```

INTRODUCTION TO SEARCH ALGORITHMS

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
INTRODUCTION TO SEARCH ALGORITHMS

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

INTRODUCTION TO SEARCH ALGORITHMS

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
INTRODUCTION TO SEARCH ALGORITHMS

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 halfway 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.

INTRODUCTION TO SEARCH ALGORITHMS

A BINARY SEARCH FUNCTION

```c
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;
}
```
INTRODUCTION TO SEARCH ALGORITHMS

BINARY SEARCH - TRADEOFFS

- Benefits:
  - Much more efficient than linear search. For array of N elements, performs at most log2N 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
INTRODUCTION TO SORTING ALGORITHMS

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

INTRODUCTION TO SORTING ALGORITHMS

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
INTRODUCTION TO SORTING ALGORITHMS

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
INTRODUCTION TO SORTING ALGORITHMS
A BUBBLE SORT FUNCTION – FROM PROGRAM

```c
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);
}
```

INTRODUCTION TO SORTING ALGORITHMS
BUBBLE SORT - TRADEOFFS

- Benefit:
  - Easy to understand and implement

- Disadvantage:
  - Inefficient: slow for large arrays
INTRODUCTION TO SORTING ALGORITHMS

SELECTION SORT

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

INTRODUCTION TO SORTING ALGORITHMS

SELECTION SORT - EXAMPLE

Array \texttt{numlist} contains:

\begin{array}{cccc}
11 & 2 & 29 & 3 \\
\end{array}

1. Smallest element is 2. Exchange 2 with element in 1\textsuperscript{st} position in array:

\begin{array}{cccc}
2 & 11 & 29 & 3 \\
\end{array}

2. Next smallest element is 3. Exchange 3 with element in 2\textsuperscript{nd} position in array:

\begin{array}{cccc}
2 & 3 & 29 & 11 \\
\end{array}

2. Next smallest element is 11. Exchange 11 with element in 3\textsuperscript{rd} position in array:

\begin{array}{cccc}
2 & 3 & 11 & 29 \\
\end{array}
INTRODUCTION TO SORTING ALGORITHMS

A SELECTION SORT FUNCTION - FROM PROGRAM

35 void selectionSort(int array[], int size)
36 {
37     int startScan, minIndex, minValue;
38     for (startScan = 0; startScan < (size - 1); startScan++)
39     {
40         minIndex = startScan;
41         minValue = array[startScan];
42         for(int index = startScan + 1; index < size; index++)
43         {
44             if (array[index] < minValue)
45             {
46                 minValue = array[index];
47                 minIndex = index;
48             }
49         }
50         array[minIndex] = array[startScan];
51         array[startScan] = minValue;
52     }
53 }

INTRODUCTION TO SORTING ALGORITHMS

SELECTION SORT - TRADEOFFS

• Benefit:
  • More efficient than Bubble Sort, since fewer exchanges

• Disadvantage:
  • May not be as easy as Bubble Sort to understand
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:
  • \texttt{vector <type> &} used in prototype
  • No need to indicate vector size – functions can use size member function to calculate