**Lab Report  
Lab 6: Pointer and Array**

|  |  |
| --- | --- |
| Student ID | Nguyễn Đức Anh |
| Student Name | 20235890 |

**Assignment 1:**

**Code:**

.data

    A: .word -2, 6, -1, 3, -2

    mes1: .asciz "The maximum prefix sum of length "

    mes2: .asciz " is: "

.text

main:

    la a0, A    #a0 = address of A

    li a1, 5    #

    j mspfx

continue:

print\_mes1:

    li a7, 4

    la a0, mes1

    ecall

print\_length:

    li a7, 1

    mv a0, s0

    ecall

print\_mes2:

    li a7, 4

    la a0, mes2

    ecall

print\_result:

    li a7, 1

    mv a0, s1

    ecall

exit:

    li a7, 10

    ecall

end\_of\_main:

mspfx:

    li s0, 0 # prefix sum length

    li s1, 0x80000000 # maximum prefix sum

    li t0, 0 # t0 = i

    li t1, 0 # running sum

loop:

    add t2, t0, t0

    add t2, t2, t2 # i = i + 4

    add t3, t2, a0 # 4i + A (address of A[i]) in t3

    lw t4, 0(t3) # load A[i] from mem(t3) into t4

    add t1, t1, t4 # add A[i] to running sum in t1

    blt s1, t1, mdfy # if(s1 < t1) modify results

    j next

mdfy:

    addi s0, t0, 1 # new max-sum prefix has length i+1

    addi s1, t1, 0 # new max sum is the running sum

next:

    addi t0, t0, 1 # advance the index i

    blt t0, a1, loop # if(i<n) repeat

done:

    j continue

mspfx\_end:

**Result:**

A screenshot of a computer

Description automatically generated

**Assignment 2:**

**Code:**

.data

    A: .word 1, -2, 5, 3, 9, 7, 13

    Aend: .word

.text

main:

    la a0, A # a0 = address(A[0])

    la a1, Aend

    addi a1, a1, -4 # a1 = address(A[n-1])

    j selection\_sort # sort

after\_sort:

    li a7, 10

    ecall

end\_main:

# a0 pointer to the first element in unsorted part

# a1 pointer to the last element in unsorted part

# t0 temporary place for value of last element

# s0 pointer to max element in unsorted part

# s1 value of max element in unsorted part

selection\_sort:

    beq a0, a1, done # single element list is sorted

    j max # call the max procedure

after\_max:

    lw t0, 0(a1) # load last element into $t0

    sw t0, 0(s0) # copy last element to max location

    sw s1, 0(a1) # copy max value to last element

    addi a1, a1, -4 # decrement pointer to last element

    j selection\_sort # repeat sort for smaller list

done:

    j after\_sort

# a0 pointer to first element

# a1 pointer to last element

max:

    addi s0, a0, 0 # init max pointer to first element

    lw s1, 0(s0) # init max value to first value

    addi t0, a0, 0 # init next pointer to first

loop:

    beq t0, a1, ret # if next=last, return

    addi t0, t0, 4 # advance to next element

    lw t1, 0(t0) # load next element into $t1

    blt t1, s1, loop # if (next)<(max), repeat

    addi s0, t0, 0 # next element is new max element

    addi s1, t1, 0 # next value is new max value

    j loop # change completed; now repeat

ret:

    j after\_max

**Result:**

**A screenshot of a computer

Description automatically generated**

**Explanation:**

Selection Sort Algorithm:

An array of **n** integers can be sorted in ascending order as follows: Find the largest element in the list and swap it with the last element of the array. The last element is now in its correct position. Repeat these steps for the remaining **n - 1** unsorted elements until only one element remains. At this point, the algorithm is complete, and the array is sorted in ascending order.

**Assignment 3:**

**Code:**

.data

    A: .word 1, -2, 5, 3, 9, 7, 13

    Aend: .word

.text

main:

    la a0, A        # a0 = address(A[0])

    la a1, Aend     # a1 = address(A[n-1])

    addi a1, a1, -4 # a1 points to last element

    j bubble\_sort

after\_sort:

    li a7, 10

    ecall

end\_main:

bubble\_sort:

    addi t2, a0, 0 # t2 = first element pointer

loop\_outer:

    beq a0, a1, done # If the list is fully sorted

    addi t0, a0, 0   # Reset inner loop pointer

loop\_inner:

    addi t1, t0, 4    # t1 = next element

    beq t1, a1, swap\_end # If at end of list, swap

    lw s0, 0(t0)      # Load current element

    lw s1, 0(t1)      # Load next element

    blt s1, s0, swap  # If next < current, swap

    addi t0, t0, 4    # Move to the next element

    j loop\_inner

swap:

    sw s1, 0(t0)      # Swap current and next

    sw s0, 0(t1)

swap\_end:

    addi a1, a1, -4   # Move the last unsorted element left

    j bubble\_sort

done:

    j after\_sort

**Result:**

**A screenshot of a computer

Description automatically generated**

**Explanation:**

Bubble Sort Algorithm:

* Adjacent elements are compared, and the larger one is "bubbled up" to the end of the list. This process is repeated until the list is sorted.
* The outer loop (loop\_outer) ensures that the last unsorted element is placed correctly.
* The inner loop (loo\_inner) compares adjacent elements and swaps them if necessary.
* Once the largest unsorted element "bubbles up" to its correct position, the outer loop shortens the range by one element.

**Assignment 4:**

**Code:**

.data

    A: .word 1, -2, 5, 3, 9, 7, 13

    Aend: .word

.text

main:

    la a0, A         # a0 = address(A[0])

    la a1, Aend      # a1 = address(A[n-1])

    j insertion\_sort

after\_sort:

    li a7, 10

    ecall

end\_main:

insertion\_sort:

    addi t0, a0, 4    # Start from the second element

    beq t0, a1, done  # If there's only one element, it's already sorted

loop\_outer:

    lw s1, 0(t0)      # Load the current element

    addi t1, t0, -4   # t1 points to the previous element

loop\_inner:

    blt t1, a0, insert # If reached the beginning, insert

    lw s0, 0(t1)      # Load the previous element

    blt s0, s1, insert # If previous < current, insert

    sw s0, 4(t1)      # Shift the previous element right

    addi t1, t1, -4   # Move left to check the next element

    j loop\_inner

insert:

    sw s1, 4(t1)      # Insert the current element in its sorted position

    addi t0, t0, 4    # Move to the next element

    beq t0, a1, done  # If at the end, we're done

    j loop\_outer

done:

    j after\_sort

**Result:**

**A screenshot of a computer

Description automatically generated**

**Explanation:**

Insertion Sort Algorithm:

* This algorithm builds a sorted list one element at a time by inserting each element into its correct position.
* The outer loop picks each element from the unsorted part.
* The inner loop shifts larger elements of the sorted part to the right to make room for the current element.
* The current element is inserted in its correct position within the sorted part.