// An iterative implementation of quick sort

#include <stdio.h>

// A utility function to swap two elements

void swap ( int\* a, int\* b )

{

int t = \*a;

\*a = \*b;

\*b = t;

}

/\* This function is same in both iterative and recursive\*/

int partition (int arr[], int l, int h)

{

int x = arr[h];

int i = (l - 1);

for (int j = l; j <= h- 1; j++)

{

if (arr[j] <= x)

{

i++;

swap (&arr[i], &arr[j]);

}

}

swap (&arr[i + 1], &arr[h]);

return (i + 1);

}

/\* A[] --> Array to be sorted,

l --> Starting index,

h --> Ending index \*/

void quickSortIterative (int arr[], int l, int h)

{

// Create an auxiliary stack

int stack[ h - l + 1 ];

// initialize top of stack

int top = -1;

// push initial values of l and h to stack

stack[ ++top ] = l;

stack[ ++top ] = h;

// Keep popping from stack while is not empty

while ( top >= 0 )

{

// Pop h and l

h = stack[ top-- ];

l = stack[ top-- ];

// Set pivot element at its correct position

// in sorted array

int p = partition( arr, l, h );

// If there are elements on left side of pivot,

// then push left side to stack

if ( p-1 > l )

{

stack[ ++top ] = l;

stack[ ++top ] = p - 1;

}

// If there are elements on right side of pivot,

// then push right side to stack

if ( p+1 < h )

{

stack[ ++top ] = p + 1;

stack[ ++top ] = h;

}

}

}

// A utility function to print contents of arr

void printArr( int arr[], int n )

{

int i;

for ( i = 0; i < n; ++i )

printf( "%d ", arr[i] );

}

// Driver program to test above functions

int main()

{

int arr[] = {4, 3, 5, 2, 1, 3, 2, 3};

int n = sizeof( arr ) / sizeof( \*arr );

quickSortIterative( arr, 0, n - 1 );

printArr( arr, n );

return 0;

}