```
 1:  TITLE    Sorting an array by insertion sort    INS_SORT.ASM
 2:  COMMENT |
 3:          Objective: To sort an integer array using insertion sort.
 4:              Input: Requests numbers to fill array.
 5:  |           Output: Displays sorted array.
 6:  .MODEL SMALL
 7:  .STACK 100H
 8:  .DATA
 9:  MAX_SIZE       EQU 100
10:  array          DW  MAX_SIZE DUP (?)
11:  input_prompt   DB  'Please enter input array: '
12:                 DB  '(negative number terminates input)',0
13:  out_msg        DB  'The sorted array is:',0
14:
15:  .CODE
16:  .486
17:  INCLUDE io.mac
18:  main    PROC
19:          .STARTUP
20:          PutStr  input_prompt ; request input array
21:          mov     BX,OFFSET array
22:          mov     CX,MAX_SIZE
```

Addressing modes: 1

```
23:  array_loop:
24:          GetInt  AX           ; read an array number
25:          nwln
26:          cmp     AX,0         ; negative number?
27:          jl      exit_loop    ; if so, stop reading numbers
28:          mov     [BX],AX      ; otherwise, copy into array
29:          add     BX,2         ; increment array address
30:          loop    array_loop   ; iterates a maximum of MAX_SIZE
31:  exit_loop:
32:          mov     DX,BX        ; DX keeps the actual array size
33:          sub     DX,OFFSET array  ; DX := array size in bytes
34:          shr     DX,1         ; divide by 2 to get array size
35:          push    DX           ; push array size & array pointer
36:          push    OFFSET array
37:          call    insertion_sort
38:          PutStr  out_msg      ; display sorted array
39:          nwln
40:          mov     CX,DX
41:          mov     BX,OFFSET array
```

Addressing modes: 2

```
42:  display_loop:
43:          PutInt  [BX]
44:          nwln
45:          add     BX,2
46:          loop    display_loop
47:  done:
48:          .EXIT
49:  main    ENDP
50:
51:  ;---------------------------------------------------------
52:  ; This procedure receives a pointer to an array of integers
53:  ; and the array size via the stack. The array is sorted by
54:  ; using insertion sort. All registers are preserved.
55:  ;---------------------------------------------------------
56:  SORT_ARRAY  EQU  [BX]
57:  insertion_sort PROC
58:         pusha                ; save registers
59:         mov     BP,SP
60:         mov     BX,[BP+18]   ; copy array pointer
61:         mov     CX,[BP+20]   ; copy array size
62:         mov     SI,2         ; array left of SI is sorted
```

Addressing modes: 3

```
63:  for_loop:
64:          ; variables of the algorithm are mapped as follws:
65:          ; DX = temp, SI = i, and DI = j
66:          mov     DX,SORT_ARRAY[SI] ; temp := array[i]
67:          mov     DI,SI         ; j := i-1
68:          sub     DI,2
69:  while_loop:
70:          cmp     DX,SORT_ARRAY[DI]  ; temp < array[j]
71:          jge     exit_while_loop
72:          ; array[j+1] := array[j]
73:          mov     AX,SORT_ARRAY[DI]
74:          mov     SORT_ARRAY[DI+2],AX
75:          sub     DI,2          ; j := j-1
76:          cmp     DI,0          ; j >= 0
77:          jge     while_loop
78:  exit_while_loop:
79:          ; array[j+1] := temp
80:          mov     SORT_ARRAY[DI+2],DX
81:          add     SI,2          ; i := i+1
82:          dec     CX
83:          cmp     CX,1          ; if CX = 1, we are done
84:          jne     for_loop
85:  sort_done:
86:          popa                  ; restore registers
87:          ret     4
88:  insertion_sort ENDP
89:          END     main
```

Addressing modes: 4

```
 1:  TITLE   Binary search of a sorted integer array   BIN_SRCH.ASM
 2:  COMMENT |
 3:          Objective: To implement binary search of a sorted
 4:                     integer array.
 5:              Input: Requests numbers to fill array and a
 6:                     number to be searched for from user.
 7:             Output: Displays the position of the number in
 8:                     the array if found; otherwise, not found
 9:  |                  message.
10:  .MODEL SMALL
11:  .STACK 100H
12:  .DATA
13:  MAX_SIZE       EQU 100
14:  array          DW  MAX_SIZE DUP (?)
15:  input_prompt   DB  'Please enter input array (in sorted order): '
16:                 DB  '(negative number terminates input)',0
17:  query_number   DB  'Enter the number to be searched: ',0
18:  out_msg        DB  'The number is at position ',0
19:  not_found_msg  DB  'Number not in the array!',0
20:  query_msg      DB  'Do you want to quit (Y/N): ',0
21:
22:  .CODE
23:  .486
24:  INCLUDE io.mac
```

Addressing modes: 5

```
25:  main    PROC
26:          .STARTUP
27:          PutStr  input_prompt ; request input array
28:          nwln
29:          sub     ESI,ESI      ; set index to zero
30:          mov     CX,MAX_SIZE
31:  array_loop:
32:          GetInt  AX           ; read an array number
33:          nwln
34:          cmp     AX,0            ; negative number?
35:          jl      exit_loop       ; if so, stop reading numbers
36:          mov     array[ESI*2],AX ; otherwise, copy into array
37:          inc     SI           ; increment array index
38:          loop    array_loop   ; iterates a maximum of MAX_SIZE
39:  exit_loop:
40:  read_input:
41:          PutStr  query_number ; request number to be searched for
42:          GetInt  AX           ; read the number
43:          nwln
44:          push    AX           ; push number, size & array pointer
45:          push    SI
46:          push    OFFSET array
47:          call    binary_search
48:          ; binary_search returns in AX the position of the number
49:          ; in the array; if not found, it returns 0.
```

Addressing modes: 6

```
50:          cmp     AX,0          ; number found?
51:          je      not_found     ; if not, display number not found
52:          PutStr  out_msg       ; else, display number position
53:          PutInt  AX
54:          jmp     user_query
55:  not_found:
56:          PutStr  not_found_msg
57:  user_query:
58:          nwln
59:          PutStr  query_msg     ; query user whether to terminate
60:          GetCh   AL            ; read response
61:          nwln
62:          cmp     AL,'Y'        ; if response is not 'Y'
63:          jne     read_input    ; repeat the loop
64:  done:                         ; otherwise, terminate program
65:          .EXIT
66:  main    ENDP
67:
68:  ;-----------------------------------------------------------
69:  ; This procedure receives a pointer to an array of integers,
70:  ; the array size, and a number to be searched via the stack.
71:  ; It returns in AX the position of the number in the array
72:  ; if found; otherwise, returns 0.
73:  ; All registers, except AX, are preserved.
74:  ;-----------------------------------------------------------
```

Addressing modes: 7

```
75:  binary_search  PROC
76:          push    BP            ; save registers
77:          mov     BP,SP
78:          push    EBX
79:          push    ESI
80:          push    CX
81:          push    DX
82:          sub     EBX,EBX       ; EBX := 0
83:          mov     BX,[BP+4]     ; copy array pointer
84:          mov     CX,[BP+6]     ; copy array size
85:          mov     DX,[BP+8]     ; copy number to be searched
86:          sub     AX,AX         ; lower := 0
87:          dec     CX            ; upper := size-1
88:  while_loop:
89:          cmp     AX,CX         ;lower > upper?
90:          ja      end_while
91:          sub     ESI,ESI
92:          mov     SI,AX         ; middle := (lower + upper)/2
93:          add     SI,CX
94:          shr     SI,1
95:          cmp     DX,[EBX+ESI*2]    ; number = array[middle]?
96:          je      search_done
97:          jg      upper_half
```

Addressing modes: 8

```
 98:  lower_half:
 99:          dec     SI              ; middle := middle-1
100:          mov     CX,SI           ; upper := middle-1
101:          jmp     while_loop
102:  upper_half:
103:          inc     SI              ; middle := middle+1
104:          mov     AX,SI           ; lower := middle+1
105:          jmp     while_loop
106:  end_while:
107:          sub     AX,AX           ; number not found (clear AX)
108:          jmp     skip1
109:  search_done:
110:          inc     SI              ; position := index+1
111:          mov     AX,SI           ; return position
112:  skip1:
113:          pop     DX              ; restore registers
114:          pop     CX
115:          pop     ESI
116:          pop     EBX
117:          pop     BP
118:          ret     6
119:  binary_search  ENDP
120:          END     main
```

Addressing modes: 9

```
 1:  TITLE       Sum of a long integer array       ARAY_SUM.ASM
 2:  COMMENT |
 3:          Objective: To find sum of all elements of an array.
 4:              Input: None
 5:  |          Output: Displays the sum.
 6:  .MODEL SMALL
 7:  .STACK 100H
 8:  .DATA
 9:  test_marks     DD  90,50,70,94,81,40,67,55,60,73
10:  NO_STUDENTS    EQU ($-test_marks)/4     ; number of students
11:  sum_msg        DB  'The sum of test marks is: ',0
12:
13:  .CODE
14:  .486
15:  INCLUDE io.mac
```

Addressing modes: 10

```
 16:  main    PROC
 17:          .STARTUP
 18:          mov     CX,NO_STUDENTS   ; loop iteration count
 19:          sub     EAX,EAX          ; sum := 0
 20:          sub     ESI,ESI          ; array index := 0
 21:  add_loop:
 22:          mov     EBX,test_marks[ESI*4]
 23:          PutLInt EBX
 24:          nwln
 25:          add     EAX,test_marks[ESI*4]
 26:          inc     ESI
 27:          loop    add_loop
 28:
 29:          PutStr  sum_msg
 30:          PutLInt EAX
 31:          nwln
 32:          .EXIT
 33:  main    ENDP
 34:          END     main
```

Addressing modes: 11

```
 1:  TITLE  Sum of a column in a 2-dimensional array  TEST_SUM.ASM
 2:  COMMENT |
 3:          Objective: To demonstrate array index manipulation
 4:                     in a two-dimensional array of integers.
 5:              Input: None
 6:  |          Output: Displays the sum.
 7:  .MODEL SMALL
 8:  .STACK 100H
 9:  .DATA
10:  NO_ROWS        EQU  5
11:  NO_COLUMNS     EQU  3
12:  NO_ROW_BYTES   EQU  NO_COLUMNS * 2  ; number of bytes per row
13:  class_marks    DW   90,89,99
14:                 DW   79,66,70
15:                 DW   70,60,77
16:                 DW   60,55,68
17:                 DW   51,59,57
18:
19:  sum_msg        DB   'The sum of the last test marks is: ',0
20:
21:  .CODE
22:  .486
23:  INCLUDE io.mac
```

Addressing modes: 12

```
 24:  main    PROC
 25:          .STARTUP
 26:          mov     CX,NO_ROWS   ; loop iteration count
 27:          sub     AX,AX        ; sum := 0
 28:          ; ESI := index of class_marks[0,2]
 29:          sub     EBX,EBX
 30:          mov     ESI,NO_COLUMNS-1
 31:  sum_loop:
 32:          add     AX,class_marks[EBX+ESI*2]
 33:          add     EBX,NO_ROW_BYTES
 34:          loop    sum_loop
 35:
 36:          PutStr  sum_msg
 37:          PutInt  AX
 38:          nwln
 39:  done:
 40:          .EXIT
 41:  main    ENDP
 42:          END     main
```

Addressing modes: 13

```
 1:  ;---------------------------------------------------------
 2:  ; This procedure receives a pointer to an array of integers
 3:  ; and the array size via the stack. The array is sorted by
 4:  ; using insertion sort. All registers are preserved.
 5:  ;---------------------------------------------------------
 6:  SORT_ARRAY  EQU  [EBX]
 7:  insertion_sort PROC
 8:         pushad               ; save registers
 9:         mov     BP,SP
10:         sub     EBX,EBX
11:         mov     BX,[BP+34]   ; copy array pointer
12:         mov     CX,[BP+36]   ; copy array size
13:         mov     ESI,1        ; array left of ESI is sorted
14:  for_loop:
15:         ; variables of the algorithm are mapped as follows:
16:         ; DX = temp, ESI = i, and EDI = j
17:         mov     DX,SORT_ARRAY[ESI*2] ; temp := array[i]
18:         mov     EDI,ESI      ; j := i-1
19:         dec     EDI
```

Addressing modes: 14

```
20:  while_loop:
21:         cmp     DX,SORT_ARRAY[EDI*2]  ; temp < array[j]
22:         jge     exit_while_loop
23:         ; array[j+1] := array[j]
24:         mov     AX,SORT_ARRAY[EDI*2]
25:         mov     SORT_ARRAY[EDI*2+2],AX
26:         dec     EDI           ; j := j-1
27:         cmp     EDI,0         ; j >= 0
28:         jge     while_loop
29:  exit_while_loop:
30:         ; array[j+1] := temp
31:         mov     SORT_ARRAY[EDI*2+2],DX
32:         inc     ESI           ; i := i+1
33:         dec     CX
34:         cmp     CX,1          ; if CX = 1, we are done
35:         jne     for_loop
36:  sort_done:
37:         popad                 ; restore registers
38:         ret     4
39:  insertion_sort ENDP
```

Addressing modes: 15