CSc 28
Discrete Structures

Chapter 14
Sorting

Herbert G. Mayer, CSU CSC

Status 1/1/2021

Syllabus

- Sorting Arrays
- Bubble Sort
- Insertion Sort
- References

Bubble Sort

Sorting Arrays

- Data in computing sometimes are to be sorted
- Data may be all distinct, or even vacuous, or contain duplicates
- If duplicates we decide, whether multiple entries are allowed in the data structure after the sort
- If not allowed, data structure will actually change: the total quantity is reduced!
- Generally it is OK to have multiple occurrences
- Decide to sort in ascending or descending order
- Occasional stickler ☺ may rephrase that as: “non-ascending” or “non-descending”

Sorting Arrays

- Other times data reside in trees, lists, graphs, on files
- May also be or get sorted; can be critical for trees!
  - Else the tree is unbalanced and any lookup may be costly
- Also critical for some applications
- Focus here simple: Sort an single-dimensional array of ints with MAX entries; array named: int a[MAX]
- Arrays are prototypical samples of Discrete Structures
- C++ specification: `int a[ MAX ];`
- For demonstration, array a[12] will be pre-initialized:

```
int a[]={ -1,1,-2,2,100,-100,12,-12,1234,-9999,0,-2 };
```

Sorting Balls

Print Array

```
#define . . . MAX etc.
#include <iostream>
using namespace std;

// global array of ints: int a[]
int a[ /* MAX */ ] = { -1, 1, -2, 2, 100, -100, 12,
   -12, 1234, -9999, 0, -2 };

// print all MAX elements of global a[]
void print( char * msg )
{ // print
   cout << "Array a[] " << msg << endl;
   for( int i = 0; i < MAX; i++ ) {
       cout << ' ' << a[ i ];
   } //end for
   cout << endl;
} //end print
```

Bubble Sort Array

```
// bubble sort array a[] of ints; global an exception ☺
// outer loop start index 0; ends 1 before list-end
// inner loop starts at index+1; ends at last index

void sort()    // descending
{ // sort
   for( int left = 0; left < MAX-1; left++ ) {
     for( int right = left+1; right < MAX; right++ ) {
       // is right element greater? if so: manual swap
       if( a[ right ] > a[ left ] ) { // descending
          int temp    = a[ right ];
          a[ right ]  = a[ left ];
          a[ left ]   = temp;
       } //end if
     } //end for
   } //end for
} //end sort
```

Print Sorted Array

```
int main( void )
{ // main
  // print global array a[] unsorted
  print( "before sorting:" );

  sort();

  // print global array a[] sorted
  print( "after sorting:" );
} //end main
```

Execute

```
a.out

Array a[] before sorting:

-1 1 -2 2 100 -100 12 -12 1234 -9999 0 -2

Array a[] after sorting:

1234 100 12 2 1 0 -1 -2 -2 -12 -100 -9999
```

Swap Array Element

- Now we include a swap(a, b) C++ function
- Note: C subset defines only value parameters
  - Except for array type parameters
  - Arrays are passed by reference in C++ and in C!
- C++ also defines reference parameters, via the & type specifier for formal parameters
  - Hence C++ arrays do not permit the reference specification & for array formal parameters
  - would be redundant
- Note: if array happens to be totally sorted from the start, the number of iterations is still $O(n^2)$

Swap Array Element

```
// assume array x[] of ints passed in; note: by ref!
// given a left index l, and right index r
// exchange elements in x[] at l and r indices!
// i.e. we swap one pair of array elements

void swap( int x[], int l, int r )  // no need for &
// note x[] in C++ passed by reference!
{ // swap
   int temp = x[ l ];
   x[ l ] = x[ r ];
   x[ r ] = temp;
} //end swap
```

Swap Array Element

```
// simple bubble sort of aray a[] of ints
// start at index 0 for the left index
// view other elements on right, from left+1 to the end

void sort()    // descending
{ // sort
   for( int left = 0; left < MAX-1; left++ ) {
     for( int right = left+1; right < MAX; right++ ) {
       // is the right element greater?
       // if so, then swap
       if( a[ right ] > a[ left ] ) {
          swap( a, left, right );
       } //end if
     } //end for
   } //end for
} //end sort
```

Print Sorted Array

```
[sort] a.out
Array a[] before sorting:
-1 1 -2 2 100 -100 12 -12 1234 -9999 0 -2
Array a[] after sorting:
1234 100 12 2 1 0 -1 -2 -2 -12 -100 -9999
```

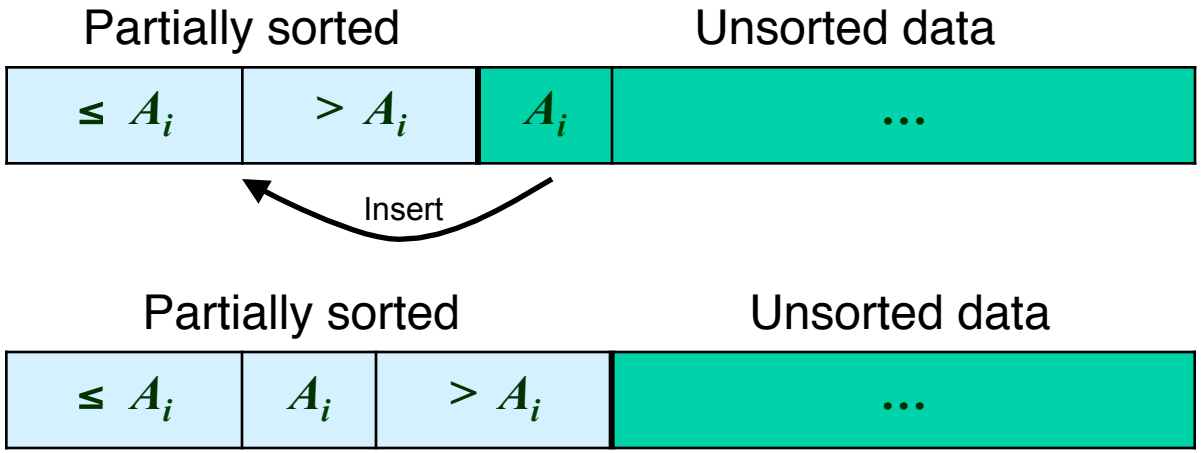
Bubble Sort

- Simple Bubble Sort uses two nested loops
- Hence time complexity is $O(n^2)$
- Can be improved, by ending nested loop early, when rest of array is *already sorted*
- Yet Bubble Sort is fairly inefficient for arrays of interesting size
- Instead must find better methods

Insertion Sort

Insertion Sort

- **Insertion sort: algorithm that is relatively efficient for mostly sorted lists**
- **List elements are removed, then placed, one at a time and inserted in their correct position in a new sorted list**
- **Rest of list is moved up (or down) by one position, enabled via the place freed by the relocated element**



Insertion Sort

- If the original list is largely unsorted, the cost for insertion sort becomes similar, worst case even equal to bubble sort
- For lists that are almost totally sorted, the cost for insertion sort ends up quite low
- Can even be O(1) in Big-O notation, something never possible with pure (dumb ☺)Bubble Sort

Insertion Sort

- Goal is a list a[] in ascending order:
- Start at index `i = 1`, fetch `value = a[i];` then all the way up the last element `i = MAX-1`
- Set `j = i-1` and compare `value` against `a[j]`
- Whenever element `a[j]` is larger than `value`, it is out of place, it must be shifted to a higher index, up to where `value` was fetched
- In the end, `value` is placed into the slot freed

Insertion Sort

```
// defined: int a[], int j, int value . . .
// very clever
for( int i = 1; i < MAX-1; i++ ) {
  value = a[ i ];
  j = i - 1;
  while( ( j >= 0 ) && ( a[ j ] > value ) ) {
     a[ j+1 ] = a[ j ];  // push up
     --j;                // more to move up?
  } //end while
  a[ j + 1 ] = value;    // the right place
} //end for
```

Insertion Sort

- Simplicity of the Insertion Sort algorithm is striking
- Cost is not worse than that of Bubble Sort
- For lucky cases, the cost function can be way lower than the $O(n^2)$ cost of the bubble sort
- Lucky means, good part of array is already sorted!
- In rare cases, when the list is mostly sorted to begin with, cost for the total sort may be as low as $O(n)$, something not possible with the bubble sort

References

1. Wiki on C+ parameters: https://www.tutorialspoint.com/cplusplus/passing_parameters_by_references.htm
2. https://en.wikipedia.org/wiki/Insertion_sort