Course: ESO207A – Data Structures and Algorithms

Indian Institute of Technology Kanpur

Programming Assignment 5 : *Quick sort versus Merge Sort*

Most Important guidelines

- It is only through the assignments that one learns the most about the algorithms and data structures. You are advised to refrain from searching for a solution on the net or from a notebook or from other fellow students. Remember - **Before cheating the instructor, you are cheating yourself**. The onus of learning from a course lies first on you. So act wisely while working on this assignment.
- Refrain from collaborating with the students of other groups. If any evidence is found that confirms copying, the penalty will be very harsh. Refer to the website at the link: https://cse.iitk.ac.in/pages/AntiCheatingPolicy.html regarding the departmental policy on cheating.
- This assignment has to be done in groups of 2 only. It is your responsibility to find a partner.
- **You need to upload 2 files. The first file will be a pdf file for the report that has the table, the graphs, the plots, and answers of all the questions of the assignment. The other file will have the C code that has the implementation of 3 algorithms as functions. Please follow the naming convention <RollNo1>-<RollNo2>-A5.pdf, <RollNo1>-<RollNo2>-A5.c while submitting your assignmment on Moodle. The code for quicksort, mergesort and improved merge sort should be in form of function and neatly separated by comments. For example: 210449-210733-A5.pdf and 210449-210733-A5.c. You may use any word processor (latex, word, ...) to prepare the report file. However, no scanned copy of a handwritten submission is allowed as report.**
- Both the students in a group must submit the same files on Moodle.
- In case of any issue related to this assignment, send an email at antreev@cse.iitk.ac.in (and not to the instructor) with the email subject [ESO207]
- The assignment is fairly simple but can be implementation heavy, so you better start early as there can't be any extensions.

The Objective of the Assignment

The followings are the objectives of this assignment.

1. To investigate whether Quick sort has better/poor performance than Merge sort in reality ?
2. To investigate how often Quick sort deviates from its average-case behavior in reality ?

Background of Assignment

We know that the average number of comparisons during Quick Sort on n distinct numbers is nearly 40% more than Merge sort. Quick Sort can take $\Theta(n^2)$ comparisons in the worst case whereas Merge Sort always perform $O(n \log n)$ comparisons. Given these facts and the fact that Quick Sort was invented after Merge Sort, why does C library have an implementation of Quick Sort but not that of Merge sort ? Is it not puzzling. Ponder over it deeply for sufficient time ...

Tasks to be done

The aim of this part is to compare Quick sort with Merge sort. For this purpose, you have to empirically compare the efficiency parameters of the quick sort and merge sort on a sequence of n uniformly randomly generated numbers for various values of n. Fill up the following tables on the basis of your experimental results. The C code has to be submitted along with the assignment.

1 Quick Sort versus Merge Sort

Notes for this section:

- Number of iterations per value of n: 500
- Units of time is microseconds (μs)

1.1 Comparisons

$n \longrightarrow$	10^2	10^3	10^4	10^5	10^6
Average number of comparisons during Quick Sort					
$2n \log_e n$					
Average number of comparisons during Merge Sort					
$n \log_2 n$					

This has to be followed by a concise inference (not exceeding 4 sentences).

1.2 Number of comparisons and time complexity of quick sort

Plot $T_q(n)$ versus $n \log_{10} n$ for the values in the table given below, where $T_q(n)$ is the average running time of quick sort on input of size n.

$n \longrightarrow$	10^5	$3 * 10^5$	$5 * 10^5$	$7 * 10^5$	$9 * 10^5$
Average running time of Quick Sort					

Based on the plot, provide a concise inference (not exceeding 2 sentences).

1.3 Time Complexity

Using the same code compare the average running time taken by merge sort and quick sort. Put on your Sherlock's Deerstalker Hat, and try to infer something strange from the results

Draw the matplotlib plots for $\log_{10}$(running time) vs $\log_{10}$(array size) for both algorithms on same graph.

$n \longrightarrow$	10^2	10^3	10^4	10^5	10^6
Average running time of Quick Sort					
Average running time of Merge Sort					
Number of times Merge Sort outperformed Quick Sort					

Based on the plot, provide a concise inference (not exceeding 2 sentences).

1.4 Can you improve merge-sort ?

So far, we have evaluated results for number of comparisons and running time of these two sorting algorithms. Can you make simple changes in the implementation of merge sort to improve its running time ? Please note that you are supposed to make simple changes and not drastic changes. In particular, your code should still employ $O(n)$ extra space. *Hint - Try to get rid of redundant overheads in the*

implementation. If you look carefully, you can spot them very easily. Do not waste time searching for it on www.

Evaluate the running time again of the quicksort with improved-merge-sort. Also, draw the same plots as previous section.

$n \longrightarrow$	10^2	10^3	10^4	10^5	10^6
Average running time of Quick Sort					
Average running time of Improved-Merge-Sort					
Number of times Improved-Merge Sort outperformed Quick Sort					

2. Reliability of Quick Sort

The aim of this part is to empirically find out the deviation from the average running time of the quick sort for various values of n. You will have to enter your experimental results in the table given below. This has to be followed by a concise inference (not exceeding 4 sentences).

$n \longrightarrow$	10^2	10^3	10^4	10^5	10^6
Average running time of Quick Sort					
No. of cases where run time exceeds average by 5%					
No. of cases where run time exceeds average by 10%					
No. of cases where run time exceeds average by 20%					
No. of cases where run time exceeds average by 30%					
No. of cases where run time exceeds average by 50%					
No. of cases where run time exceeds average by 100%					

Note

- Number of iterations per value of n: 500
- Units of time is microseconds (μs)

Important points you should consider in this assignment

1. n has been kept sufficiently large at 500.
2. For random number generator, we will use rand() function. In the given code snippet, a single random number is geenrated between 0 - MAX_RANGE. This has to be used to populate the input array

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>

int main() {
    srand((unsigned)time(NULL));
    int MAX_RANGE = 50;
    double random_num = ((double) rand() / (double) RAND_MAX)*MAX_RANGE;

    ...
}
```

3. Precise time has been measured in microseconds the given code snippet. The C library function clock_t clock(void) returns the number of clock ticks elapsed since the program was launched. To get the number of seconds used by the CPU, you will need to divide by CLOCKS_PER_SEC.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>

int main() {
    ...

        clock_t t;
        clock_t time_taken;
        t = clock();

        // SOME COMPUTATION

        time_taken = (clock() - t);

        int time_taken_microseconds = ((double)total_time_taken*1000000)/
                                      (double)CLOCKS_PER_SEC;
    ...
}
```

4. Instead of using stdout to print the outputs on console and later copying them to create the plots, it might be better to redirect the output to a file. Here is a simple C code to do it.

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{

   FILE *fptr;
   fptr = fopen("./output.txt","w");

   if(fptr == NULL)
   {
      printf("Error! File doesn't exist");
      exit(1);
   }

   int num = 1;
   fprintf(fptr,"%d\n",num);
   fclose(fptr);
   return 0;
}
```

5. Good programming practices like indentation, suitable variable names, functions for repititive tasks have been kept in mind while forming the model solutions.

6. Quicksort and Merge sort have to implemented on your own.

7. Since every operation has to be performed atleast 500 times, it might take 2-3 minutes to run get results for large arrays like 10^5, it would be convenient if you can redirect the outputs in a file and move the process to background.