Question 1 (15 points)

Write an **AVR Assembly program** that sorts an array of **8 unsigned bytes** stored in **SRAM** (starting at address 0x0100) into **descending order**, using the **Bubble Sort** algorithm, and then stores the sorted array starting from 0x0110.

The initial data stored in address 0x0100 is **{23, 5, 17, 12, 45, 8, 30, 2}**.

The bubble sort algorithm is when $A[i] > A[i+1]$ swaps the data.

Bubble Sort

```
;=========================================================
; ATmega328P - Copy Flash array to SRAM and sort (Descending)
;=========================================================

.include "m328pdef.inc"

;-----------------------------
; Flash (Program) Memory
;-----------------------------
.cseg
.org 0x0000
    rjmp start

; Source array in Flash
SOURCE_DATA:
    .db 23, 5, 17, 12, 45, 8, 30, 2

.equ SIZE = 8
```

```
;-----------------------------
; SRAM (Data Memory)
;-----------------------------
.dseg
DEST_DATA: .byte SIZE

;-----------------------------
; Code Section
;-----------------------------
.cseg

start:

    ;---------------------------------
    ; Initialize Stack Pointer
    ;---------------------------------
    ldi r16, low(RAMEND)
    out SPL, r16
    ldi r16, high(RAMEND)
    out SPH, r16

    ;---------------------------------
    ; Copy from Flash to SRAM
    ;---------------------------------
    ; Z pointer -> Flash source
    ldi r30, low(SOURCE_DATA * 2)
    ldi r31, high(SOURCE_DATA * 2)

    ; Y pointer -> SRAM destination
    ldi r28, low(DEST_DATA)
    ldi r29, high(DEST_DATA)

    ldi r20, SIZE

copy_loop:
    lpm r16, Z+        ; Load from Flash
    st  Y+, r16        ; Store into SRAM
    dec r20
    brne copy_loop

    ;---------------------------------
    ; Bubble Sort (Descending)
    ;---------------------------------

    ldi r21, SIZE      ; Outer loop counter

outer_loop:
    ldi r28, low(DEST_DATA)
    ldi r29, high(DEST_DATA)

    mov r20, r21
    dec r20
    breq done

inner_loop:
    ld  r16, Y         ; A[i]
    ldd r17, Y+1       ; A[i+1]

    cp  r16, r17
    brsh no_swap       ; If A[i] >= A[i+1], skip swap

    ; Swap
    st  Y, r17
    std Y+1, r16

no_swap:
    adiw r28, 1        ; Y++
    dec r20
    brne inner_loop

    dec r21
    brne outer_loop

done:
    rjmp done
```

Bubble Sort (Ascending)

```
;=========================================================
; ATmega328P - Copy Flash array to SRAM and sort (Ascending)
;=========================================================

.include "m328pdef.inc"

;-----------------------------
; Flash (Program) Memory
;-----------------------------
.cseg
.org 0x0000
    rjmp start

; Source array in Flash
SOURCE_DATA:
    .db 23, 5, 17, 12, 45, 8, 30, 2

.equ SIZE = 8

;-----------------------------
; SRAM (Data Memory)
;-----------------------------
.dseg
DEST_DATA: .byte SIZE
```

```
;-----------------------------
; Code Section
;-----------------------------
.cseg

start:

    ;---------------------------------
    ; Initialize Stack Pointer
    ;---------------------------------
    ldi r16, low(RAMEND)
    out SPL, r16
    ldi r16, high(RAMEND)
    out SPH, r16

    ;---------------------------------
    ; Copy from Flash to SRAM
    ;---------------------------------
    ; Z -> Flash source
    ldi r30, low(SOURCE_DATA * 2)
    ldi r31, high(SOURCE_DATA * 2)

    ; Y -> SRAM destination
    ldi r28, low(DEST_DATA)
    ldi r29, high(DEST_DATA)
    ldi r20, SIZE

copy_loop:
    lpm r16, Z+
    st  Y+, r16
    dec r20
    brne copy_loop

    ;---------------------------------
    ; Bubble Sort (Ascending)
    ;---------------------------------

    ldi r21, SIZE

outer_loop:
    ldi r28, low(DEST_DATA)
    ldi r29, high(DEST_DATA)

    mov r20, r21
    dec r20
    breq done

inner_loop:
    ld  r16, Y          ; A[i]
    ldd r17, Y+1        ; A[i+1]
    cp  r16, r17
    brlo no_swap        ; If A[i] < A[i+1], correct order (ascending)

    ; Swap if A[i] >= A[i+1]
    st  Y, r17
    std Y+1, r16

no_swap:
    adiw r28, 1
    dec r20
    brne inner_loop

    dec r21
    brne outer_loop

done:
    rjmp done
```

Insertion Sort (Ascending)

```
.include "m328pdef.inc"

.equ SIZE = 8

.dseg
array: .byte SIZE

.cseg
.org 0x0000
    rjmp start

source:
    .db 23, 5, 17, 12, 45, 8, 30, 2

start:
    ldi r16, low(RAMEND)
    out SPL, r16
    ldi r16, high(RAMEND)
    out SPH, r16

    clr r1                  ; r1 used as Zero Register

    ldi r30, low(source*2)  ; Z points to source (Flash)
    ldi r31, high(source*2)
    ldi r28, low(array)     ; Y points to array (SRAM)
    ldi r29, high(array)
    ldi r20, SIZE

copy:
    lpm r16, Z+
    st  Y+, r16
    dec r20
    brne copy

    ldi r18, 1              ; i = 1

outer:
    cpi r18, SIZE
    brge done

    ; --- Load Key (r17 = array[i]) ---
    ldi r28, low(array)
    ldi r29, high(array)
    add r28, r18
    adc r29, r1
    ld  r17, Y

    mov r19, r18
    dec r19                 ; j = i - 1
```

```
inner:
    cpi r19, 0xFF           ; Check for underflow (j == -1)
    breq insert

    ; --- Load array[j] ---
    ldi r28, low(array)
    ldi r29, high(array)
    add r28, r19
    adc r29, r1
    ld  r16, Y

    cp  r16, r17            ; Compare array[j] with Key
    brlo insert             ; If array[j] < Key, found spot (Ascending)

    ; --- Shift: array[j+1] = array[j] ---
    adiw r28, 1             ; Y points to j+1
    st  Y, r16

    dec r19                 ; j--
    rjmp inner

insert:
    ; --- FIX IS HERE ---
    ; We need to write Key to array[j+1].
    ; If j was 0xFF (-1), we need offset 0.
    ; Previous code did: Base + 255 + 1 = Base + 256 (Memory Error)
    mov r20, r19            ; Move j to temp register
    inc r20                 ; r20 = j + 1. (0xFF becomes 0x00 correctly)

    ldi r28, low(array)
    ldi r29, high(array)
    add r28, r20            ; Add valid offset
    adc r29, r1             ; Propagate carry
    st  Y, r17              ; Store Key

    inc r18                 ; i++
    rjmp outer

done:
    rjmp done
```

Insertion Sort (Descending)

```
.include "m328pdef.inc"

.equ SIZE = 8

.dseg
array: .byte SIZE

.cseg
.org 0x0000
```

```
    rjmp start

source:
    .db 23, 5, 17, 12, 45, 8, 30, 2

start:
    ; --- Stack Initialization ---
    ldi r16, low(RAMEND)
    out SPL, r16
    ldi r16, high(RAMEND)
    out SPH, r16

    clr r1                  ; r1 used as Zero Register

    ; --- Copy Data from Flash to SRAM ---
    ldi r30, low(source*2)  ; Z points to source
    ldi r31, high(source*2)
    ldi r28, low(array)     ; Y points to array
    ldi r29, high(array)
    ldi r20, SIZE

copy:
    lpm r16, Z+
    st  Y+, r16
    dec r20
    brne copy
    ; --- Begin Insertion Sort ---
    ldi r18, 1              ; i = 1

outer:
    cpi r18, SIZE
    brge done

    ; --- Load Key (r17 = array[i]) ---
    ldi r28, low(array)
    ldi r29, high(array)
    add r28, r18
    adc r29, r1
    ld  r17, Y              ; Key is in r17

    mov r19, r18
    dec r19                 ; j = i - 1

inner:
    cpi r19, 0xFF           ; Check boundary (j == -1)
    breq insert

    ; --- Load array[j] ---
    ldi r28, low(array)
    ldi r29, high(array)
    add r28, r19
    adc r29, r1
    ld  r16, Y              ; r16 = array[j]

    ; --- DESCENDING COMPARE ---
    cp  r16, r17            ; Compare array[j] with Key
    brsh insert             ; IF array[j] >= Key, STOP shifting.
                            ; (Because we want large numbers at the top)

    ; --- Shift: array[j+1] = array[j] ---
    adiw r28, 1             ; Y is now pointing to j+1
    st  Y, r16              ; Move smaller number down

    dec r19                 ; j--
    rjmp inner

insert:
    ; --- Insert Key into array[j+1] ---
    mov r20, r19            ; Move j to temp
    inc r20                 ; r20 = j + 1 (Fixes the 0xFF underflow issue)

    ldi r28, low(array)
    ldi r29, high(array)
    add r28, r20
    adc r29, r1
    st  Y, r17              ; Store Key
    inc r18                 ; i++
    rjmp outer

done:
    rjmp done
```

Selection Sort (Descending)

```
.include "m328pdef.inc"

.equ SIZE = 8

.dseg
array: .byte SIZE

.cseg
.org 0x0000
    rjmp start

source:
    .db 23, 5, 17, 12, 45, 8, 30, 2

start:
    ; --- Stack & Setup ---
    ldi r16, low(RAMEND)
    out SPL, r16
    ldi r16, high(RAMEND)
    out SPH, r16
    clr r1

    ; --- Copy Flash to SRAM ---
    ldi r30, low(source*2)
    ldi r31, high(source*2)
    ldi r28, low(array)
    ldi r29, high(array)
    ldi r20, SIZE

copy:
    lpm r16, Z+
    st  Y+, r16
    dec r20
    brne copy

    ; ==========================================
    ; SELECTION SORT (DESCENDING)
    ; ==========================================

    clr r18                 ; i = 0

outer:
    cpi r18, SIZE - 1
    brge done
    ; --- Assume array[i] is the max initially ---
    mov r20, r18            ; max_idx = i (Store Index)

    ; Load array[i] into r21 (max_val)
    ldi r28, low(array)
    ldi r29, high(array)
    add r28, r18
    adc r29, r1
    ld  r21, Y              ; r21 = max_val (Store Value)

    ; --- Inner Loop Setup ---
    mov r19, r18
    inc r19                 ; j = i + 1

inner:
    cpi r19, SIZE
    brge perform_swap       ; **CHANGED LABEL HERE**

    ; --- Load array[j] ---
    ldi r28, low(array)
    ldi r29, high(array)
    add r28, r19
    adc r29, r1
    ld  r16, Y              ; r16 = array[j]
    ; --- COMPARE ---
    cp  r16, r21            ; Compare array[j] with max_val
    brlo next_j             ; Descending: If array[j] < max, skip

    ; --- Update Max ---
    mov r21, r16            ; New max_val
    mov r20, r19            ; New max_idx

next_j:
    inc r19
    rjmp inner

perform_swap:               ; **CHANGED LABEL HERE**
    ; --- Swap array[i] and array[max_idx] ---
    cp  r18, r20            ; If i == max_idx, skip swap
    breq next_i

    ; 1. Load array[i] (Current position)
    ldi r28, low(array)
    ldi r29, high(array)
    add r28, r18
    adc r29, r1
    ld  r22, Y              ; r22 holds value at array[i]

    ; 2. Store max_val (r21) at array[i]
    st  Y, r21
    ; 3. Store old array[i] (r22) at array[max_idx]
    ldi r28, low(array)
    ldi r29, high(array)
    add r28, r20            ; Point to max_idx
    adc r29, r1
    st  Y, r22              ; Store old array[i] there

next_i:
    inc r18
    rjmp outer

done:
    rjmp done
```

Selection Sort (Ascending)

```
.include "m328pdef.inc"

.equ SIZE = 8

.dseg
array: .byte SIZE

.cseg
.org 0x0000
```

```
    rjmp start

source:
    .db 23, 5, 17, 12, 45, 8, 30, 2

start:
    ; --- Stack & Setup ---
    ldi r16, low(RAMEND)
    out SPL, r16
    ldi r16, high(RAMEND)
    out SPH, r16
    clr r1                  ; Zero register

    ; --- Copy Flash to SRAM ---
    ldi r30, low(source*2)
    ldi r31, high(source*2)
    ldi r28, low(array)
    ldi r29, high(array)
    ldi r20, SIZE

copy:
    lpm r16, Z+
    st  Y+, r16
    dec r20
    brne copy
    ; ==========================================
    ; SELECTION SORT (ASCENDING)
    ; r18 = i (Current Position)
    ; r19 = j (Scanner)
    ; r20 = min_idx (Index of smallest value found)
    ; r21 = min_val (Value of smallest value found)
    ; ==========================================

    clr r18                 ; i = 0

outer:
    cpi r18, SIZE - 1
    brge done

    ; --- Assume array[i] is the min initially ---
    mov r20, r18            ; min_idx = i

    ; Load array[i] into r21 (min_val)
    ldi r28, low(array)
    ldi r29, high(array)
    add r28, r18
    adc r29, r1
    ld  r21, Y              ; r21 = min_val

    ; --- Inner Loop Setup ---
    mov r19, r18
    inc r19                 ; j = i + 1

inner:
    cpi r19, SIZE
    brge perform_swap       ; Finished scanning, go swap

    ; --- Load array[j] ---
    ldi r28, low(array)
    ldi r29, high(array)
    add r28, r19
    adc r29, r1
    ld  r16, Y              ; r16 = array[j]

    ; --- ASCENDING COMPARE ---
    cp  r16, r21            ; Compare array[j] with min_val
    brsh next_j             ; IF array[j] >= min_val, skip update
                            ; (We only want smaller numbers)

    ; --- Update Min ---
    mov r21, r16            ; New min_val found
    mov r20, r19            ; Store its index

next_j:
    inc r19
    rjmp inner
```

```
perform_swap:
    ; --- Swap array[i] and array[min_idx] ---
    cp  r18, r20            ; If i == min_idx, no need to swap
    breq next_i

    ; 1. Load array[i] (Current position) - let's call it 'temp'
    ldi r28, low(array)
    ldi r29, high(array)
    add r28, r18
    adc r29, r1
    ld  r22, Y              ; r22 holds old array[i]

    ; 2. Store min_val (r21) at array[i]
    st  Y, r21

    ; 3. Store old array[i] (r22) at array[min_idx]
    ldi r28, low(array)
    ldi r29, high(array)
    add r28, r20            ; Point Y to min_idx
    adc r29, r1
    st  Y, r22              ; Store old array[i] there

next_i:
    inc r18
    rjmp outer
```

```
done:

    rjmp done
```