COMP 3711

Bubble Sort

The Sorting Problem

Input: An array $A[1 \dots n]$ of elements

[4 1 8 2 5]

Output: Array $A[1 \dots n]$ of elements in sorted order (ascending)

[1 2 4 5 8]

Bubble Sort

Input: An array $A[1 \dots n]$ of elements

Output: Array $A[1 \dots n]$ of elements in sorted order (ascending)

(*) Walk through all items from first to last
Compare each item to its successor
If they are in wrong order, swap them
If any item was swapped go to (*)

<u>Bubble-Sort(A):</u>
repeat
  $swapped \leftarrow$ **false**
  for $i \leftarrow 1$ **to** $n - 1$
    if $A[i] > A[i+1]$ **then**
      swap $A[i]$ **and** $A[i+1]$
      $swapped \leftarrow$ **true**
until not $swapped$

1st Pass: (4 1 8 2 5) → (1 4 8 2 5) → (1 4 8 2 5) → (1 4 2 8 5) → (1 4 2 5 8)
2nd Pass: (1 4 2 5 8) → (1 4 2 5 8) → (1 2 4 5 8) → (1 2 4 5 8)
3rd Pass: (1 2 4 5 8) → (1 2 4 5 8) → (1 2 4 5 8) → (1 2 4 5 8)

3rd Pass had no swaps, so terminate. Array is sorted

Correctness of bubble sort

Claim: When bubble sort terminates, the array must be sorted.

Proof: Trivial.
The algorithm terminates only if the last pass did not swap any pair, i.e.,

$$A[1] \le A[2],\ \ A[2] \le A[3],\ \ A[3] \le A[4],\ \ \dots,\ \ A[n-1] \le A[n].$$

That is

$$A[1] \le A[2] \le A[3] \le \cdots \le A[n-1] \le A[n].$$

SORTED!

Claim: Bubble sort terminates after at most $n-1$ passes.

Proof:

- After the 1st pass, the largest element must be at $A[n]$, and it will not be swapped any more.
- After the 2nd pass, the 2nd largest element must be at $A[n-1]$, and it will not be swapped any more.
- …
- After the $(n-1)\mathrm{st}$ pass, the 2nd smallest element must be at $A[2]$, and the smallest element must be at $A[1]$.

Running time of bubble sort

Claim: Bubble sort terminates after at most $n - 1$ passes.

Since each phase requires $O(n)$ comparisons and there are $O(n)$ passes
=> entire algorithm requires $O(n^2)$ comparisons

Tighter analysis: Suppose i < j and , in original array, A[i] > A[j].
We then say that the pair (i,j) form an INVERSION (pair) in the original array.

Example: In array (4 1 8 2 5), inversion pairs are
(4,1), (4,2), (8,2), (8,5)

Recall the execution of Bubble sort on this input:

1st Pass: (4 1 8 2 5) → (1 4 8 2 5) → (1 4 8 2 5) → (1 4 2 8 5) → (1 4 2 5 8))
2nd Pass: (1 4 2 5 8) → (1 4 2 5 8) → (1 2 4 5 8) → (1 2 4 5 8)
3rd Pass: (1 2 4 5 8) → (1 2 4 5 8) → (1 2 4 5 8) → (1 2 4 5 8)

The swaps made by bubble sort were EXACTLY the inversion pairs

Running time of bubble sort

Claim: Bubble sort terminates after at most $n-1$ passes.

Since each phase requires $O(n)$ comparisons and there are $O(n)$ passes
=> entire algorithm requires $O(n^2)$ comparisons

Tighter analysis: Suppose i < j and , in original array, A[i] > A[j].
We then say that the pair (i,j) form an INVERSION (pair) in the original array.

A swap can only be caused by an original inversion pair (WHY).
Also, every inversion pair will at some point be compared and then cause a swap.

$\Rightarrow$ The number of swaps performed by bubble sort is exactly equal to the number of inversion pairs in the original array A [ ]. This is called the Inversion Number of A

$\Rightarrow$ Number of comparisons performed by bubble sort $\geq$ Inversion # of A

If the Array is in reverse sorted order then EVERY pair is an Inversion Pair so
inversion number = $\binom{n}{2} = \Theta(n^2)$.
(This is a worst case for bubble sort)

Worst case running time of Bubble Sort is $\Theta(n^2)$.