ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 212E MICROPROCESSOR SYSTEMS

Homework 2 Report

150210097 : Ali Emre Kaya

FALL 2024

Contents

1 Introduction **1**

2 System Timer **1**

3 Sorting Algorithm **1**
3.1 Merge Sort 1
3.2 Bubble Sort 2

4 BigO Analysis **2**

5 Conclusion **4**

REFERENCES **5**

1 Introduction

In this homework, analyze the difference of bubblesort and mergesort algorithms, mergesort algorithm is already given to the homework in the file **ft_lstsort.c**, I only wrote the bubblesort algorithm to the file **ft_lstsort_asm.s**. To facilitate time comparison, a SysTick comparative logic was initially provided in the **timing.c** file, but I need to convert this to an assembly file which named **timing_asm.s**.

2 System Timer

Systic Timer has three main function named **SysTick_Handler**, **Systick_Start**, **Systick_Stop**.

- **Systick Start**: This function is responsible for initializing and starting the SysTick timer. SysTick Control and Status Register which stored into the **0xE000E010** serves as the primary control point for enabling, disabling, and monitoring the SysTick timer. To enable SysTick processor and interrups #7 (0111) value assigned to this address. Reload value is **249**. .It is calculated by ((SystemCoreClock/Frequency) - 1) formula, Frequency is given as 100000 (1µs interval), and SystemCoreClock is given as 25000000 which is **0x017D7840**.
- **SysTick Handler**: Handler function increments the ticks value to count the number of interrupts that have occurred.
- **Systick Stop**: Stop function will disable the SysTick timer. It loads the base address of the SysTick Control and Status Register (at 0xE000E010) into register R0 and writes 0 to this register to disable the SysTick timer. Then it takes the ticks value and retrive it, after that reset it to zero for next measurements.

3 Sorting Algorithm

3.1 Merge Sort

Merge sort is a sorting algorithm that operates based on the divide and conquer logic. It recursively divides the input into smaller subarrays, sorts them, and then merges the sorted subarrays to produce the final sorted array. This algorithm has a time complexity of $O(n * log(n))$, making it efficient for large datasets. Merge sort uses auxiliary space due to the recursion and temporary arrays created during the recursion which approximately holds the $O(n)$ space. This additional space is necessary to hold intermediate values while sorting and merging. Despite this, it is a stable sorting algorithm, ensuring that the relative order of elements with equal values remains unchanged in the sorted output.

Mergo sort algorithm code is already given in **ft_lstsort.c** file.

3.2 Bubble Sort

Bubble sort is a sorting algorithm, which compare the two number and swap those of a certain condition occur. It traverse whole array and compare the numbers which next to each other. Wanted order in the homework is ascending order, so if the previous number is bigger than current number, those should be swapped. This algorithm has a time complexity of $O(n^2)$. Rather than the merge sort, bubble sort don't use auxiliary space, and it is a stable sorting algorithm just like merge sort. The advantages like no additional memory required in sorting, make bubble sort more efficient in small datasets.

In the homework, I implement a bubble sort algorithm in **ft_lstsort_asm.s** file. As mentioned in the homework document, head of linkedlist is always defined in **0x20000004** address.

If we assume that **0x20000008** holds the value **0x2000000C**. linkedlist ca nexplain like that:

$$\texttt{[0x20000004]} \rightarrow \texttt{val}$$
$$\texttt{0x20000004} + \#4 \rightarrow \texttt{next}$$

$$\texttt{[[0x20000008]]} \rightarrow \texttt{next->val}$$
$$\texttt{[0x20000008]} + \#4 \rightarrow \texttt{next->next}$$

I will take current node's value and next in two register, and next node's value and next in different two register. After that loading operations, I made a compare operation between two node's values, if previous one is bigger than next one an swapping operation will occur. To make swapping operation, a temporary register should be used due to protect value of node. This process will continue until the end of the linkedlist, end of the linkedlist understood by the address of current node, if current node's address labeled as **0**, that there is no other meaningful value, and stop the while. These operations must be repeated as long as the list is length to ensure linkedlist totally sorted.

4 BigO Analysis

The hypothesis that merge sort is better than bubble sort is conceivable because of algorithms' time complexities, merge sort has $O(n*log(n))$ and bubble sort has $O(n^2)$ time complexity.

In the **entry.s** file, a DCD array is given. The elements of this DCD array sent to the **main.c** file will be sent to both mergesort and bubblesort algorithms, increasing by 5 in each loop, those operations' time measured with SysTick timer. The results of sorting the array with each input length from 5 to 100 are calculated using the **main.c** file as follows:

Table 1: Measured Sorting Times

Array Size	Merge Sort	Bubble Sort
5	5	1
10	E	6
15	18	F
20	22	$1B$
25	$2E$	$2B$
30	$3A$	$3E$
35	45	56
40	52	$6F$
45	60	$8E$
50	$6F$	AF
55	$7A$	$D3$
60	89	FC
65	97	126
70	$A5$	156
75	$B3$	187
80	$C1$	$1BE$
85	$D2$	$1F8$
90	$E1$	234
95	$F1$	274
100	101	$2B8$

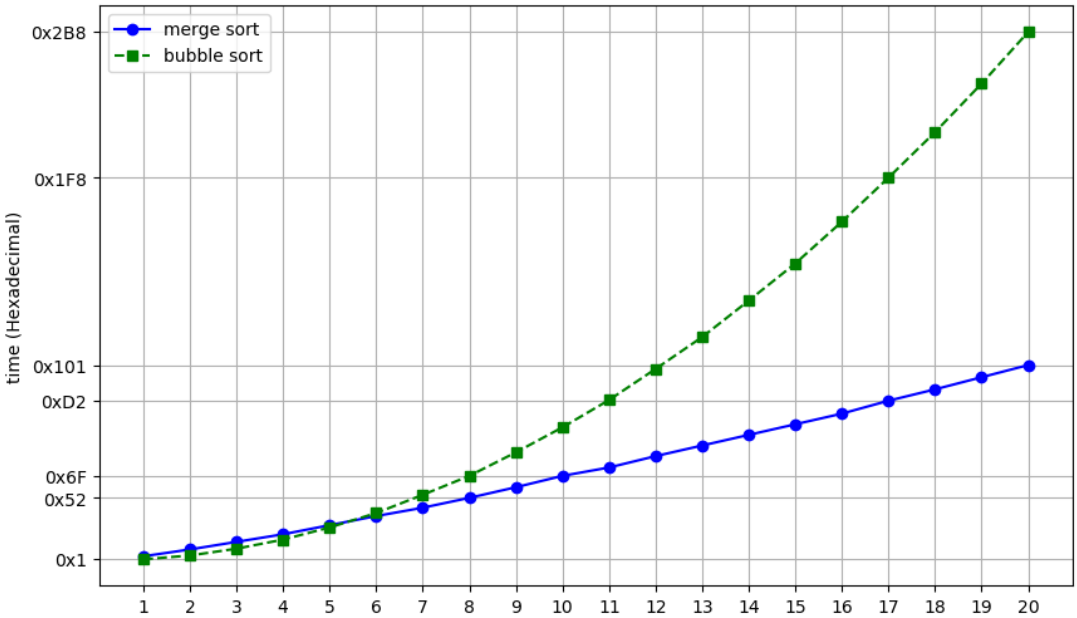

Figure 1: Time Complexity Comparison Table for Bubble and Merge Sort

As can be seen from this table and graph, using bubble sort is better than using merge sort for short arrays, because the cost of creating temporary arrays is high and this slows down the process. However, for long arrays, the advantage

of merge sort is incredibly high, because the cost of traversing the array exceeds the cost of creating temporal arrays.

5 Conclusion

In conclusion, this homework has allowed me to explore the differences between the bubble sort and merge sort algorithms through practical implementation and time complexity analysis. While bubble sort is advantageous for small datasets due to its simplicity and lack of auxiliary space requirements, merge sort's efficiency shines with larger datasets due to its $O(n \log n)$ time complexity. The experiments conducted, using SysTick timers to measure sorting times, demonstrated that bubble sort outperforms merge sort for smaller arrays, as the overhead of creating temporary arrays in merge sort becomes a bottleneck. However, as the array size increases, merge sort's performance becomes significantly superior, handling larger datasets much more efficiently. Ultimately, the choice of sorting algorithm should depend on the size of the dataset, with bubble sort suitable for small arrays and merge sort preferred for larger ones.

REFERENCES