

```
; Don Spickler
;
; Java implementation of the bubble sort.
;
;public static void BubbleSort(int A[]) {
;    for (int i = A.length - 1; i > 0; i--) {
;        for (int j = 0; j < i; j++) {
;            if (A[j] > A[j + 1]) {
;                int temp = A[j];
;                A[j] = A[j + 1];
;                A[j + 1] = temp;
;            }
;        }
;    }
;}

; Simple bubble sort array program. Reads array contents from commandline, sorts and prints sorted array.
; Compile with: nasm -f elf arraysort.asm
; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 arraysort.o -o arraysort
; Or run make
; Run with: ./array a1 a2 a3 ... an
```

```
%include 'functions.asm'
```

SECTION .data

```
msg1      db      'Sorted Contents: ' ; a message string for output result
msg2      db      ' '                ; blank
```

SECTION .bss

```
A         resd 100 ; array storage for up to 100 integers.
n         resd 1   ; size of the array
```

SECTION .text

```
global _start
```

```
_start:
    pop     ecx          ; first element is the number of arguments
    cmp     ecx, 1       ; if ecx is 1 then halt, no argument or more than one.
    je      finish       ; jump to finish if ecx is 1
    dec     ecx          ; decrement to get size of array
    mov     [n], ecx     ; Store array size

    pop     eax          ; program name.
    mov     edx, A       ; store starting location of memory block for array

readloop:
    cmp     ecx, 0       ; if counter is 0 we stop reading.
    je      sortarray    ; when finished, sort array
    pop     eax          ; read next argument from stack (command line)
    call    atoi         ; convert to integer
    mov     [edx], eax   ; store integer in array at current pointer position
    add     edx, 4       ; increment pointer to next location. Storing double words = 4 bytes

    dec     ecx          ; decrement counter
    jmp     readloop     ; loop back

sortarray:
    mov     eax, [n]     ; eax is the i loop, set to n

iloop:
    dec     eax          ; decrement eax, so start is at n - 1 and takes care of the i--
    cmp     eax, 0       ; if i = 0 finish and write array
    je      writearray
    mov     ebx, 0       ; ebx is the j loop, set to 0
    mov     ecx, A       ; ecx is the pointer position for j, i.e. A[j]
    mov     edx, A       ; edx is the pointer position for j + 1, i.e. A[j + 1]
    add     edx, 4       ; set edx to the j + 1 position

jloop:
    cmp     ebx, eax     ; if j < i continue
    jge     iloop        ; otherwise break to the i loop

    mov     esi, [ecx]   ; move A[j] to spare index register for comparison
    mov     edi, [edx]   ; move A[j + 1] to spare index register for comparison
    cmp     esi, edi     ; if A[j] <= A[j + 1] continue, otherwise swap values
```

```
jle     next
mov     [ecx], edi ; swap A[j] and A[j + 1]
mov     [edx], esi
|
next:
add     ecx, 4      ; increment j pointer
add     edx, 4      ; increment j + 1 pointer
inc     ebx         ; increment j, i.e. j++
jmp     jloop       ; loop back to j loop
|
writearray:
mov     ecx, [n]    ; reset counter to array size
mov     edx, A      ; store starting location of memory block for array
mov     eax, msg1    ; print out the output message
call    sprint
|
writeloop:
cmp     ecx, 0      ; test if finished writing.
je      finish      ; if so go to finish
mov     eax, [edx]   ; load array element into eax
call    iprint       ; print integer
mov     eax, msg2    ; load the space
call    sprint        ; print space

add     edx, 4      ; increment pointer to next location. Storing double words = 4 bytes
dec     ecx         ; decrement counter
jmp     writeloop    ; loop back
|
finish:
mov     eax, msg2    ; load the space
call    sprintLF     ; print the space with a line feed
call    quit         ; end program
```