COP 3331
OBJECT ORIENTED DESIGN
SPRING 2017

WEEK 14: SEARCH AND SORT ALGORITHMS

SCHINNEL SMALL

UNIVERSITY OF SOUTH FLORIDA

SEARCH ALGORITHMS

SEARCH ALGORITHMS

- A search algorithm is a process for locating a specific item in a larger collection of data
- The two common methods of searching for data are the linear (or sequential search) and the binary search
- The concepts will be described for use in an array but can be applied to other data structures, such as vectors

THE LINEAR SEARCH

- In the linear search, a loop is used to sequentially step through an array, examining each element until it locates the value it is searching for
- Example: Array `numlist` contains:

17	23	5	11	2	29	3

- Searching for the the value `11`, the 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`

THE LINEAR SEARCH

- Example of linear search function (example 1, week 14 docs)

```
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
}
```

THE LINEAR SEARCH

- Advantages:
  - Simple algorithm
  - Contents of array can be in any order
- Disadvantages:
  - Inefficient and slow for large arrays
    - For an array of size N, the algorithm may examine N/2 elements on average if the value is in the array
    - If the value is not in the same array, it examines N values

THE BINARY SEARCH

- The binary search is more efficient than the linear search

- It does, however, require the array elements to be sorted (i.e. in order)

- This search 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

THE BINARY SEARCH

- If the middle element is the correct value, then we are done!

- If the middle element is not the correct value, then its value must be higher or lower than the desired value
  - If it is higher then search the first half of the list
  - If it is lower, then search the second half

- The process is repeated for the selected half until the value is found (if it exists)

THE BINARY SEARCH

- 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

THE BINARY SEARCH

- Example of linear search function (example 2, week 14 docs)

```
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;
}
```

THE BINARY SEARCH

- Advantages:
  - More efficient than linear search
    - Can perform at most $\boldsymbol{log_2 N}$ comparisons
- Disadvantages:
  - Requires that array elements must be sorted

SORT ALGORITHMS

SORT ALGORITHMS

- A sort algorithm is a process that is used to arrange data into some order

- There are two common sorting algorithms to consider:
  - Bubble Sort
  - Selection Sort

- Once again, we illustrate the concept using an array, but it can be applied to other structures

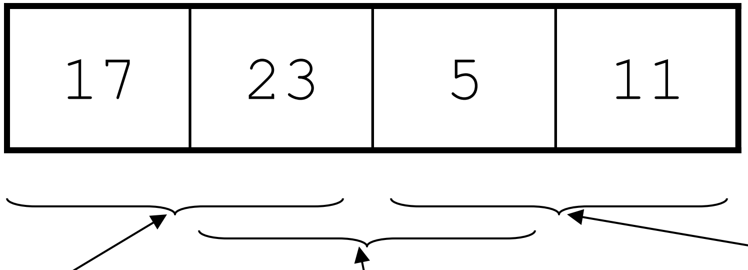
BUBBLE SORT

- In the bubble sort algorithm, the first two elements are compared
  - If they are out of order, exchange them to put in order
- The algorithm moves through the array, comparing the next two elements and exchanging if necessary
- The algorithm passes through the entire structure repeatedly until no exchanges can be made

BUBBLE SORT

- Array `numlist3` contains:

17	23	5	11



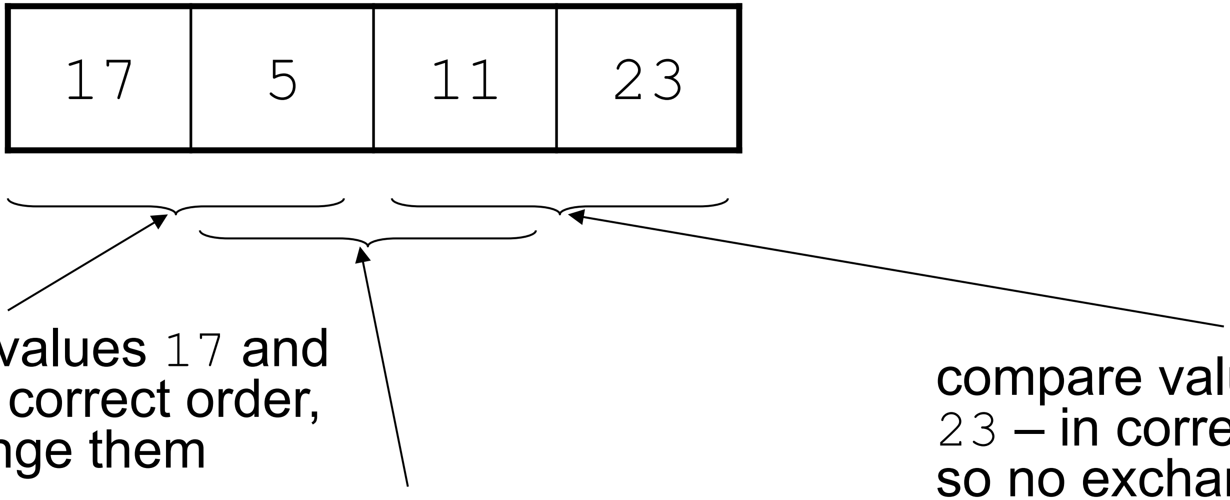
compare values 17 and 23 – in correct order, so no exchange

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

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

BUBBLE SORT

- After first pass, array `numlist3` contains:



17	5	11	23

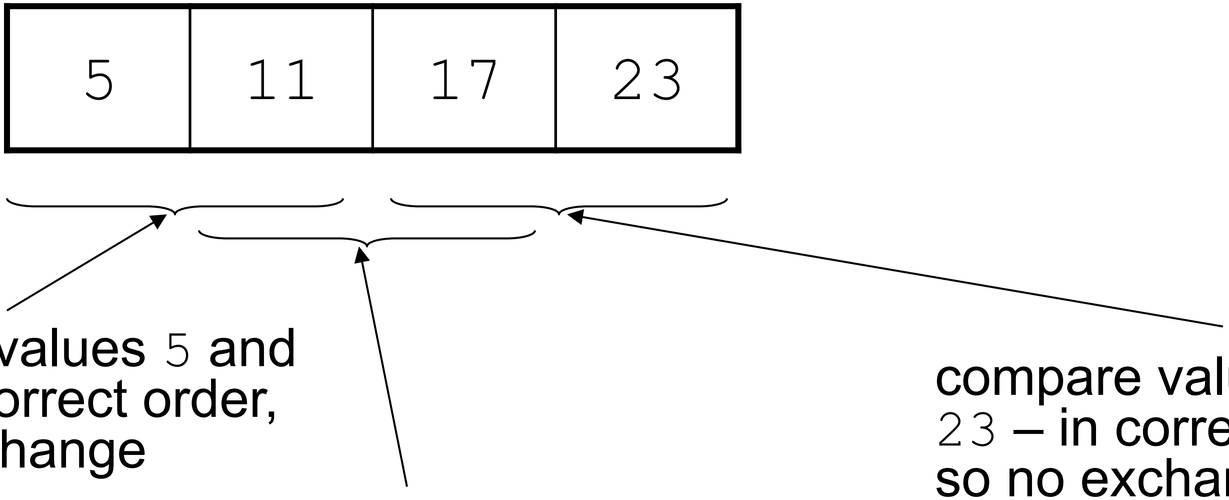
compare values 17 and 5 – not in correct order, so exchange them

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

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

BUBBLE SORT

- After second pass, array `numlist3` contains:



5	11	17	23

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

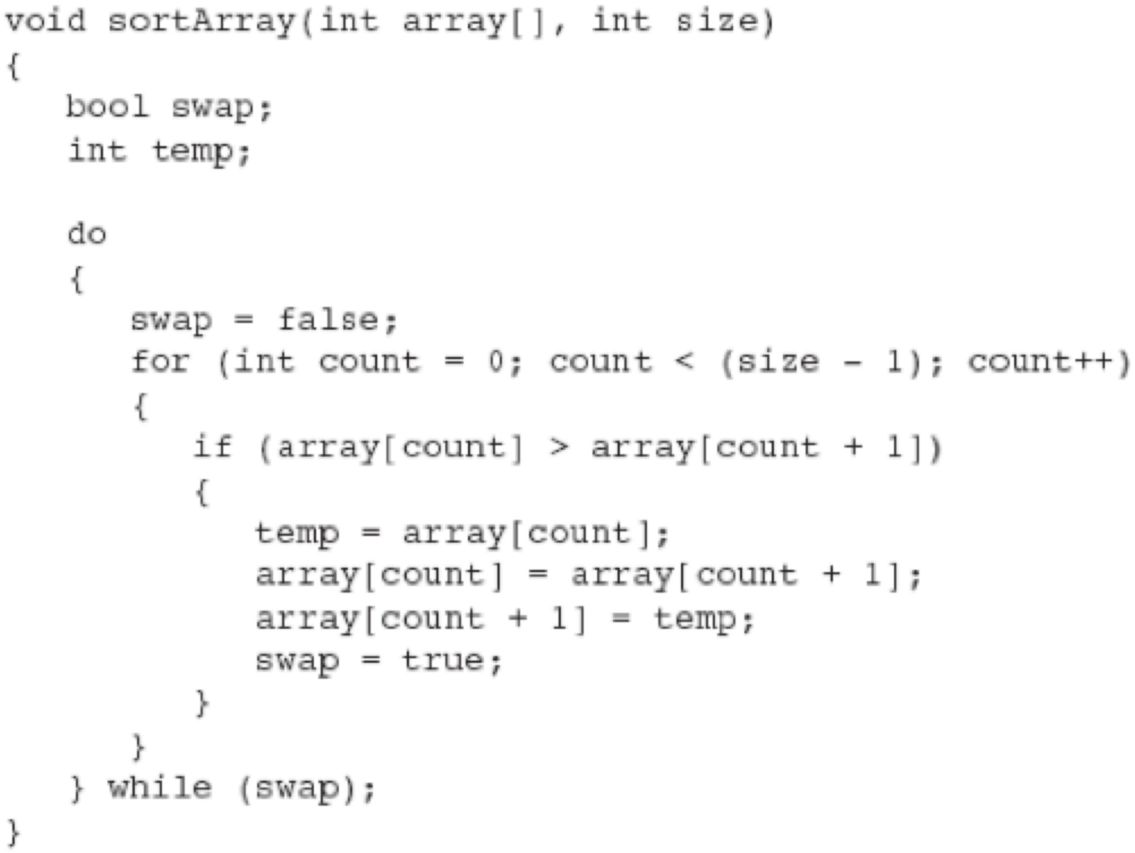
compare values 11 and 17 – in correct order, so no exchange

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

No exchanges, so array is in order

BUBBLE SORT

- Example of a bubble sort function (see example 3 in wk 14 docs)



```
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

- Advantages:
  - Simple to implement
- Disadvantages
  - Inefficient: slow for large arrays

SELECTION SORT

- The selection sort is more efficient than the bubble sort
- It locates the smallest element in the array and exchanges it with the element in the first position
- It then locates the next smallest element in the array and exchanges with the element in the second position
- This process repeats until all elements are arranged in order

SELECTION SORT

- 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

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

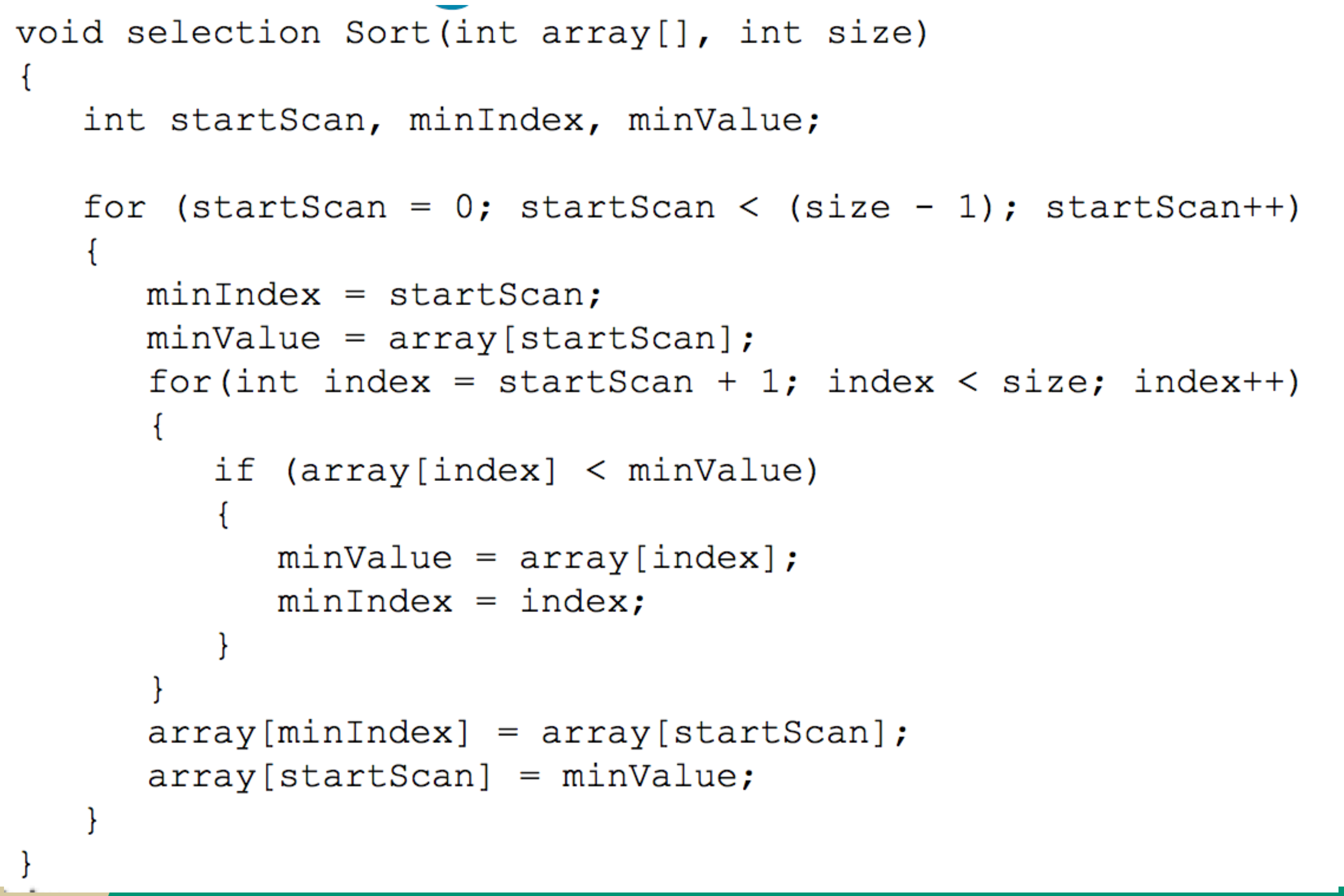
2	3	29	11

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

2	3	11	29

SELECTION SORT

- Example of a selection sort function (see example 4 in wk 14 docs)



```
void selection Sort(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

- Advantage:
  - More efficient algorithm
- Disadvantage:
  - Harder to understand/implement