# Assignment – 5

The goal of this assignment is to get students familiar with gprof and how various arguments affect the execution time of the program.

## gprof

The gprof profiler, the standard for gcc, collects call counts and time intervals for functions. Only user-CPU time is recorded, and the timing information is gathered by waking up the profiler at random intervals and reporting what function is currently being executed. This kind of time measurement is only an approximation, thus making it unreliable for individual program executions. Thus, timing information might require averaging several programing executions to be useful.

## Using gprof

Using gprof is simple:

- Compile with the -pg flag.
- Run the program. The gmon.out file is overwritten with profiling data. Note: move gmon.out to another file if you plan to do multiple runs, e.g. gmon.1.
- Run gprof to view human-readable results.
- Alternatively, run gprof -s to incorporate a series of program executions, e.g. gprof -s a.out gmon.1 gmon.2. Now gprof a.out gmon.sum shows the summary.

## Reading the output

The gprof output starts with a flat profile that contains:

- Self seconds reports how much time is spent in a subroutine.
- Calls reports the number of times a subroutine is invoked.
- Self ms/call reports the average time spent only in a particular subroutine.
- Total ms/call reports the time spent in subroutines called by that subroutine as well (only a guess).

## Exercise

Write a C program. Name the file "bad_implementation.c". The file will contain 3 functions.

1. int is_prime(int n)
   Function to check if n is prime or not. Implement naïve implementation. Use for loop from x = 2 to x = n – 1 and check if x divides n or not
2. int* random_array(int n, int maxVal)
   This function returns a array of size n with random values in range [1, maxVal]

3. int largest_prime(int n, int *arr)
   This function should return the largest prime number in arr[]. n is the size of arr[]. This function should loop through all values in arr[] and check if arr[i] is prime or not using is_prime(int) and then return the largest prime. Return -1 if no prime number is found.

Steps in program

1. Read size of array and range of random values as command line arguments.
2. Generate the random array using random_array(int, int).
3. Use largest_prime(int n, int *arr) to print the largest prime number in randomly generated array.

Perform the following steps on the program:

1. Compile bad_implementation.c for gprof.
2. Do a few experiments to develop some intuition for how the program's execution time varies with the size of the array. Use large values for size of array and observe how the gprof output changes. Write your observations in the report and also give reason for it. Perform for at least 5 different array sizes. Fix the range of random array at 100000.
3. Next fix the array size at N = 1000000 and vary the range of random array. Write observations and also give reason for the observations. Perform this for at least 5 different ranges.

After that write another program "efficient_implementation.c". This program is identical to "bad_implementation.c" but in this program you need to use more efficient algorithm to find prime number.

After that perform the experiment on "efficient_implementation.c" that you performed on "bad_implementation.c".

What to Submit?

1. bad_implementation.c file
2. efficient_implementation.c file
3. Report

The report should contain the output of gprof in table format along with your observations and reason why output of gprof changes with changing the arguments for both "bad_implementation.c" and "efficient_implementation.c". Also, plot a graph for largest_prime(int, int*) function – argument (as x-axis) vs time spend in the subroutine (as y-axis).

Also attach gprof flat profile screenshots in the lab report as proof.