Sorting (III): Bubble Sort

CSCD 300 – Data Structures

Eastern Washington University

Goal

We will learn the mechanism of the Bubble Sort algorithm and then analyze its time complexity in the best as well as in the worst case.

Outline

Bubble sort

Basic idea

- Scan through the sequence from the left to the right, exchange any two pair of neighboring numbers if they are not in ascending order.
- Do such scanning repeatedly until there is no exchange happened in the previous pass of scanning: the sequence is sorted already.

An example: sort the sequence 4 2 3 1 using Bubble sort

The first pass: $\boxed{4\ 2}\ 3\ 1 \rightarrow 2\ \boxed{4\ 3}\ 1 \rightarrow 2\ 3\ \boxed{4\ 1} \rightarrow 2\ 3\ 1\ 4$

The second pass: $\boxed{2\ 3}\ 1\ 4 \rightarrow 2\ \boxed{3\ 1}\ 4 \rightarrow 2\ 1\ \boxed{3\ 4} \rightarrow 2\ 1\ 3\ 4$

The third pass: $\boxed{2\ 1}\ 3\ 4 \rightarrow 1\ \boxed{2\ 3}\ 4 \rightarrow 1\ 2\ \boxed{3\ 4} \rightarrow 1\ 2\ 3\ 4$

The fourth pass: $\boxed{1\ 2}\ 3\ 4 \rightarrow 1\ \boxed{2\ 3}\ 4 \rightarrow 1\ 2\ \boxed{3\ 4} \rightarrow 1\ 2\ 3\ 4$

There are no exchanges happend in the fourth pass, so the Bubble sort stops and the sequence is sorted.

Pseudocode [1]

BubbleSort1(A)

Input: An array $A[0 \ldots n-1]$ of n numbers
Output: The sorted A.

```
exchanged ← true; /* exchanged = true, if exchanges have
   happened in the previous pass; false, otherwise          */

while exchanged = true do
    exchanged ← false
    for i = 0 ... n − 2 do
        if A[i] > A[i + 1] then
            exchange(A[i], A[i + 1])
            exchanged ← true;
```

Can we make it faster ? Yes !

[1] We use 0-based indexing.

Do we really need to scan to the end of the whole sequence for each pass ? No !

Observations

- After the first pass of scanning, the largest number must have already occupied the right most position $A[n-1]$, so in the second pass, we only need to compare the numbers in $A[0 \ldots n-2]$.
- After the second pass of scanning, the second largest number must have already occupied the position $A[n-2]$, so in the third pass, we only need to compare the numbers in $A[0 \ldots n-3]$.
- ...
- After the ith pass of scanning, the ith largest number must have already occupied the position $A[n-i]$, so in the $(i+1)$th pass, we only need to compare the numbers in $A[0 \ldots n-i-1]$.

This observation speeds up the sorting process and leads to the improved version of Bubble sort.

An improved version of Bubble sort

BubbleSort2(A)

Input: An array $A[0 \ldots n-1]$ of n numbers
Output: The sorted A.

```
exchanged ← true; /* exchanged = 1 if exchanges have happened
    in the previous pass; false, otherwise                    */
right ← n-2
while exchanged = true do
    exchanged ← false
    for i = 0 ... right do
        if A[i] > A[i+1] then
            exchange(A[i], A[i+1])
            exchanged ← true;
    right ← right - 1
```

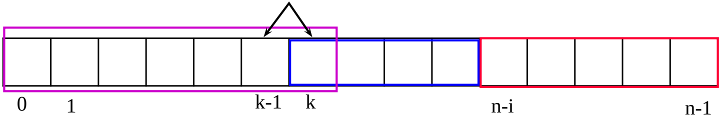
Can we make it even better ? Yes !

Observations

- We already know that when we do the $(i+1)$th pass, the pass can stop at $A[n-i-1]$. (Page 7)
- In fact, in the $(i+1)$th pass, we even do not need to reach $A[n-i-1]$, if we know the location of the rightmost flipping in the ith pass. $\Longrightarrow$ The $(i+1)$th pass can just stop whenever it meets that location, which of course will not be on the right of $A[n-i-1]$.

Next, we will prove this observation.

Proof of the observation



The array A at the end of the ith travel

- We know at the end of the ith travel, $A[n-i, ..., n-1]$ (the red area) have had their final numbers safely landed. (Page 7).
- Suppose the rightmost flip in the ith travel is between $A[k-1]$ and $A[k]$, for some $k \leq n-1$. We claim: if $k < n-i$, then $A[k, \ldots, n-i-1]$ (the blue area) also have already had their numbers safely landed.
- The above claim is correct because:
  - $A[k]$ is the largest among $A[0 \ldots k]$ (the pink area) after the ith travel.
  - All numbers in the blue area are in increasing order already, because $A[k-1, k]$ is the rightmost flip.

Next, we will use the observation to further speed up the Bubble Sort in practice and leads to a better version of Bubble sort.

An even better version of Bubble sort

BubbleSort3(A)

Input: An array $A[0 \dots n-1]$ of n numbers
Output: The sorted A.

```
right = n − 2
while right ≥ 0 do
    new_right = −1
    for i = 0 ... right do
        if A[i] > A[i + 1] then
            exchange(A[i], A[i + 1])
            new_right = i − 1;
    right = new_right
```

The time complexity of Bubble Sort

The best case: $O(n)$

The best case is when the input sequence is already sorted. In that case, we only need one pass of scanning, which obviously takes $O(n)$ time.

The worst case: $O(n^2)$

The worst case is when the input sequence is completely in descending order. In that case, we will need $n-1$ passes:

- The first pass: $n-1$ exchanges.
- The second pass: $n-2$ exchanges.
- ...

Each exchange takes constant amount of time, denoted as c, so the total time cost is:

$$((n-1)+(n-2)+\ldots+2+1)c = \frac{n(n-1)c}{2} = O(n^2)$$

Question

How do you use Bubble sort if the data sequence is saved in a singly linked list ?