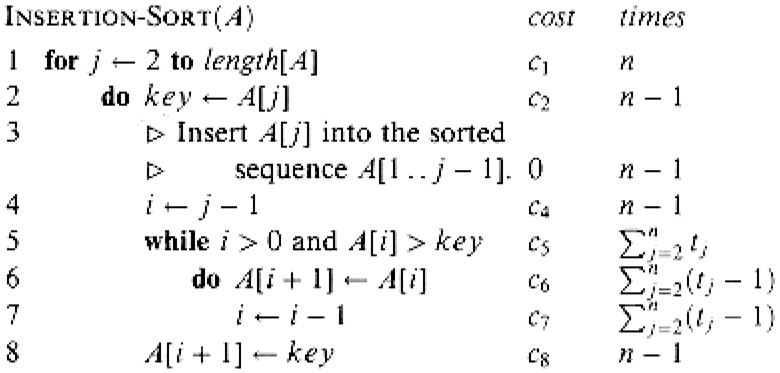
Example: A bank office records customers in an array A[1...n] sorted by account numbers. However, a bank officer accidentally shifts the array elements k < n positions to the right circularly. Note that k is not necessarily constant (e.g. might be a function of n). For example, let A = [9, 12, 17, 21, 33, 41]. When k = 2 positions are shifted to the right, A becomes [33, 41, 9, 12, 17, 21]. However, if k = 4, A becomes [17, 21, 33, 41, 9, 12]. The officer realizes the problem and wants to fix it by sorting the shifted version of the array with Insertion-sort. Find the complexity of the Insertion sort for such an input scenario. Generate random arrays of size n = 216 and shift the array for k = 1, n/16, 2n/16, 3n/16, 4n/16, 5n/16, 6n/16, 7n/16, 8n/16, 9n/16, 10n/16, 11n/16, 12n/16, 13n/16, 14n/16, 15n/16, n − 1 positions. For each k, measure the running times to sort the shifted array and organize the calculated times into a table. Discuss the results and compare the behaviors of the actual running times and the theoretical complexity.

Insertion Sort Algo (Pseudocode)

```
for j <- 2 to length[A]
    do key <- A[j]
     Insert A[j] into the sorted sequence A[1 . . j - 1].
     i <- j - 1
     while i > 0 and A[i] > key
        do A[i + 1] <- A[i]
           i <- i - 1
     A[i + 1] <- key
```

E.g.#1 Seq: {4, 2, 1, 6, 4} (index starts from 1)

1. j= 2: 4 2 1 6 4 => 4 **2** 1 6 4=> i = 1 & key = 2 => 4 **4** 1 6 4 => i = 0 => 2 **4** 1 6 4
2. j = 3: 2 4 **1** 6 4 => i = 2 & key = 1 => 2 4 **4** 6 4 => i = 1 & A[1] > key => 2 2 **4** 6 4 => i = 0 => 1 2 **4** 6 4
3. j = 4: 1 2 4 **6** 4 => i = 3 & key = 6 & A[3] < key => 1 2 4 **6** 4
4. j = 5: 1 2 4 6 **4** => i = 4 & key = 4 => 1 2 4 6 **6** => i = 3 & A[3] == key => 1 2 4 4 **6** **(sorted)**



INSERTION-SORT(A)		*cost*	*times*
1	**for** $j \leftarrow 2$ **to** *length*$[A]$	c_1	n
2	**do** *key* $\leftarrow A[j]$	c_2	$n-1$
3	▷ Insert $A[j]$ into the sorted ▷ sequence $A[1..j-1]$.	0	$n-1$
4	$i \leftarrow j-1$	c_4	$n-1$
5	**while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6	**do** $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
7	$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$
8	$A[i+1] \leftarrow key$	c_8	$n-1$

E.g.#2 (above): {9, 12, 17, 21, 33, 41}

1. j = 2: 9 **12** 17 21 33 41 => i = 1 & key = 12 => A[i]= 9 < key = no change
2. j = 3: 9 12 **17** 21 33 41 => i = 2 & key = 17 => A[i] = 12 < key = no change
3. … j = n: no change

   Running cost with sorted list only n times with the outer loop. In a sorted list, the algorithm never enters the inner loop.

   However, in the case of the mistaken shift: k = 4, [17, 21, 33, 41, 9, 12]

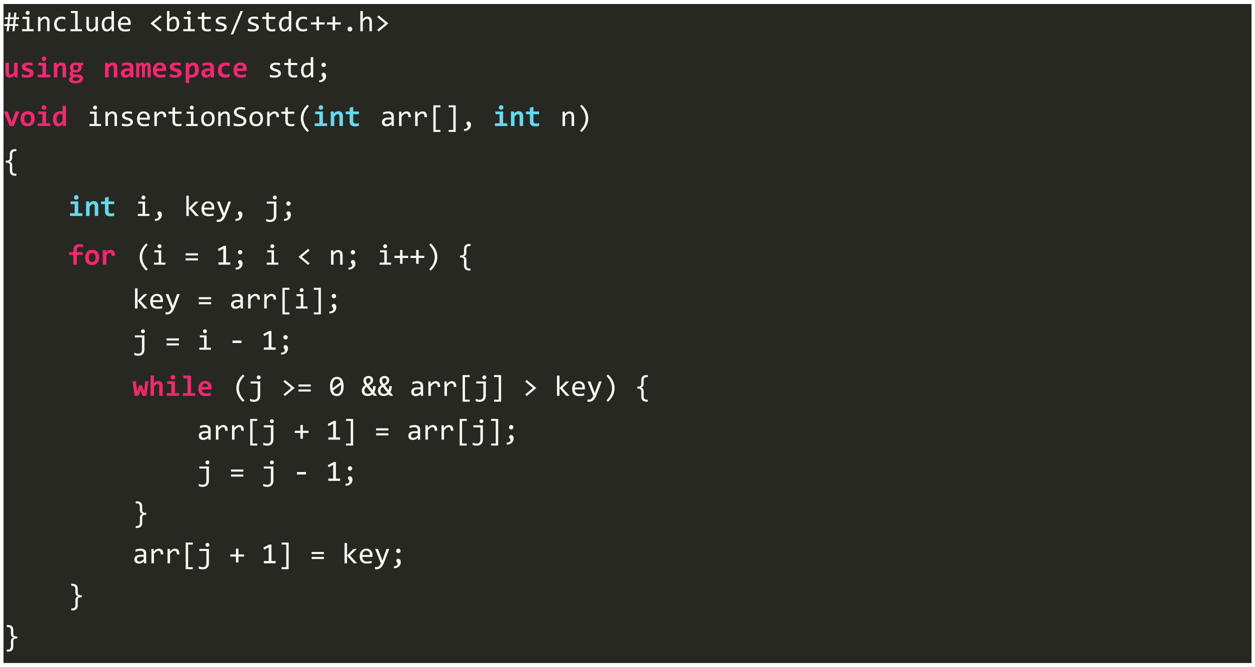
   1. j = 2: no change
   2. j = 3: no change
   3. j = 4: no change
   4. j = 5: 17 21 33 41 **9** 12 => i = 4 & key = 9 & A[i] > key => 17 21 33 41 **41** 12 => i=3 & A[i] > key => 17 21 33 33 **41** 12 => i = 2 & A[i]> key => 17 21 21 33 **41** 12 => i=1 & A[i] > key => 17 17 21 33 **41** 12 => i=0 => 9 17 21 33 **41** 12
   5. j = 6: 9 17 21 33 41 **12** => i=5 & key= 12 & A[i] > key => 9 17 21 33 41 **41** => i=4 & A[i] > key => 9 17 21 33 33 **41** => i=3 & A[i]>key => 9 17 21 21 33 **41** => i=2 & A[i]> key => 9 17 17 21 33 **41** => i=1 & A[i]=9 < key=12 => 9 12 17 21 33 **41**

   c = constant time, in asymptotic analysis it is equivalent to 1 unit and discarded later
   n = the variable of array size
   k = the variable of the number of shifts

$$\sum_{j=2}^{j=n-k} c + \sum_{j=n-k}^{j=n} n = (n - k - 2)c + n(n - (n - k))$$

$$= (n - k - 2)c + n^2 - n^2 + nk = nc - kc - 2c + nk$$

$$= nc - kc + nk \leq nk + c2 \equiv O(nk)$$



```
#include <bits/stdc++.h>
using namespace std;
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```