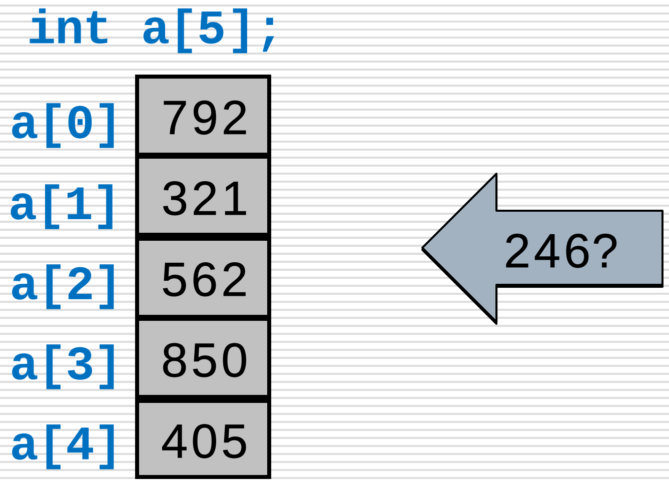
Ch8: Searching and Sorting Arrays

- Searching Arrays
  - Linear Search
  - Binary Search
- Sorting Arrays
  - Bubble Sort
  - Selection Sort

Searching Algorithms

- Search algorithms attempt to locate a specific item in a larger collection of data
  - Linear Search
  - Binary Search



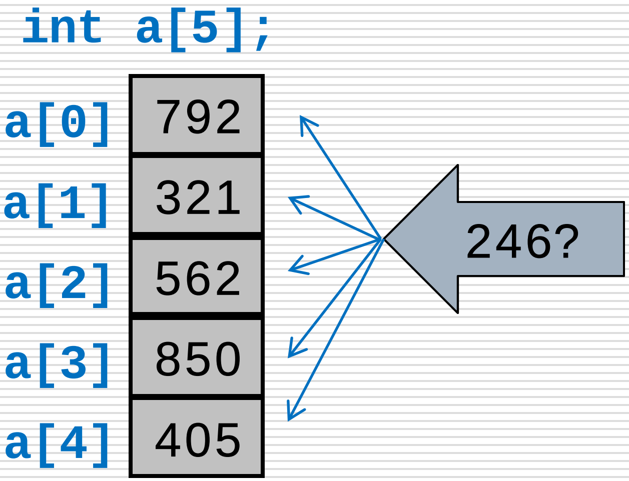
`int a[5];`

a[0]	792
a[1]	321
a[2]	562
a[3]	850
a[4]	405

246?

Linear Search

- Uses loop to sequentially compare each array element with search value
  - Stops when element is found or end of array is reached
  - Array elements are unordered



int a[5];

a[0] 792

a[1] 321

a[2] 562

a[3] 850

a[4] 405

246?

Linear Search

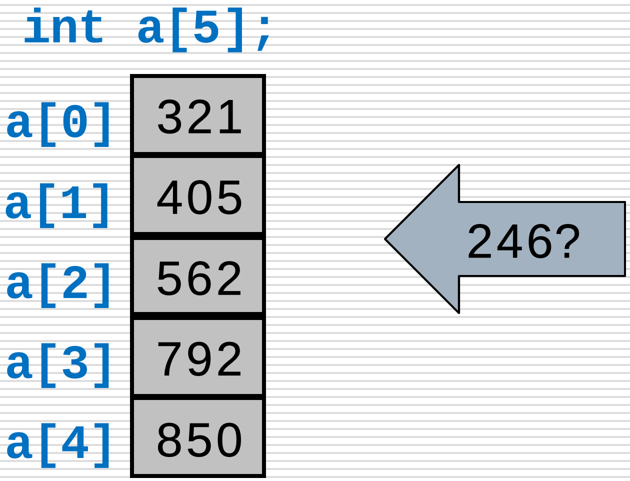
```
Set found to false
Set position to -1
Set index to 0
While found is false and index < number of elements
    If list[index] is equal to search value
        found = true
        position = index
    End If
    Add 1 to index
End While
Return position
```

Linear Search

```
int linearSearch(const 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 the 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
}
```

Binary Search

- Loops to divide array in half by comparing search value with middle element and determine relevant half
  - □ Stops when element is found or potential array location has passed
  - □ More efficient than linear search
  - □ Array elements must be in sorted order



```
int a[5];
```

a[0]	321
a[1]	405
a[2]	562
a[3]	792
a[4]	850

Binary Search

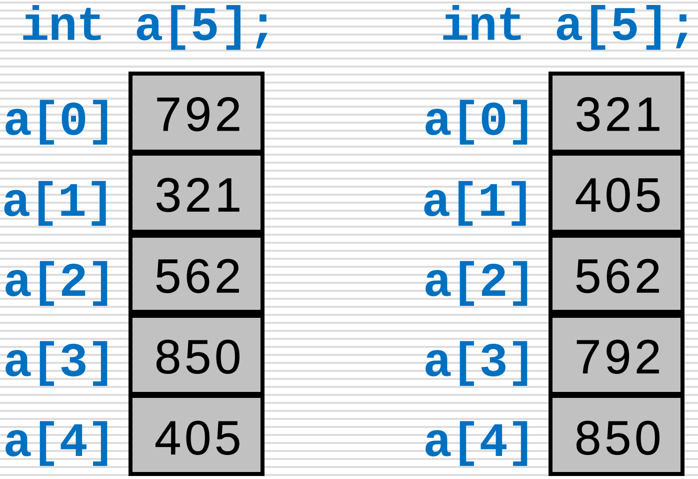
```
Set first index to 0
Set last index to the last subscript in the array
Set found to false
Set position to −1
While found is not true and first is less than or equal to last
    Set middle to the subscript halfway between array[first] and array[last]
    If array[middle] equals the desired value
        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
```

Binary Search

```
int binarySearch(const int array[], int numElems, int value)
{
    int first = 0,                          // First array element
    last = numElems – 1,                    // Last array element
    middle,                                 // Midpoint of search
    position = –1;                          // Position of search value
    bool found = false;                     // Flag
    while (!found && first <= last) {
        middle = (first + last) / 2;       // Calculate midpoint
        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;
}
```

Sorting Algorithms

- Sorting algorithms arrange array elements in a defined order
  - Ascending or increasing
  - Descending or decreasing
- Examples
  - Bubble Sort
  - Selection Sort



```
int a[5];
```

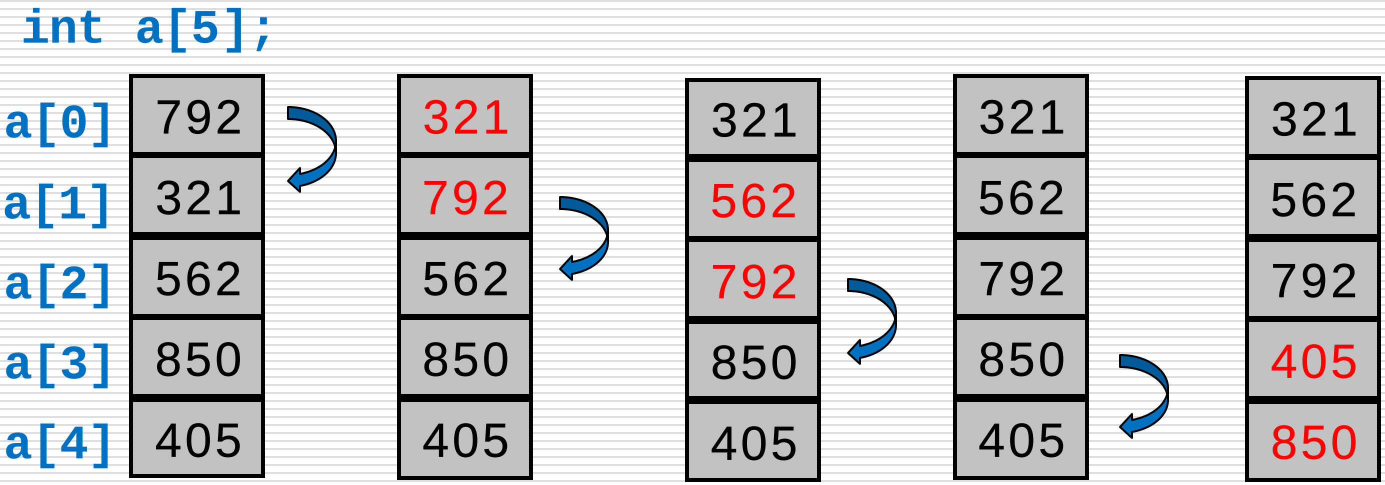
a[0]	792
a[1]	321
a[2]	562
a[3]	850
a[4]	405

```
int a[5];
```

a[0]	321
a[1]	405
a[2]	562
a[3]	792
a[4]	850

Bubble Sort

- Nested loops to compare successive elements and swaps if out of order
  - Largest item is ‘bubbled’ to the bottom



```
int a[5];
```

a[0]	792	321	321	321	321
a[1]	321	792	562	562	562
a[2]	562	562	792	792	792
a[3]	850	850	850	850	405
a[4]	405	405	405	405	850

Bubble Sort Pseudocode

```
Do
    Set swap flag to false
    For count is set to each subscript in array from 0 through the next-to-last subscript
        If array[count] is greater than array[count+1]
            Swap the contents of array[count] and array[count+1]
            Set swap flag to true
        End If
    End For
While any elements have been swapped
```

Bubble Sort C++ Function

```
void bubbleSort(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);
}
```

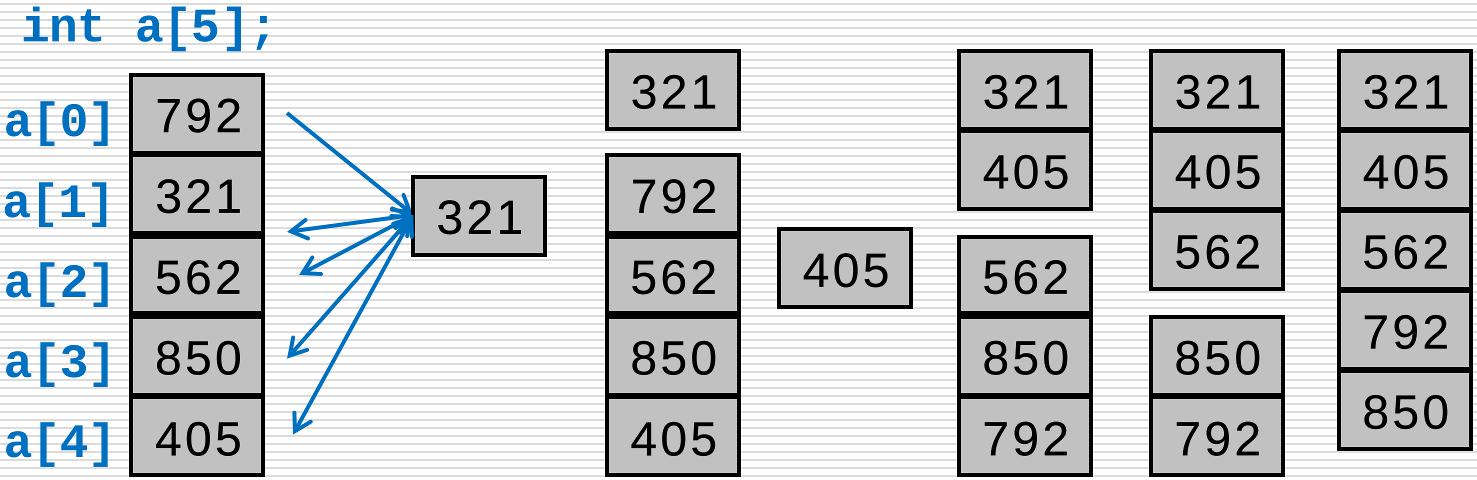
swap values

see if current is greater than next

continue while swap has been made

Selection Sort

- Moves elements immediately to their final position in the array
  - Loop with unsorted sublist
    - Locate largest/smallest element and place in final position



Selection Sort Pseudocode

```
For startScan is set to each subscript in array from 0 through the next-to-last subscript
    Set index variable to startScan
    Set minIndex variable to startScan
    Set minValue variable to array[startScan]
    For index is set to each subscript in array from (startScan + 1) through the last subscript
        If array[index] is less than minValue
            Set minValue to array[index]
            Set minIndex to index
        End If
    End For
    Set array[minIndex] to array[startScan]
    Set array[startScan] to minValue.
End For.
```

Selection Sort C++ Code

```
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;
    }
}
```

initialize smallest

find smallest

swap value