

SORTING

ALGORITHMS & DATA STRUCTURES – I

COMP 221

Sorting

Sorting is one of the most frequently performed computing tasks.

It is often required to keep data in order.

- Sorting makes many questions about the array easier to answer. In a sorted array, searching can be done efficiently (dictionaries, telephone books, class lists are sorted).
- Often, sorting is required as an auxiliary step for more complex algorithms.
- Sorting has been studied intensively and many algorithms have been developed.

Selection sort

We start with a straightforward algorithm called Selection Sort. For simplicity, we assume that the data items are numbers and they should be sequenced in the ascending order.

An array $A[0..n-1]$ of n elements $A[0], A[1], \dots, A[n-1]$ is sequenced in the ascending order if

$$A[0] \leq A[1] \leq A[2] \leq \dots \leq A[n-1].$$

The general idea of the algorithm can be described as follows.

- ❑ We search the array for the smallest element and swap that element with the first one.
- ❑ Then we search for the second smallest, and we swap that element with the second one.
- ❑ These steps are repeated until we have sorted all of the data.

Example

Suppose that the following list is to be sorted into ascending order:

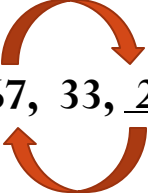
67, 33, 21, 84, 49, 50, 75

We scan the list to locate the smallest element and find it in position 3:

67, 33, 21, 84, 49, 50, 75

We interchange this element with the first element and thus properly position the smallest element at the beginning of the list:


67, 33, 21, 84, 49, 50, 75

A diagram consisting of two curved orange arrows. The top arrow starts above the first element '67' and points to the third element '21'. The bottom arrow starts above the third element '21' and points to the first element '67', illustrating the swap between the first and third elements.

We now scan the sublist consisting of the elements from position 2 on to find the smallest element

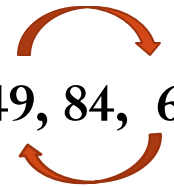
21, | 33, 67, 84, 49, 50, 75

and exchange it with the second element (itself in this case), thus properly positioning the next-to-smallest element in position 2:


21, 33, 67, 84, 49, 50, 75

We continue in this manner, locating the smallest element in the sublist of elements from position 3 on and interchanging it with the third element, then properly positioning the smallest element in the sublist of elements from position 4 on, and so on until we eventually do this for the sublist consisting of the last two elements:

21, 33, | 49, 84, 67, 50, 75




21, 33, 49, | 50, 67, 84, 75



21, 33, 49, 50, | 67, 84, 75



21, 33, 49, 50, 67, | 75, 84



Positioning the smallest element in this last sublist obviously also positions the last element correctly and thus completes the sort.

Selection sort Algorithm

For $i = 1$ to $n - 1$ do the following:

// Find the smallest element in the sublist $x[i] \dots x[n]$.

a. Set $smallPos = i$ and $smallest = x[smallPos]$.

b. For $j = i + 1$ to $n - 1$ do the following:

If $x[j] < smallest$: His maller element found

Set $smallPos = j$ and $smallest = x[smallPos]$.

End for

// Now interchange $smallest$ with $x[i]$, first element of this sublist.

c. Set $x[smallPos] = x[i]$ and $x[i] = smallest$.

End for.

The time complexity of the selection sort is $O(n^2)$.

Bubble Sort

With the selection sort, we make one exchange at the end of one pass.

- The bubble sort improves the performance by making more than one exchange during its pass.**
- By making multiple exchanges, we will be able to move more elements toward their correct positions using the same number of comparisons as the selection sort makes.**
- The key idea of the bubble sort is to make pair-wise comparisons and exchange the positions of the pair if they are out of order.**

Example

To illustrate bubble sort, consider again the list

67, 33, 21 , 84, 49, 50, 75

On the first pass, we compare the first two elements, 67 and 33, and interchange them because they are out of order:

67, 33, 21 , 84, 49, 50, 75



33, 67, 21 , 84, 49, 50, 75

Now we compare the second and third elements, 67 and 21 and interchange them:

33, 67, 21 , 84, 49, 50, 75



33, 21, 67 , 84, 49, 50, 75

Next we compare 67 and 84 but do not interchange them because they are already in the correct order:

Next, 84 and 49 are compared and interchanged:

33, 21, 67 , 84, 49, 50, 75



33, 21, 67 , 49, 84, 50, 75

Then 84 and 50 are compared and interchanged:

33, 21, 67 , 49, 84, 50, 75



33, 21, 67 , 49, 50, 84, 75

Finally 84 and 75 are compared and interchanged:

33, 21, 67 , 49, 50, 84, 75



33, 21, 67 , 49, 50, 75, 84

The first pass through the list is now complete.

We are guaranteed that on this pass, the largest element in the list will "sink" to the end of the list, since it will obviously be moved past all smaller elements:

33, 21, 67 , 49, 50, 75, 84

We now scan the list again, but this time we leave out the last item because it is already in its proper position.

33, 21, 67, 49, 50, 75, 84

21, 33, 67, 49, 50, 75, 84

21, 33, 67, 49, 50, 75, 84

21, 33, 49, 67, 50, 75, 84

21, 33, 49, 50, 67, 75, 84

21, 33, 49, 50, 67, 75, 84

21, 33, 49, 50, 67, 75, 84

Bubble Sort Algorithm

1. Initialize *numCompares* to $n-1$. // number of comparisons on next pass
 2. While *numCompares* not equal to 0, do the following:
 - a. Set *last* = 1. // location of last element involved in a swap
 - b. For $i = 1$ to *numCompares*:
 - If $X_i > X_{i+1}$
 - Swap x_i and x_{i+1} and set *last* = i .
 - c. Set *numCompares* = *last* - 1.
- End while.

The time complexity of the Bubble sort is $O(n^2)$.

Insertion Sort

Reduces number of key comparisons made in selection sort

Can be applied to both arrays and linked lists.

Sorts list by

- Finding first unsorted element in list**
- Moving it to its proper position**

Insertion sorts are based on the idea of repeatedly inserting a new element into a list of already sorted elements so that the resulting list is still sorted.

The method used is similar to that used by a card player when putting cards into order as they are dealt.

Example

67, 33, 21, 84, 49, 50, 75

Initial sorted sublist of 1 element

33, 67, 21, 84, 49, 50, 75

Insert 33 to get 2-element sorted sublist

21, 33, 67, 84, 49, 50, 75

Insert 21 to get 3-element sorted sublist

21, 33, 67, 84, 49, 50, 75

Insert 84 to get 4-element sorted sublist

21, 33, 49, 67, 84, 50, 75

Insert 49 to get 5-element sorted sublist

21, 33, 49, 50, 67, 84, 75

Insert 50 to get 6-element sorted sublist

21, 33, 49, 50, 67, 75, 84

Insert 75 to get 7-element sorted sublist

Insertion Sort Algorithm

For $i = 2$ to n do the following:

// Insert $x[i]$ into its proper position among $x[1], \dots, x[i - 1]$

a. Set $nextElement = x[i]$ and $x[0] = nextElement$.

b. Set $j = i$.

c. While $nextElement < x[j - 1]$ do the following:

// Shift element to the right to open a spot

Set $x[j]$ equal to $x[j - 1]$ and decrement j by 1.

End while.

// Now drop $nextElement$ into the open spot.

d. Set $x[j]$ equal to $nextElement$.

End for.

The time complexity of the selection sort is $O(n^2)$

THANK YOU