Birla Institute of Technology & Science, Pilani
Data Structures & Algorithms (CS F211)
Lab – 4 (Code Profiling)

Problem –

Code and profile various sorting techniques for a sufficiently large dataset. Consider following algorithms –

1) Insertion Sort
2) Merge Sort
3) Quick Sort with pivot element selected as
   a) First element in the list
   b) Median of three (median of first, mid and last element)
   c) Median of median [take home assignment]
   d) Quick-select [take home assignment]

Datasets –

Input →

Generate a file, name as *input.txt*, with 10^7 (1 billion) integers, with each integer written on a line. You may use the script given in Appendix for data generation.

Output →

Generate a file, name as *output.txt* and write the sorted output.

Execution –

- Profile each of the above mentioned sorting algorithms against data size of 10^4, 10^5, 10^6, 10^7 and 10^8. Last two data sizes can be tried as take home assignment.
- Data should be read in an array and this array should be passed to sorting function.
- There should be a separate executable for each sorting algorithm mentioned above.
- User should only give data size as command line argument. Given data size should be read in from *input.txt* by calling *file_read()* function and sorted data should be written (over written) to *output.txt* by calling *file_write()* function.

Deliverables–

1) Source Code

- `driver.c` → should have following functions along with call to sorting function
  - `int main(int argc, char *argv[])`
  - `int* file_read(char *input_file_name, long int N)`
  - `void file_write(char *output_file_name, int* data, long int N)`
- `insertion.c` → all functions related to insertion sort
- `merge.c` → all functions related to merge sort
- `quick.c` → all functions related to quick sort
- `pivot.c` → all functions for pivot selection

2) Executable Files →

- `insert` (should run as ./insert 100000)
- `merge`
- `quickFirst`
- `quickThree`
- `quickMedian`
- `quickSelect`

3) Helper Files →

- `Makefile` → object file based compilation commands
- `Profile.csv` → table containing time taken by each sorting strategy as rows and data size as column

Appendix

1) Data set generation script –

- Create a file *data_generator.sh* and write following code –

```
for i in $(seq $1)
do
     echo $RANDOM
done
```

- Set it permission to be executable

```
chmod +x data_generator.sh
```

- Execute it by passing required data size as argument (Don't try to generate data size bigger than 10 Million in lab)

```
./data_generator.sh 10000000 > input.txt
```

2) Profiling a code (sample.c) using gprof –

- Compile using –pg flag

```
gcc –pg sample.c –o sample
```

- Execute the code as usual, with command line arguments, if required

```
./sample 10000
```

- Generate profiling output

```
gprof –b sample gmon.out > profile.txt
```

- Note down the time taken by required functions from *self seconds* column in front of the function name in flat profile in *profile.txt* file.