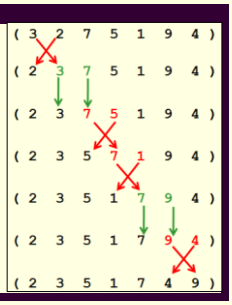
Bubble Sort

CS 241
Data Organization using C

Instructor: **Joel Castellanos**
e-mail: joel@unm.edu
Web: http://cs.unm.edu/~joel/



9/19/2019

Bubble Sort Algorithm

- Given a list of unsorted numbers.
- Each pass consists of:
  1. Starting with the first number in the list.
  2. Walking through the list and comparing each number with the number that is one position father down the list.
  3. In each comparison, if the first number is greater than the second, then swap the two numbers.
- Continue making passes until a full pass is completed without making any swaps.
- *Optimization:* After `m` passes it is guaranteed that the last `m` numbers in the list will be sorted. Therefore, each pass need not compare the last `m` numbers.

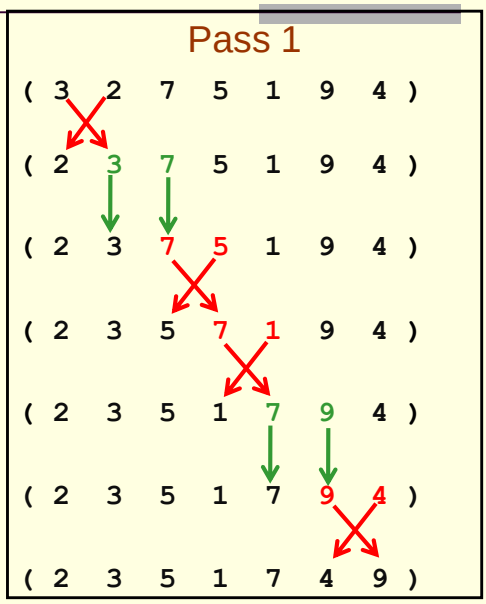
Bubble Sort – Pass 1

Original Numbers

`{3, 2, 7, 5, 1, 9, 4}`

Each pass consists of walking through the list and comparing each number with the number that is one position father down the list.

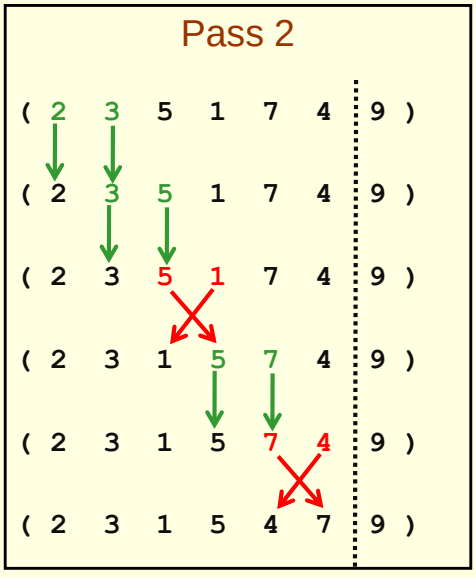
In each comparison, if the first number is greater than the second, then swap.



3

3

Bubble Sort – Pass 2



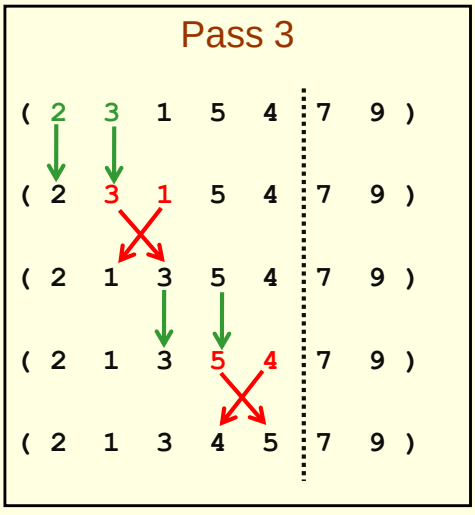
After the first pass, the largest number will have been moved to the end of the list.

Therefore, in pass 2, only the first 6 numbers need to be compared.

4

4

Bubble Sort – Pass 3



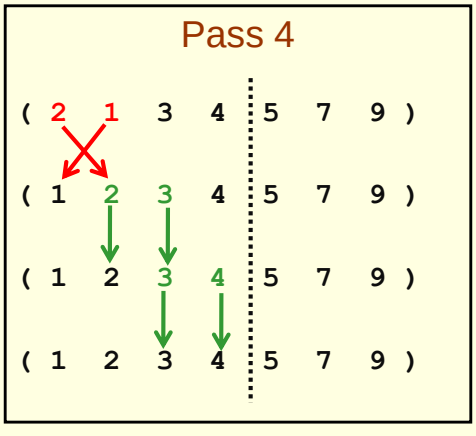
After the second pass, the second largest number will have been moved to the second to last position in the list.

Therefore, in pass 3, only the first 5 numbers need to be compared.

5

5

Bubble Sort – Pass 4

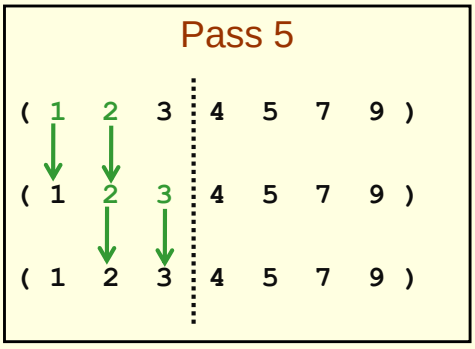


After the first step of pass 4, the list is completely sorted; However, the algorithm does not “know” the list is sorted until a full pass is made with no swapping.

6

6

Bubble Sort – Pass 5



After 4 passes, the last 4 numbers are sorted, so only the first 3 numbers need to be compared.

Nothing is swapped in pass 5. Therefore, the list is fully sorted.

Bubble Sort: Performance

- Bubble sort has worst-case and average complexity both O(n^2), where n is the number of items being sorted.
- There exist many sorting algorithms with the substantially better worst-case or average complexity of O(n log n).
- For example, for n = 1 million,
  - n^2 = 1,000,000,000,000
  - n log n = 13,815,511
- Therefore bubble sort is not a practical sorting algorithm when n is large, except in rare specific applications where the array is known to be very close to being initially sorted.

Bubble Sort on an `int` Array

```
1) void bubbleSort(int array[], int n)
2) { int i, swap = 1;
3)
4)   while(swap)
5)   { swap = 0;
6)     for (i=0; i<n-1; i++)
7)     {
8)       if (array[i] > array[i+1])
9)       { int tmp = array[i];
10)        array[i] = array[i+1];
11)        array[i+1] = tmp;
12)        swap = 1;
13)      }
14)    }
15)  }
16)}
```

9

Bubble Sort: `main(...)`

```
void main(void)
{
  int numList[] = {3, 2, 7, 5, 1, 9, 4};
  int n = sizeof(numList)/sizeof(int);

  bubbleSort(numList, n);

  int i;
  for (i=0; i<7; i++)
  {
    printf("%d ", numList[i]); //1 2 3 4 5 7 9
  }
  printf("\n");
}
```

10

Use of `sizeof` with Arrays

```
#include <stdio.h>
void foo(int array[])
{
  int n = sizeof(array);
  printf("foo: sizeof(array)=%d\n", n);
}

void main(void)
{
  int array[] = {3, 2, 7, 5, 1};
  int n = sizeof(array);
  printf("main: sizeof(array)=%d\n", n);

  foo(array);
}
```

Output:

```
main: sizeof(array)=20
foo: sizeof(array)=8
```

11

11

Quiz: Bubble Sort

```
1. void bubbleSort(int array[], int n)
2. { int i, swap = 1;
3.   while(swap)
4.   { swap = 0;
5.     for (i=0; i<n-1; i++)
6.     {
7.       if (array[i] > array[i+1])
8.       { int tmp = array[i];
9.         array[i] = array[i+1];
10.        array[i+1] = tmp;
11.        swap = 1;
12.      }
13.    }
14.  }
15.}
```

Efficiency can be improved by adding the statement `n--;` between which two lines of code?

a) 2 & 3 b) 4 & 5 c) 6 & 7
d) 12 & 13 e) 13 & 14

12

12