Concepts of Algorithms
CS-211
Chapter 4
Sorting (Insertion Sort)

Dr. Fethi Fkih
College of Computer
Qassim University

Sorting

- Of all the algorithmic problems that computer scientists have studied, the one with the broadest practical impact is certainly the ***sorting*** *problem*, which is the problem of arranging the elements of an array in some order i.e. (Ascending or Descending) .

  For Example

  Input: sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.
  Output: permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ Such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

  Example:

  Input: 8 2 4 9 3 6
  Output: 2 3 4 6 8 9 (Ascending Order)
  Output: 9 8 6 4 3 2 (Descending Order)

Sorting (Continued)

- The sorting problem comes up, for example, in alphabetizing a telephone directory, arranging library records by catalogue number, and organizing a bulk mailing by ZIP code.
- There are many algorithms that one can use to sort an array. Because these algorithms vary extremely in their efficiency, it is critical to choose a good algorithm, particularly if the application needs to work with large arrays.

Insertion Sort

Of the many sorting algorithms, one which we describe now is the ***Insertion Sort***, which appears in the text like this:

Suppose an array ***A*** with ***N*** elements ***A***[0], ***A***[1], ... ***A***[N-1] is in the memory. The insertion sort algorithm scan ***A*** from ***A***[0] to ***A***[N-1], inserting each element ***A***[k] into its proper position in the previously sorting sub array A[0], A[1], . . .A[N-1] That is:

Pass 1: A[1] is inserted either before or after A[0] so that: A[0], A[1] are sorted.

Pass 2: A[2] is inserted into its proper place in A[0], A[1], so that A[0], A[1], A[2] are sorted.

Pass 3: A[3] is inserted into its proper place in A[0], A[1], A[2] so that: A[0], A[1], A[2], A[3] are sorted.

. . . . . .

Pass N-1: A[N-1] is inserted into its proper place in A[0], A[1], ... A[N-1] so that: A[0], A[1], ... A[N-1] are sorted.

Insertion Sort

Pass	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K=1	80	77	33	44	11	88	22	66	55
K=2	77	80	33	44	11	88	22	66	55
K=3	33	77	80	44	11	88	22	66	55
K=4	33	44	77	80	11	88	22	66	55
K=5	11	33	44	77	80	88	22	66	55
K=6	11	33	44	77	80	88	22	66	55
K=7	11	22	33	44	77	80	88	66	55
K=8	11	22	33	44	66	77	80	88	55
	11	22	33	44	55	66	77	80	88
Sorted	11	22	33	44	55	66	77	80	88

The algorithm is simplified if there always is an element A[i] such that A[i] ≤ A[K]; other *wise we must constantly check to see if we are comparing A[K] with A[0].*

Insertion Sort

Pseudocode

```
INSERTION-SORT (A, N)        ▷ A[0 . . N-1]
    for K = 1 to N-1
        Temp = A[ K]
            i = K –1
            while i >= 0 and A[i ] > Temp
                    A[i +1] = A[i]
                        i = i–1
            A[i+1] = Temp
```

Algorithm”

Algorithm: (INSERTION SORT) INSERTION (A, N)
[Where A is an array and N is the Size of values in the array]

1. Repeat steps 2 to 4 for K=1,2,3, ... N-1:
2. Set TEMP = A[K] and i =K-1.
3. Repeat while i >= 0 and TEMP < A[i]
   a) Set A[i+1] = A[i]. [Moves element forward.]
   b) Set i = i -1.
   [End of loop.]
4. Set A[i+1] =TEMP. [Insert element in proper place.]
   [End of Step 2 loop.]
5 Return.

Insertion Sort (Continued)

- This sorting algorithm is frequently used when ***N*** is small. There remains only the problem of deciding how to insert A[K] in its proper place in the subarray A[0], A[1], ... A[K-1]. This can be accomplished by comparing A[K] with A[K-1], comparing A[K] with A[K-2], comparing A[K] with A[K-3], and so on, until first meeting an element A[i] (where ***i*** start from k-1) such that A[i] $\leq$ A[K]. then each of elements A[K-1], A[K-2], ... A[i+1] is moved forward one location, and A[K] is then inserted in the ***i***+1 position in the array.

The Execution Time of Algorithm

	Cost	Times
INSERTION-SORT (A, N) ▷ $A[0 \ldots N-1]$		
for $K = 1$ to $N-1$	c_1	n
$Temp = A[K]$	c_2	n-1
$i = K-1$	c_3	n-1
while $i >= 0$ and $A[i] > Temp$	c_4	$\sum_{k=1}^{n-1}$ Tk
A[i +1] = A[i]	c_5	$\sum_{k=1}^{n-1}$ T'k = Tk-1
$i = i-1$	c_6	$\sum_{k=1}^{n-1}$ T'k
$A[i+1] = Temp$	c_7	n-1

Total cost = $nc_1+(n-1)(c2+c3-c5-c6+c7)+(c4+c5+c6)\sum Tk$

$= n(c1+c2+c3-c5-c6+c7)+(c4+c5+c6)\sum Tk)-(c2+c3-c5-c6+c7)$

$= a\ n +b \sum Tk +c$

Tk — Number of comparisons

T'k — Number of shifting
= Number of Comparisons -1
= Tk -1

The Execution Time of Algorithm

Pass	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	Tk
K=1	80	77	33	44	11	88	22	66	55	1
K=2	77	80	33	44	11	88	22	66	55	2
K=3	33	77	80	44	11	88	22	66	55	3
K=4	33	44	77	80	11	88	22	66	55	4
K=5	11	33	44	77	80	88	22	66	55	1
K=6	11	33	44	77	80	88	22	66	55	6
K=7	11	22	33	44	77	80	88	66	55	4
K=8	11	22	33	44	66	77	80	88	55	5
Sotred	11	22	33	44	55	66	77	80	88	

Tk is the total number of comparisons made in each pass

Which is needed for analysis purpose

Kinds of analyses

Total cost = a n +b $\sum$Tk +c

Best-case:

- Elements are already in Sorted Order: Tk = 1 ➔ see*
  running time = f(n), i.e. Linear time. ➔ O(n)

Worst-case: (usually)

- Elements are sorted in inverse order: Tk = K ➔ see**
  running time = f(n^2) i.e. quadratic time. Maximum time of algorithm on any input of size n. ➔ O(n^2)

* $\sum_{k=1}^{n-1} 1 = n - 1$

** $\sum_{k=1}^{n-1} k = {}^{n(n-1)}/_{2}$

END