```
# File:    bubble.c
#
# Purpose: Use bubble sort to sort a list of ints.
#
# Compile: gcc -g -Wall -o bubble bubble.c
# Usage:   bubble <n> <gli>
#            n:   number of elements in list
#           'g':  generate list using a random number generator
#           'i':  user input list
#
# Args:                   rdi = a
#                         rsi = n
#
#


     .section .text

     .global bubble_sort


Bubble_sort:
                        push   %rbp
                        mov    %rsp,  %rbp
                        sub    $16,    %rsp                  # Make space to put i, n on the
stack

                        # Array a = %rdi
                        # NUmber of elements list_length = %r8 = n = %rsi. Save on the
stack
                        mov    %rsi,   %r8
                        mov    %r8,    8(%rsp)

                        # Current element in subscript i = %r9
                        mov    $0,     %r9
                        mov    %r9, 0(%rsp)                  # Store i = r9 on stack

loop_tst1:
                        cmp    $2,     %r8                   # Is list_length = n >= 2?
                        jge    loop_tst2                     # If list_length = n >= 2, jump to
the second loop
                        jmp    done_loop1                    # If list_length = n < 2, we're done

loop_tst2:
                        sub    $1,     %r8                   # Put list_length-1 in r8
                        cmp    %r9,    %r8                   # If i = r9 >= list_length-1 = r8?
                        jge    done_loop2                    # If i = r9 >= list_length-1 = r8,
jump to done_loop2

                        # Put a[i] in r10
                        mov     0(%rdi, %r9, 8), %r10       # a[i] is located at a + i*8 = rdi +
r9*8 = r10

                        # Put a[i+1] in r11
                        add     $1,     %r9                 # r9 = i+1
                        mov     0(%rdi, %r9, 8), %r11       # a[i+1] is located at a + (i+1)*8 =
rdi + r9*8 = r11

                        cmp     %r11,  %r10                 # If a[i] = r10 <= a[i+1] = r11?
                        jle     done_if                     # If a[i] = r10 <= a[i+1] = r11,
jump to done_if


                        mov     0(%rsp), %r9                # Reteive r9 = i
                        imul    $8,     %r9                 # r9 = r9*8
                        add     %rdi,   %r9                 # Get the absolute address of
a[i]: r9 = rdi + i*8
                        mov     %r9,    %rdi                # Put the absolute address of a[i]
in the first arg

                        mov     0(%rsp), %r9                # Reteive r9 = i
                        add     $1,     %r9                 # i = i+1
                        imul    $8,     %r9                 # r9 = (i+1)*8
                        add     %rdi,   %r9                 # Get the absolute address of a[i
+1]: r9 = rdi +(i+1)*8
                        mov     %r9,    %rsi                # Put the absolute address of a[i
+1] in the second arg

                        call    Swap

                        # In case r8 (= n)or r9 (= i) has been modified . . .
                        mov     0(%rsp), %r9
                        mov     8(%rsp), %r8

                        add     $1,     %r9                 # i++
                        jmp     loop_tst2

done_if:
                        mov     0(%rsp), %r9                # Reteive i = r9
                        add     $1,     %r9                 # i++
                        jmp     loop_tst2

done_loop2:
                        mov     8(%rsp), %r8                # Reteive list_length = r8
```

```
                    sub     $1,     %r8                 # list_length--
                    jmp     loop_tst1

done_loop1:
                    leave
                    ret

################################################################
#
 # Function:    Swap
 # Purpose:     Swap contents of x_p and y_p
 #
 # Args:
 #                          rdi = *x_p
 #                          rsi = *y_p


Swap:
        push   %rbp
        mov    %rsp,  %rbp

        mov    %rdi,  %r8                 # temp = *x_p
        mov    %rsi,  %rdi                # rdi = *x_p = *y_p = rsi
        mov    %r8,   %rsi                # rsi = *y_p = temp =r8
```