Project 4 Sorting

Sorting is one of the most fundamental algorithmic problems within computer science. It has been claimed that as many as 25% of all CPU cycles are spent sorting, which provides a lot of incentive for further study and optimization of sorting. In addition, there are many other tasks (searching, calculating the median, finding the mode, remove duplicates, etc.) that can be implemented much more efficiently once the data is sorted. Your next tasks are to do some exploration into sorting algorithms.

Your tasks are:

1. Implement the BUBBLE SORT algorithm, QUICK SORT algorithm and MERGE SORT algorithm.
   You have to read the input numbers from the file "input.txt" and redirect the output to the file "output.txt". You should generate the file input.txt by yourself which contains at least 1000 randomly generated integers.

2. Modify the direct insertion sort into binary insertion sort.

3. Analysis the time complexity of above algorithms. And check your time complexity analysis by timing the performance of above functions by measuring elapsed system time.
   To measure the performance of a function, we may use C's standard library **time.h** as the following:

```
#include <time.h>
clock_t start, stop; /* clock_t is a built-in type for processor time (ticks) */
double duration; /* records the run time (seconds) of a function */
int main ( )
{ … …
/* clock() returns the amount of processor time (ticks) that has elapsed
        since the program began running */
start = clock(); /* records the ticks at the beginning of the function call */
sort_function(); /* run your function here */
stop = clock(); /* records the ticks at the end of the function call */
duration = ((double)(stop - start))/CLK_TCK;
      /* CLK_TCK is a built-in constant = ticks per second */
      … …
      return 1;
}
```

Note: If a function runs so quickly that it takes less than a tick to finish, we may repeat the function calls for *K* times to obtain a total run time, and then divide the

total time by K to obtain a more accurate duration for a single run of the function. The repetition factor must be large enough so that the number of elapsed ticks is at least 10 if we want an accuracy of at least 10%.