

Program 3

Due Nov 17, 2018 by 11:59pm **Points** 30 **Submitting** a text entry box or a file upload
Available Oct 24, 2018 at 8pm - Nov 18, 2018 at 12:10am 24 days

This assignment was locked Nov 18, 2018 at 12:10am.

Sorting - Improved Mergesort

Purpose

Although merge sort shows the same execution complexity as quick sort, (i.e., $O(n \log n)$), its practical performance is much slower than quick sort due to array copying operations at each recursive call. This programming assignment improves the performance of merge sort by implementing a non-recursive, semi-in-place version of the merge-sort algorithm. You will learn how to estimate the complexity of an algorithm through computer simulation.

In-Place Sorting

In-place sorting is to sort data items without using additional arrays. For instance, quicksort performs partitioning operations by simply repeating a swapping operation on two data items in a given array, which thus needs no extra arrays.

On the other hand, mergesort we have studied allocates a temporary array, sorts partial data items in that array, and copies them back to the original array at each recursive call. Due to this repetitive array-copying operations, mergesort is much slower than quicksort although their running time is upper-bounded to $O(n \log n)$.

It has been an research interest to develop in-place mergesort algorithms, almost all of which are however impractically complicated. Yet, we can improve the performance of mergesort with adding the following two restrictions:

1. Using a non-recursive method (use iterative method)
2. Using only one additional array, (i.e., a semi-in-place method). Merge data from the original into the additional array at the very bottom stage, and thereafter perform the next merge from the additional into the original array, in which way merge operations are applied to the original and additional array in turn as you go through each repetitive stage. In other words, at the first merge, data is from original to additional array. And the next merge, data is from additional to original array, etc.

Note that we still need to allow data items to be copied between the original and this additional arrays as many times as $O(\log n)$.

Statement of Work

1. Design and implement a non-recursive, semi-in-place version of the merge-sort algorithm. The framework of your mergesort function will be :


```
#include <vector>
#include <math.h> // may need to use pow( )
using namespace std;

template <class Comparable>
void mergeImproved( vector<Comparable> &a ) {

    int size = a.size( );
    vector<Comparable> b( size ); // this is only one temporary array.

    // implement a nonrecursive mergesort only using vectors a and b.
}
```

Needless to say, the above *mergeImproved()* function must not call itself or some other recursive functions. Furthermore, the algorithm should still be based on the same divide-and-conquer approach.

2. Use the driver file (Files-->Program3/[driver.cpp](#) ) to verify and evaluate the performance of your non-recursive, semi-in-place mergesort program. The code below assumes that your program is written in the "mergeImproved.cpp" file.
3. You can use the usual mergesort and quicksort from the Files->Programs/Program3/ (or Sample codes folder)
4. Modify the driver file to compare the performance among the usual quicksort, the usual mergesort, and your improved mergesort as increasing the array size. You will need a loop that increase the array size by 20 each until size=1,000. Initial size is 20.
5. Use Excel to draw a graph comparing the three algorithms, and estimate its Big O. You will need to draw $O(n \log n)$, $O(n)$, and $O(\log n)$ to estimate its Big O.

What to Turn in

Clearly state in your code comments any other assumptions you have made. Turn in:

- (1) your nonrecursive, semi-in-place mergesort program, (i.e., template <class Comparable> void mergeImproved(vector<Comparable> &a) in "mergeImproved.cpp".) (**Don't use different function name or file name!**)
- (2) your modified driver.cpp that include a function that compares performance of merge, quick and mergeImproved algorithms.
- (3) a separate report in .doc or .docx that must includes:
 - (a) One-page output of your improved mergesort program (when #items = 30), and

- (b) A chart that compares the performance among the usual quicksort, the usual mergesort, and your improved mergesort, as well as plots of $n \log n$, n , $\log n$.
- (c) Estimated Big O for each algorithm.

Grading Guide and Answers

Check the following grading guide to see how your homework will be graded. Key answer will be made available after the due date through [Solution](#) page.

Program 3 Grade Guideline

1. Documentation (10 pts)

One page output (a.out 30 will fit one page.)

Correct(2pts) 1 ~ 2 errors(1pt) 3+ errors or no results(0pt)

Performance comparison between your algorithm and ordinary merge/quick sorts

All three plots are given and Big-O is estimated using $n \log n$, n , and $\log n$ plots (total 6 plots). Comparison results are clearly stated.(8 pts)

No chart: 0 pts

Each of the following cases be taken off 2 points :

- No Big-O estimation or incorrect estimation
- Less than 6 plots
- No statement for comparison results

2. Correctness (16 pts)

Compilation errors(0pt)

Successful Compilation(4 pts)

+ Non-recursive method(2 pts)

+ Only one additional array besides the original array passed from main(2 pt)

+ One way assignment from one to another array in each iteration (2 pts)

+ Your algorithm is still based on the same divide-and-conquer approach(2 pts)

+ A new loop is added so that the size can be increased from 20 to 1,000 [increment by 20]

(2 pts)

+ Your new algorithm is clearly improved compared to normal mergesort and quicksort (2 pts)

3. Program Organization (4pts)

You must write a plenty of comments to help the professor or the grader understand your code.

Proper comments

Good (2pts) Poor(1pts) No explanations(0pt)

Coding style (proper identations, blank lines, variable names, and non-redundant code)

Good (2pts) Poor(1pts) No explanations(0pt)