In this problem, you have to implement a variation of Insertion Sort as described below.

Suppose X is an array of N positive integers to be sorted. In this scheme, another array Y is used to store the sorted integers. This array is to be viewed as a circular array, ie. index (N-1)+1 = index 0 and index 0-1 = index N-1. The sorted integers get stored in a circular manner in Y, ie, it is possible that the index of the smallest integer may be greater than the index of the largest integer. Eg. 6 8 \_ \_ \_ 1 2 4 5 is a view of the array Y sometime into the algorithm. ’\_’ indicates unused locations of the array.

Smallest integer 1 is at index 5, largest integer 8 is at index 1. So the sorted array in Y is to be generated by printing the contents from index 5 to 1, assuming the array wraps around at the end, ie. after index 8, the next index is 0.

Assume that h holds the index of the smallest integer in Y and t holds the index of the largest integer in Y.  
Initially,

1. h = t = 0
2. Y[0] = X[0] ie. the first integer in X[] is copied as the first integer in Y[].
3. All other elements in Y[] are initialised to a dummy value -1.

The rest of the integers in X[] are now inserted one by one into Y[] such that Y[] always contains a sorted list of integers, with the smallest integer at index h and the largest at index t. This is done in the following manner:

Let I be the next integer from X[] to be inserted into Y[]. Scan the array Y downwards from index h (with wrap-around at the end) till index t and find out the place in Y[] where I has to fit in. If I fits in at either end of the list, then insert it at the appropriate place in Y[]. Modify either t or h as appropriate to indicate the new array structure; ie. either t is incremented or h is decremented (with wrap-around).

If I fits in somewhere in the middle of the list, then I should be inserted by shifting all the S smaller integers one place to the left or by shifting all the L larger integers one place to the right, depending on the number of integers to be shifted. That is, if S < L, the smaller integers should be shifted one place to the left and if S >= L, the larger integers should be shifted one place to the right. Again either h or t should be modified appropriately.

**Example**

Integers to be sorted X[]: 25 57 37 48 12 92 86 33  
Contents of Y[] after inserting each integer from X[]:  
25 –1 –1 –1 –1 –1 –1 –1 Initially (t=0, h=0)  
25 57 –1 –1 –1 –1 –1 –1 57 fits in at end (t=1)  
25 37 57 –1 –1 –1 –1 –1 37 fits in middle, S=1, L=1, so shift 57 right. (t=2)  
25 37 48 57 –1 –1 –1 –1 48 fist in middle, S=2, L=1, So shift 57 right. (t=3)  
25 37 48 57 –1 –1 –1 12 12 fits in at beginning, circular property, (h=8, t=3)  
25 37 48 57 92 –1 –1 12 92 fits in at end (t=4).  
25 37 48 57 86 92 –1 12 86 fits in middle, S=5, L=1, so shift 92 right, (t=5).  
33 37 48 57 86 92 12 25 33 fits in middle, S=2, L=5, so shift 12, 25 left (h=7, t=5).

#include <stdio.h>

#define PREV(x,y) ( ( (((x)-1) < 0) ? (y)-1 : (x)-1 ) )

#define SUCC(x,y) ( ( (((x)+1) >= (y)) ? 0 : (x)+1 ) )

int main ()

{

int N;

int x[100];

int y[100];

int f,g;

int h,t;

int k,p;

int s,l;

int hp1, hp2;

int tp1, tp2;

int a,b;

scanf ("%d", &N);

for (f = 0; f < N; f++)

scanf ("%d", &x[f]);

h = t = 0;

for (b = 0; b < N; b++)

y[b] = -1;

y[0] = x[0];

for (b = 0; b < N; b++)

printf ("%d ", y[b]);

printf ("\n");

for (f = 1; f < N; f++)

{

k = x[f];

p = h;

s = 0;

for (g = 0; g < f; g++)

{

if ( k < y[p] )

break;

p = SUCC(p, N);

s++;

}

if ( g == 0 ) /\* k to be inserted in begining \*/

{

h = PREV(h, N);

y[h] = k;

}

else

if ( g == f ) /\* k to be inserted at end \*/

{

t = SUCC(t, N);

y[t] = k;

}

else /\* k to be inserted in middle \*/

{

l = f - s;

if (s < l) /\* shift left \*/

{

hp2 = h;

h = PREV(h,N);

hp1 = h;

for (a = 1; a <= s; a++)

{

y[hp1] = y[hp2];

hp1 = SUCC(hp1, N);

hp2 = SUCC(hp2, N);

}

y[hp1] = k;

}

else /\* shift right \*/

{

tp2 = t;

t = SUCC (t, N);

tp1 = t;

for (a = 1; a <= l; a++)

{

y[tp1] = y[tp2];

tp1 = PREV(tp1, N);

tp2 = PREV(tp2, N);

}

y[tp1] = k;

}

}

for (b = 0; b < N; b++)

printf ("%d ", y[b]);

printf ("\n");

}

return 0;

}