# BLG 212E Microprocessor Systems Homework 2 Report

**Ömer Yıldız**

Spring 2024

## 1. Introduction

This project focuses on implementing and benchmarking sorting algorithms on an ARM Cortex M0+ microcontroller. The primary goals include:

- Implementing Bubble Sort in assembly (`ft_lstsort_asm`).
- Using the SysTick timer to measure execution times in both C and assembly implementations.
- Comparing Bubble Sort's performance against MergeSort.
- Comparing empirical results with theoretical $O(n \log n)$ and $O(n^2)$ complexities.

## 2. Implementations

### 2.1 Bubble Sort Assembly Implementation

The Bubble Sort algorithm for singly linked lists was implemented in ARM assembly, with the key operations being:

- Traversing the linked list and comparing adjacent nodes.
- Swapping the data of nodes if necessary.
- Repeating until no swaps are needed.

Below is the implementation:

Listing 1: Bubble Sort Assembly Code (ft_lstsort_asm)

```
AREA    Sorting_Code, CODE, READONLY
ALIGN
THUMB
EXPORT  ft_lstsort_asm

ft_lstsort_asm FUNCTION
    PUSH    {R0-R1, LR}                ; Save registers
    MOVS     R3, #1                    ; R3 = flag for swaps

sort_loop
    LDR     R2, [SP]                  ; R2 = head pointer
    CMP     R3, #0                    ; Check if swaps occurred
    BEQ     done                      ; Exit if sorted
    MOVS     R3, #0                   ; Reset swap flag
    LDR     R4, [R2]                  ; Start at head

traverse
    LDR     R5, [R4, #4]              ; R5 = next node pointer
    CMP     R5, #0                    ; End of list?
    BEQ     check_swaps               ; If yes, check swaps

    ; Compare current and next node data
    LDR     R6, [R4]                  ; R6 = current data
    LDR     R7, [R5]                  ; R7 = next data
    LDR     R2, [SP, #4]              ; Preserve function pointer
    MOVS     R0, R6                   ; R0 = current data
    MOVS     R1, R7                   ; R1 = next data
    PUSH    {R3}                      ; Save flag
    BLX     R2                        ; Call comparison function
    POP     {R3}                      ; Restore flag

    CMP     R0, #1                    ; Check comparison result
    BEQ     no_swap                     ; Skip if sorted

    ; Swap data
    STR     R7, [R4]                    ; Swap current
       with next
    STR     R6, [R5]                    ; Swap next with
       current
    MOVS     R3, #1                     ; Flag swap
       occurred

no_swap
    MOVS     R4, R5                     ; Move to next
       node
    B          traverse                 ; Continue
       traversal

check_swaps
    CMP     R3, #0                      ; Check if swaps
       occurred
    BNE     sort_loop                   ; Repeat if swaps
       occurred

done
    POP     {R0-R1, PC}                 ; Restore and
       return
ENDFUNC
```

## 2.2 SysTick Timer

The SysTick timer was configured in assembly to measure the execution time of sorting functions. Below is the implementation:

Listing 2: SysTick Timer Assembly Code

```
AREA    Timing_Code, CODE, READONLY
ALIGN
THUMB
EXPORT  Systick_Start_asm
EXPORT  Systick_Stop_asm
EXPORT  SysTick_Handler
EXTERN  ticks

SysTick_Handler FUNCTION
    PUSH    {R4, LR}
    LDR     R4, =ticks
    LDR     R1, [R4]
    ADDS    R1, R1, #1
    STR     R1, [R4]
    POP     {R4, PC}
ENDFUNC

Systick_Start_asm FUNCTION
    PUSH    {R4-R7, LR}
    LDR     R5, =0xE000E010
    LDR     R6, =0xE000E014
    LDR     R7, =0xE000E018
    LDR     R0, =ticks
    MOVS    R1, #0
    STR     R1, [R0]
    MOVS    R4, #249
    STR     R4, [R6]
    STR     R1, [R7]
    MOVS    R0, #7
    STR     R0, [R5]
    POP     {R4-R7, PC}
ENDFUNC

Systick_Stop_asm FUNCTION
    PUSH    {R4, LR}
    LDR     R4, =0xE000E010
    MOVS    R1, #0
    STR     R1, [R4]
    LDR     R4, =ticks
    LDR     R0, [R4]
    POP     {R4, PC}
ENDFUNC
```

# 3. Results and Analysis

The sorting times for each iteration were measured and compared. Below is the graph illustrating the results of MergeSort (C), BubbleSort (ASM), and their theoretical complexities ($O(n \log n)$ and $O(n^2)$).

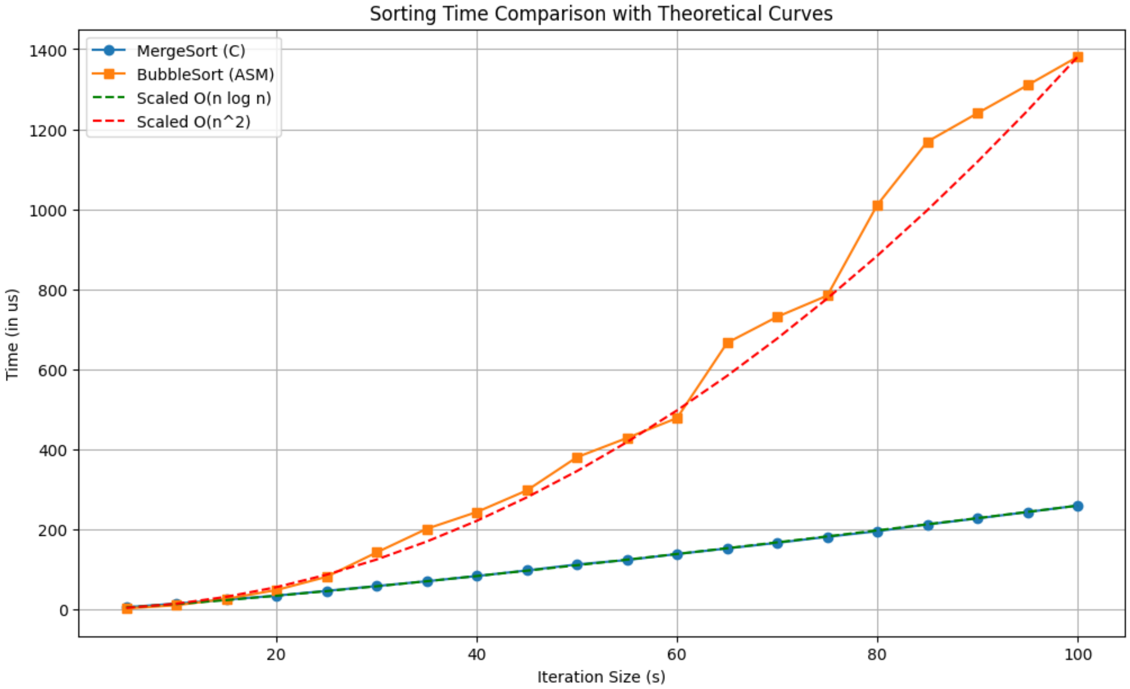

Figure 1: Sorting Time Comparison: MergeSort (C), BubbleSort (ASM), and Theoretical Curves

### Observations

- MergeSort follows the expected $O(n \log n)$ complexity closely.
- BubbleSort exhibits $O(n^2)$ behavior, with significantly higher times for larger inputs.
- Assembly implementation shows expected functionality but highlights inefficiencies of BubbleSort for large datasets.

## 4. Conclusion

This project successfully implemented and benchmarked sorting algorithms in C and ARM assembly. The results align with theoretical expectations, demonstrating the efficiency of MergeSort compared to BubbleSort. The assembly implementation provided insights into low-level optimizations and their impact on performance.