DATA STRUCTURES AND ALGORITHMS

Lecture: Array Sorting

Lecturer: Mohsin Abbas

National University of Modern Languages, Islamabad

SORTING

- Sorting takes an unordered collection and makes its an ordered one.
- Let A be a list of n elements A_1, A_2, … A_n in memory.
- Sorting A refers to the operation of rearranging the contents of A
  - Ascending order
  - Descending order

SORTING

- In this lecture, we are going to look at three simple sorting techniques:
  - Bubble Sort
  - Selection Sort
  - Insertion Sort

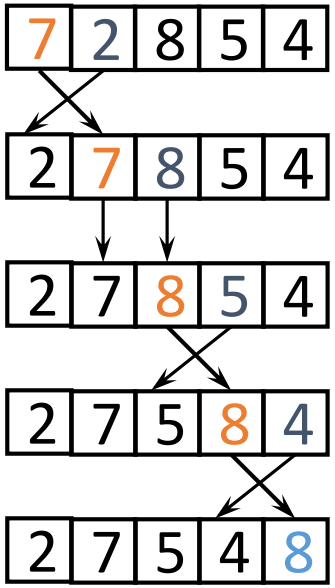
BUBBLE SORT

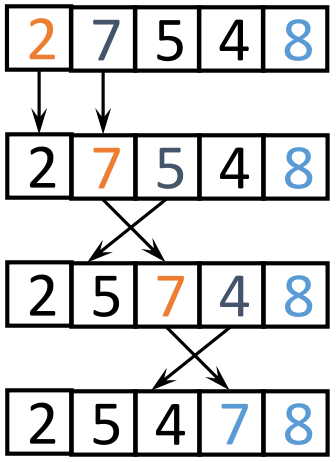
- Traverse a collection of elements.
- Move an element from its position to the front (if smallest) or to the end (if largest) in case of ascending order and vice versa.
- “Bubble” the value to the front or end using the operations
  - Pair-wise comparison
  - Swapping

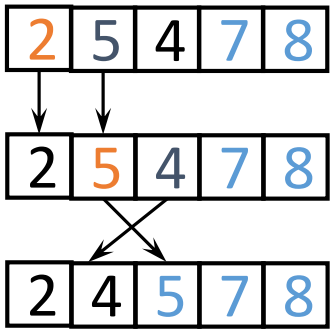
BUBBLE SORT

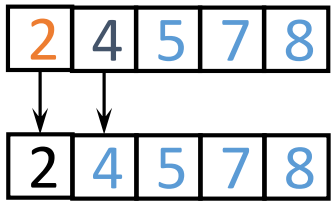
- Compare each element (except the last *one*) with its neighbor to the right.
  - If they are out of order, swap them
  - This puts the largest element at the very end
  - The last element is now in the correct and final place
- Compare each element (except the last *two*) with its neighbor to the right.
  - If they are out of order, swap them
  - This puts the second largest element next to last
  - The last two elements are now in their correct and final places
- Compare each element (except the last *three*) with its neighbor to the right
  - Continue as above until you have no unsorted elements on the left

BUBBLE SORT


7 2 8 5 4
2 7 8 5 4
2 7 8 5 4
2 7 5 8 4
2 7 5 4 8


2 7 5 4 8
2 7 5 4 8
2 5 7 4 8
2 5 4 7 8


2 5 4 7 8
2 5 4 7 8
2 4 5 7 8


2 4 5 7 8
2 4 5 7 8

BUBBLE SORT

```
void BubbleSort ( int arr[], int size )
{
    int temp;
    for ( int outer = size-1; outer > 0; outer-- )
    {
        for ( int inner = 0; inner < outer; inner++ )
        {
            if ( arr[inner] > arr[inner+1] )
            {
                temp = arr[inner];
                arr[inner] = arr[inner+1];
                arr[inner+1] = temp;
            }
        }
    }
}
```

ANALYSIS OF BUBBLE SORT

```
for ( int outer = size-1; outer > 0; outer-- ) {
    for ( int inner = 0; inner < outer; inner++ ) {
        if ( arr[inner] > arr[inner+1] ) {
            //code for swapping is skipped
    } } }
```

- Let size = arr.length = size of the array
- The outer loop is executed n-1 times (call it n, that's close enough)
- Each time the outer loop is executed, the inner loop is executed
  - Inner loop executes n-1 times at first, linearly dropping to just once
  - On average, inner loop executes about n/2 times for each execution of the outer loop
  - In the inner loop, the comparison is always done (constant time), the swap might be done (also constant time)
- Result is n * n/2 * k, that is, $O(n^2/2 + k) = O(n^2)$

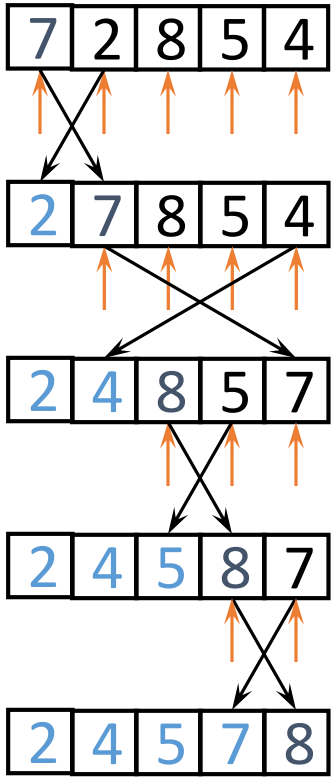
SELECTION SORT

- Define the entire array as the unsorted portion of the array.
- While the unsorted portion of the array has more than one element:
  - Find its largest element
  - Swap with last element (assuming their values are different)
  - Reduce the size of the unsorted portion of the array by 1.

SELECTION SORT

- Given an array of length n,
  - Search elements 0 through n-1 and select the smallest
    - Swap it with the element in location 0
  - Search elements 1 through n-1 and select the smallest
    - Swap it with the element in location 1
  - Search elements 2 through n-1 and select the smallest
    - Swap it with the element in location 2
  - Continue in this fashion until there's nothing left to search

SELECTION SORT




```
void SelectionSort ( int arr[], int size )
{
    int temp;
    int min;
    for ( int outer = 0; outer < size-1; outer++ )
    {
        min = outer;
        for ( int inner = outer+1; inner < size; inner++ )
        {
            if ( arr[inner] < arr[min] )
            {
                min = inner;
            }
        }
        temp = arr[outer];
        arr[outer] = arr[min];
        arr[min] = temp;
    }
}
```

ANALYSIS OF SELECTION SORT

- The Selection Sort might swap an array element with itself--this is harmless, and not worth checking for
- Analysis:
  - The outer loop executes $n-1$ times
  - The inner loop executes about $n/2$ times on average (from n to 2 times)
  - Work done in the inner loop is constant (swap two array elements)
  - Time required is roughly $(n-1)*(n/2)$
- You should recognize this as $O(n^2)$

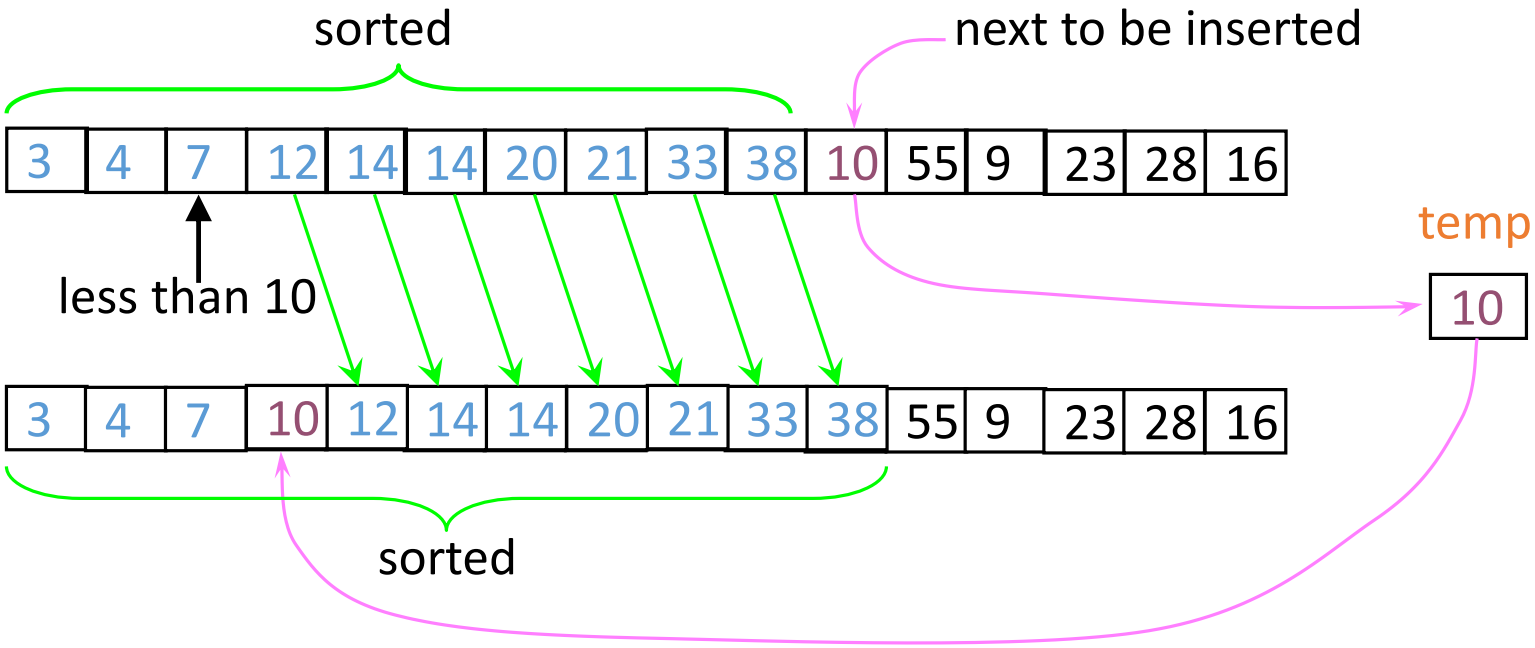
INSERTION SORT

- The list is divided into two parts: sorted and unsorted.
- In each pass, the following steps are performed:
  - First element of the unsorted part (i.e., sub-list) is picked up
  - Transferred to the sorted sub-list
  - Inserted at the appropriate place
- A list of n elements will take at most n-1 passes to sort the data

INSERTION SORT

```
void InsertionSort ( int arr[], int size )
{
    int temp;
    for ( int outer = 1; outer < size; outer++ )
    {
        temp = arr[outer];
        int inner = outer;
        while ( inner > 0 && arr[inner-1] >= temp )
        {
            arr[inner] = arr[inner-1];
            inner--;
        }
        arr[inner] = temp;
    }
}
```

INSERTION SORT



ANALYSIS OF INSERTION SORT

- We run once through the outer loop, inserting each of n elements; this is a factor of n
- On average, there are $n/2$ elements already sorted
  - The inner loop looks at (and moves) half of these
  - This gives a second factor of $n/2$
- Hence, the time required for an insertion sort of an array of n elements is proportional to $n^2/4$
- Discarding constants, we find that insertion sort is $O(n^2)$

CONCLUSION

- In this lecture we have studied:
  - Bubble Sort
  - Selection Sort
  - Insertion Sort

Question?