CS 201(Fall 2023) Homework 2

Eda Alparslan/22102234

1.0 Implementation

First, we should implement the algorithms for this assignment. As you can see from the submitted .cpp file, I used the implementations that are given in slides.

2. Analysis of The Algorithms

Analysis of the mentioned algorithms are as follows:

2.1 Analysis of Merge Sort

In the worst case, there will be k = $2^0 * (2 * 2^{n-1} - 1) + 2^1 * (2 * 2^{n-2} - 1) + \ldots + 2^{n-1} * (2 * 2^0 - 1)$ comparisons. So the complexity of the worst case is O(n*$\log_2$n). By the same calculations, time complexity for the average case is found to be O(n*$\log_2$n), which is the same for worst case. This algorithm needs to allocate an extra array. (In my implementation, an automatically allocated array.)

2.2 Analysis of Quick Sort

Also uses divide and conquer technique like Merge Sort. I chose the pivot from the middle of array in my implementation. Worst case for this algorithm is that the array is already sorted, and the time complexity for this situation is O(n^2). Best and average cases have the same time complexity which is O(n*$\log_2$n). Implementation does not allocate any arrays.

2.3 Analysis of Bubble Sort

The worst case for bubble sort is that if the array is in reverse order and the time complexity is O(n^2). Best case is O(n) and average case is O(n^2). Function does not allocate an array in its implementation.

2.4 Conclusion

According to the evaluations above, the best choice for the library's problem would be Quick Sort. As we can obtain from the data we had from the experiments conducted, the worst choice would be Bubble Sort. Even if the time complexity is similar for Merge Sort and Quick Sort in larger array sizes, memory usage of Merge Sort is more, so Quick Sort would be better to use.

3. New Insights

As one can see from the tables below, Quick Sort is still the best option to use. (Even though it is slow with a sorted array, only %10 randomness helps it work.)

Bubble sort is better in this scenario but still is not the best. Merge sort is good but not as good as Quick Sort.

Memory usage (number of arrays allocated) for ALL Algorithms (Because it is the same outcome for all kind of arrays, it is just dependent on size. This is basically 3 same tables in short.)

n	Bubble Sort	Merge Sort	Quick Sort
2^4	0	Log 2^4	0
2^5	0	Log 2^5	0
2^6	0	Log 2^6	0
2^7	0	Log 2^7	0
2^8	0	Log 2^8	0
2^9	0	Log 2^9	0
2^{10}	0	Log 2^{10}	0
2^{11}	0	Log 2^{11}	0
2^{12}	0	Log 2^{12}	0
2^{13}	0	Log 2^{13}	0

Descending Array Time Table

n	Bubble Sort	Merge Sort	Quick Sort
2^4	0	0	0
2^5	0	0	0
2^6	0	0	0
2^7	0	0	0
2^8	0	0	0
2^9	0,0033	0,0021	0
2^{10}	0,0062	0,0021	0
2^{11}	0,0257	0,0021	0,0021
2^{12}	0,0785	0,0022	0,0021
2^{13}	0,4203	0,0022	0,0038

Ascending Array Time Table

n	Bubble Sort	Merge Sort	Quick Sort
2^4	0	0	0
2^5	0	0	0
2^6	0	0	0
2^7	0	0	0
2^8	0	0	0
2^9	0	0	0
2^{10}	0	0	0
2^{11}	0	0	0,0036
2^{12}	0	0,0017	0,0110
2^{13}	0	0,0036	0,0416

Random Array Time Table

n	Bubble Sort	Merge Sort	Quick Sort
2^4	0	0	0
2^5	0	0	0
2^6	0	0	0
2^7	0,0001	0	0
2^8	0,0003	0,001	0
2^9	0,0015	0,001	0
2^{10}	0,0047	0,002	0
2^{11}	0,0214	0,003	0,0001
2^{12}	0,0862	0,003	0,0020
2^{13}	0,3328	0,011	0,0024