CS3811 - High Performance Computing and Big Data Lab

Lab 3

> Name: M K Lokesh Kumar

> Registration No.: 2201113026

> Class: Cyber Security(Semester 5)

Experiment 1

Objective

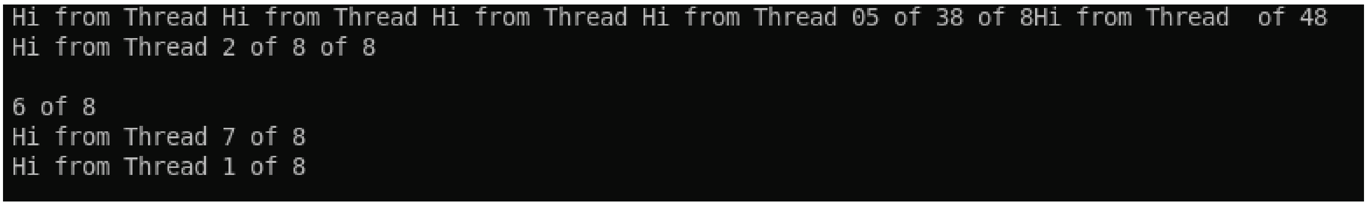
To run Hello World for OpenMP, the first parallel OpenMP code and change the values of basic function calls in OpenMP.

Code

Written in C++.

```
#include <iostream>
#include <omp.h>

using namespace std;

void Hello(int my_id, int total_threads) {
    cout << "Hi from Thread " << my_id << " of " << total_threads << endl;
}

int main() {
    omp_set_num_threads(8);
    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        int nt = omp_get_num_threads();
        Hello(id, nt);
    }

    return 0;
}
```

Output



```
Hi from Thread Hi from Thread Hi from Thread Hi from Thread 05 of 38 of 8Hi from Thread  of 48
Hi from Thread 2 of 8 of 8

6 of 8
Hi from Thread 7 of 8
Hi from Thread 1 of 8
```

Experiment 2

Objective

Write a C/C++ program to time the code for estimation of teh value of Pi using different OpenMP code snippets(1 serial code + 4 parallel attempts)

Code

Writen in C++.

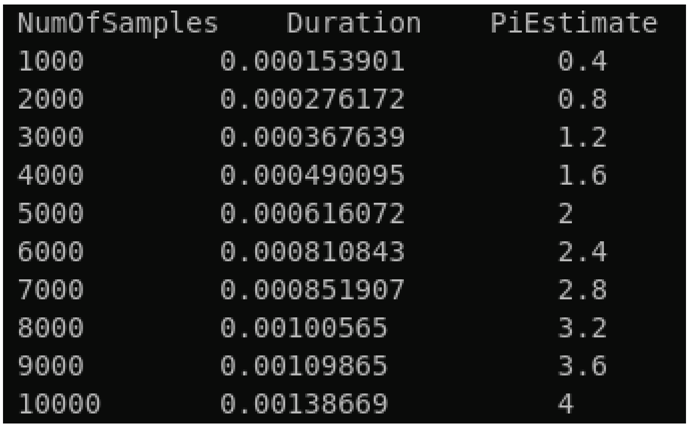
```

#include <iostream>
#include <random>
#include <cmath>
#include <omp.h>
#include <fstream>

# define RADIUS 10.0
# define N_SAMPLES 10000

#define NUM_THREADS 8

using namespace std ;
float get_rand_num() {
    static random_device rd;
    static mt19937 gen(rd());
    static uniform_real_distribution<> dis(0.0, 1.0);

    return dis(gen);
}

int main()
{
    ofstream outfile("data2_0.dat");
    outfile << "NumOfSamples" << "\t" << "Duration" << "\t" << "PiEstimate"
<< endl;
    for (size_t j = 1000; j <= N_SAMPLES; j += 1000) {
        float pi_estimated = 0.0;
        size_t inside_circle = 0;
        double start = omp_get_wtime();
        for (size_t i = 0; i < j; i++) {
            float x_ = get_rand_num();
            float y_ = get_rand_num();
            if (pow ((x_ * x_ + y_ * y_), 0.5) < RADIUS)
```

```
                inside_circle++;
        }
        double duration = omp_get_wtime() - start;
        cout << "Original value of pi is " << M_PI << endl;
        cout << "Number of samples is " << j << endl;
        pi_estimated = 4.0 * (inside_circle / (N_SAMPLES * 1.0));
        cout << "Time to estimate pi is " << duration << " s " << endl;

        outfile << j << "\t\t" << duration << "\t\t\t" << pi_estimated <<
endl;
    }

    outfile.close();

    // attempt 1
    ofstream outfile1("data2_1.dat");
    outfile1 << "NumOfThreads" << "\t" << "NumOfSamples" << "\t" <<
"Duration" << endl;
    for (int num_of_threads = 2; num_of_threads < NUM_THREADS;
num_of_threads += 2) {
        for (size_t j = 1000; j <= 10000; j += 1000) {
            float pi_estimated = 0.0;
            double start = omp_get_wtime();

            float A[num_of_threads] = {0};

            #pragma omp parallel num_threads(num_of_threads)
            {
                size_t inside_circle = 0;
                int thread_id = omp_get_thread_num();
                size_t samples_per_thread = j / num_of_threads;

                #pragma omp for
                for (size_t i = 0; i < samples_per_thread; i++) {
                    float x_ = get_rand_num();
                    float y_ = get_rand_num();
                    if (pow((x_ * x_ + y_ * y_), 0.5) < (RADIUS)) {
                        inside_circle++;
                    }
                }

                A[thread_id] = 4.0 * (inside_circle / static_cast<float>
(samples_per_thread));
            }

            for (int i = 0; i < num_of_threads; i++) {
                pi_estimated += A[i];
            }
            pi_estimated /= num_of_threads;

            double duration = omp_get_wtime() - start;

            outfile1 << num_of_threads << "\t\t" << j << "\t\t" << duration
<< endl;
```

```
        }
    }
    outfile1.close();

    // attempt 2
    ofstream outfile2("data2_2.dat");
    outfile2 << "NumOfThreads" << "\t" << "NumOfSamples" << "\t" << "Duration" << endl;
    for (int num_of_threads = 2; num_of_threads < NUM_THREADS; num_of_threads += 2) {
        for (size_t j = 1000; j <= 10000; j += 1000) {
            float pi_estimated = 0.0;
            double start = omp_get_wtime();

            double A[num_of_threads][4];

            #pragma omp parallel num_threads(num_of_threads)
            {
                size_t inside_circle = 0;
                int thread_id = omp_get_thread_num();
                size_t samples_per_thread = j / num_of_threads;

                #pragma omp for
                for (size_t i = 0; i < samples_per_thread; i++) {
                    float x_ = get_rand_num();
                    float y_ = get_rand_num();
                    if (pow((x_ * x_ + y_ * y_), 0.5) < (RADIUS)) {
                        inside_circle++;
                    }
                }

                A[thread_id][0] = 4.0 * (inside_circle / static_cast<float>(samples_per_thread));
            }

            for (int i = 0; i < num_of_threads; i++) {
                pi_estimated += A[i][0];
            }
            pi_estimated /= num_of_threads;

            double duration = omp_get_wtime() - start;

            outfile2 << num_of_threads << "\t\t" << j << "\t\t" << duration << endl;
        }
    }
    outfile2.close();

    // attempt 3
    ofstream outfile3("data2_3.dat");
    outfile3 << "NumOfThreads" << "\t" << "NumOfSamples" << "\t" << "Duration" << endl;
    for (int num_of_threads = 2; num_of_threads < NUM_THREADS; num_of_threads += 2) {
        for (size_t j = 1000; j <= 10000; j += 1000) {
            float pi_estimated = 0.0;
            double start = omp_get_wtime();

            #pragma omp parallel num_threads(num_of_threads)
            {
                size_t inside_circle = 0;
                int thread_id = omp_get_thread_num();
                size_t samples_per_thread = j / num_of_threads;

                #pragma omp for
                for (size_t i = 0; i < samples_per_thread; i++) {
                    float x_ = get_rand_num();
                    float y_ = get_rand_num();
                    if (pow((x_ * x_ + y_ * y_), 0.5) < (RADIUS)) {
                        inside_circle++;
                    }
                }

                #pragma omp critical
                {
                    pi_estimated = 4.0 * (inside_circle / static_cast<float>(samples_per_thread));

                }

                #pragma omp barrier
                {
                    if (thread_id == 0) {
                        pi_estimated /= num_of_threads;
                    }
                }
            }

            double duration = omp_get_wtime() - start;

            outfile3 << num_of_threads << "\t\t" << j << "\t\t" << duration << endl;
        }
    }
    outfile3.close();

    // attempt 4
    ofstream outfile4("data2_4.dat");
    outfile4 << "NumOfThreads" << "\t" << "NumOfSamples" << "\t" << "Duration" << endl;
    for (int num_of_threads = 2; num_of_threads < NUM_THREADS; num_of_threads += 2) {
        for (size_t j = 1000; j <= 10000; j += 1000) {
            float pi_estimated = 0.0;
            double start = omp_get_wtime();

            double A[num_of_threads][4];
            #pragma omp parallel num_threads(num_of_threads)
            {
                size_t inside_circle = 0;
                int thread_id = omp_get_thread_num();
                size_t samples_per_thread = j / num_of_threads;

                #pragma omp parallel for
                for (size_t i = 0; i < j; i++) {
                    float x_ = get_rand_num();
                    float y_ = get_rand_num();
                    if (pow((x_ * x_ + y_ * y_), 0.5) < (RADIUS)) {
                        A[i % num_of_threads][0] += 1.0;
                    }
                }

                for (int i = 0; i < num_of_threads; i++)
                {
                    pi_estimated = 4.0 * A[i][0] / (j * 1.0);

                }
            }

            double duration = omp_get_wtime() - start;

            outfile4 << num_of_threads << "\t\t" << j << "\t\t" << duration << endl;
        }
    }
    outfile4.close();

    return 0;
}
```

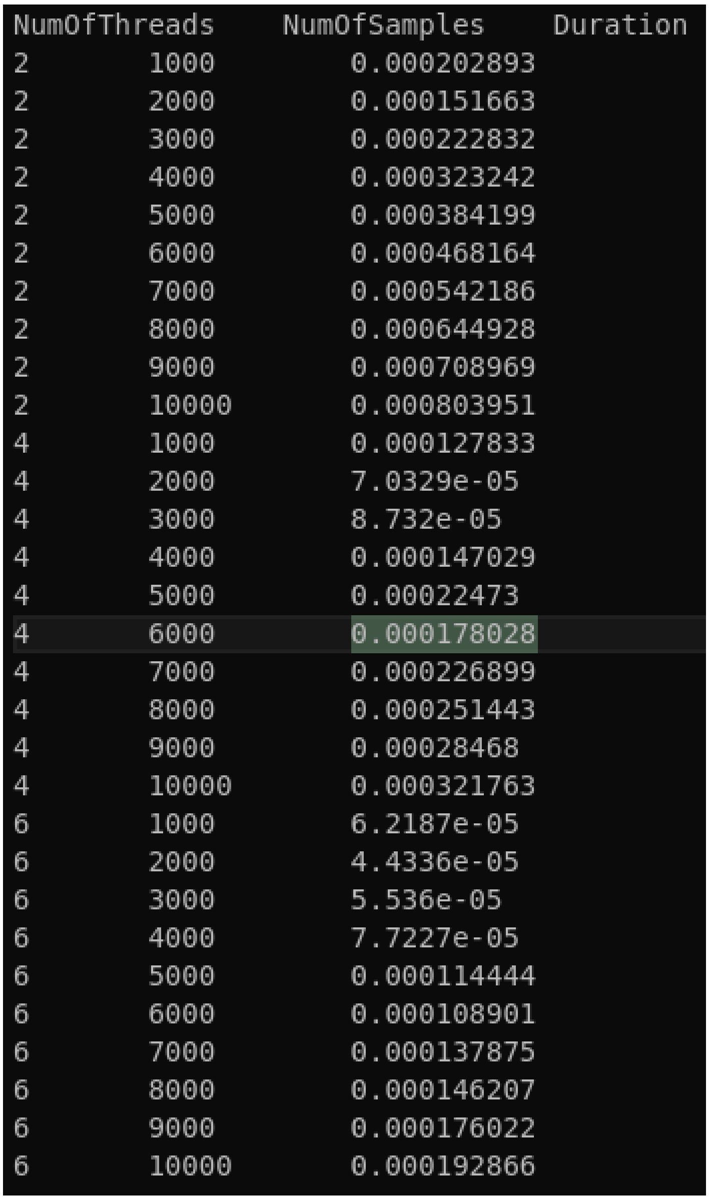
Output

- Serial code



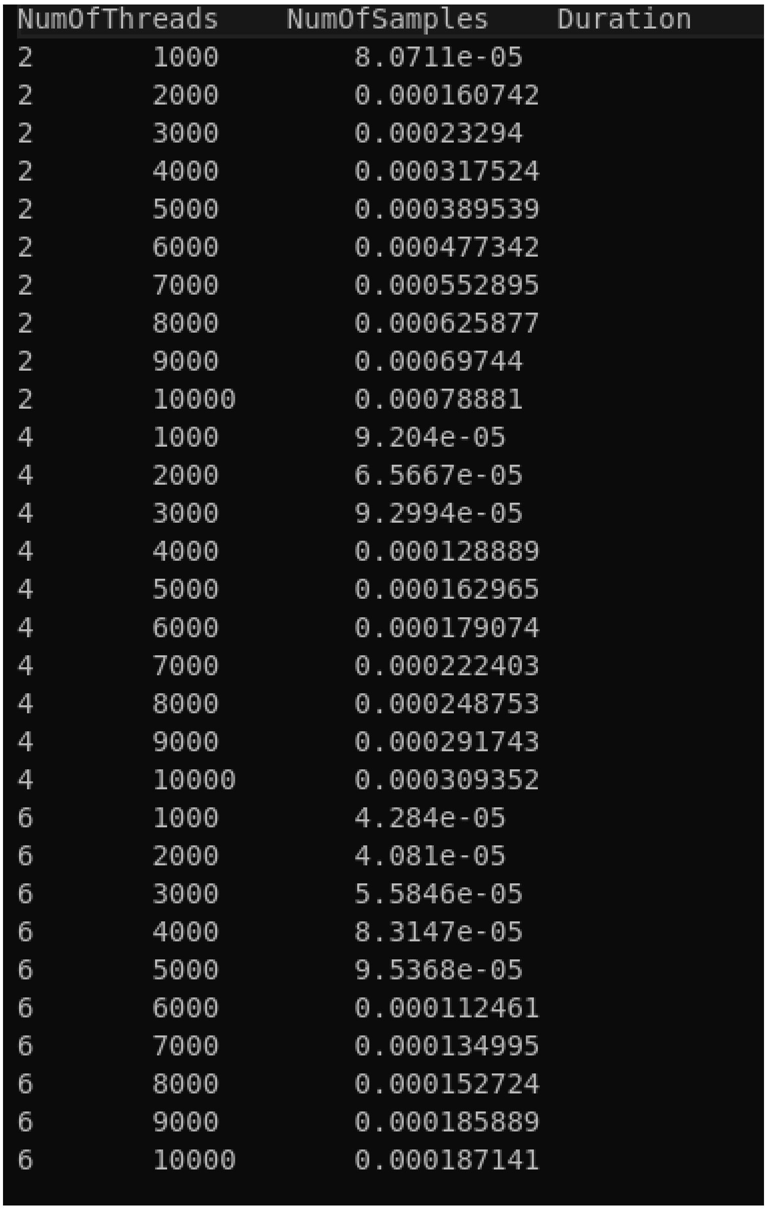
```
NumOfSamples     Duration       PiEstimate
1000         0.000153901        0.4
2000         0.000276172        0.8
3000         0.000367639        1.2
4000         0.000490095        1.6
5000         0.000616072        2
6000         0.000810843        2.4
7000         0.000851907        2.8
8000         0.00100565         3.2
9000         0.00109865         3.6
10000        0.00138669         4
```

- Attempt 1



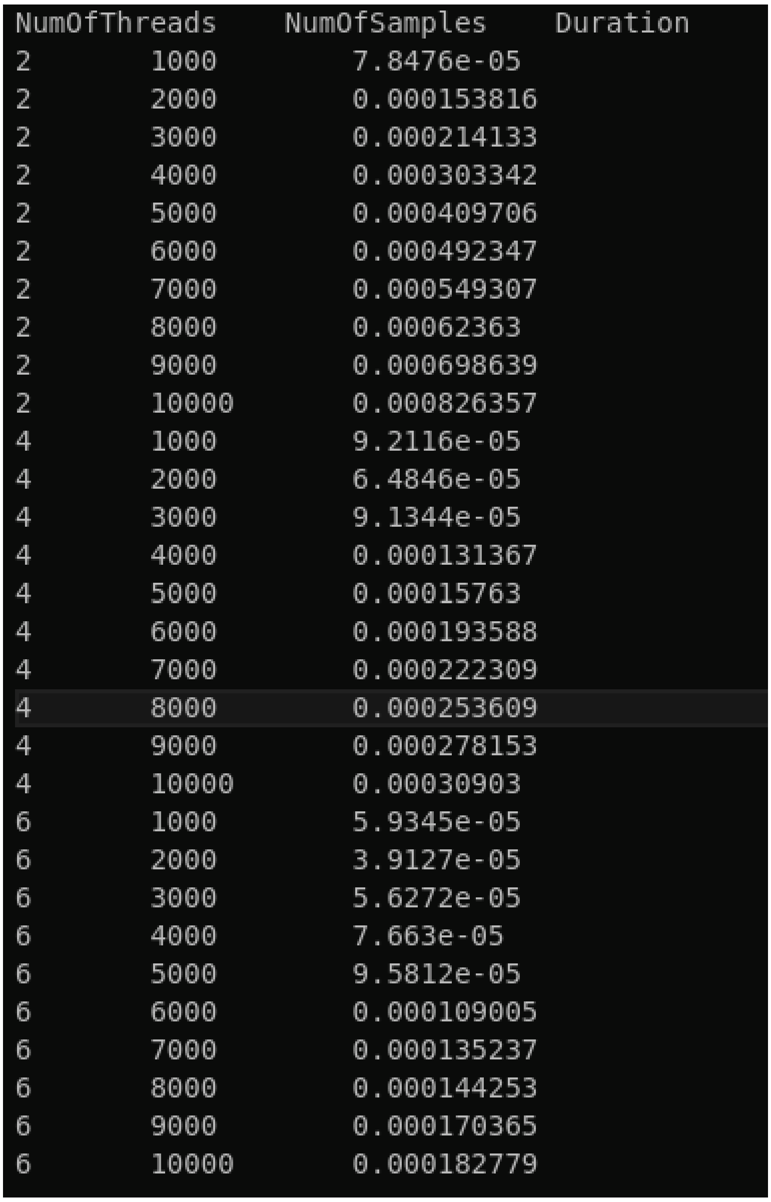
```
NumOfThreads     NumOfSamples     Duration
2        1000        0.000202893
2        2000        0.000151663
2        3000        0.000222832
2        4000        0.000323242
2        5000        0.000384199
2        6000        0.000468164
2        7000        0.000542186
2        8000        0.000644928
2        9000        0.000708969
2        10000       0.000803951
4        1000        0.000127833
4        2000        7.0329e-05
4        3000        8.732e-05
4        4000        0.000147029
4        5000        0.00022473
4        6000        0.000178028
4        7000        0.000226899
4        8000        0.000251443
4        9000        0.00028468
4        10000       0.000321763
6        1000        6.2187e-05
6        2000        4.4336e-05
6        3000        5.536e-05
6        4000        7.7227e-05
6        5000        0.000114444
6        6000        0.000108901
6        7000        0.000137875
6        8000        0.000146207
6        9000        0.000176022
6        10000       0.000192866
```

- Attempt 2



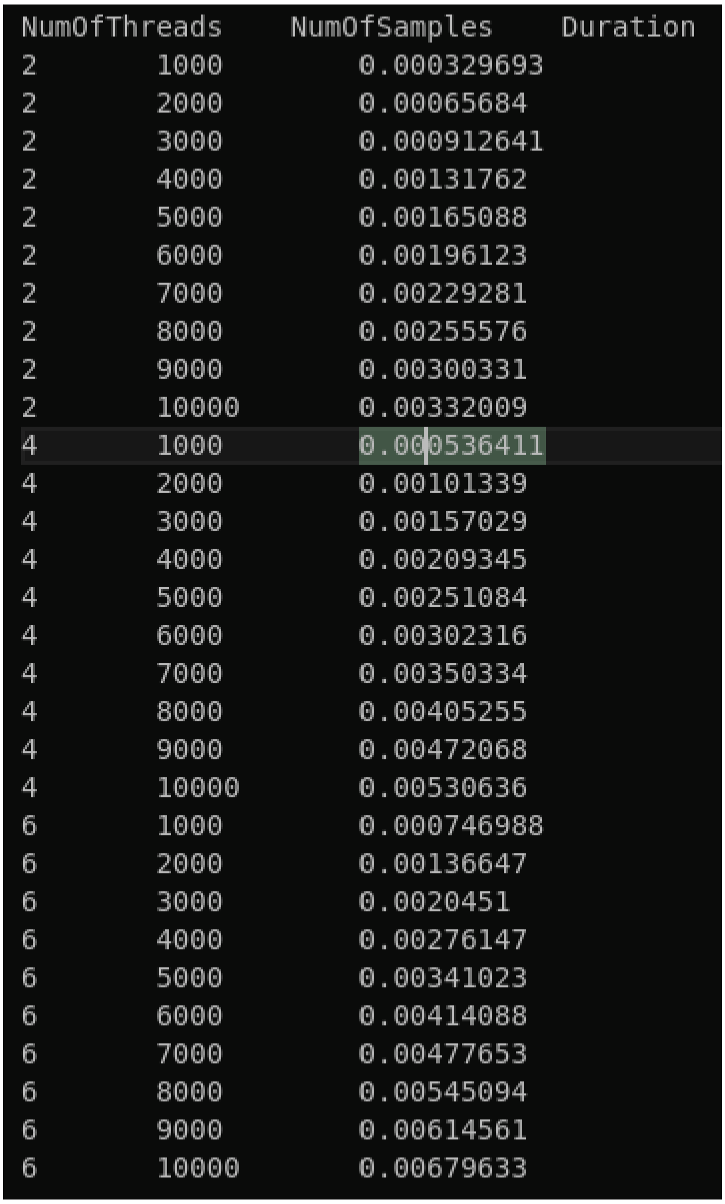
```
NumOfThreads     NumOfSamples     Duration
2       1000        8.0711e-05
2       2000        0.000160742
2       3000        0.00023294
2       4000        0.000317524
2       5000        0.000389539
2       6000        0.000477342
2       7000        0.000552895
2       8000        0.000625877
2       9000        0.00069744
2       10000       0.00078881
4       1000        9.204e-05
4       2000        6.5667e-05
4       3000        9.2994e-05
4       4000        0.000128889
4       5000        0.000162965
4       6000        0.000179074
4       7000        0.000222403
4       8000        0.000248753
4       9000        0.000291743
4       10000       0.000309352
6       1000        4.284e-05
6       2000        4.081e-05
6       3000        5.5846e-05
6       4000        8.3147e-05
6       5000        9.5368e-05
6       6000        0.000112461
6       7000        0.000134995
6       8000        0.000152724
6       9000        0.000185889
6       10000       0.000187141
```

- Attempt 3



```
NumOfThreads    NumOfSamples    Duration
2       1000        7.8476e-05
2       2000        0.000153816
2       3000        0.000214133
2       4000        0.000303342
2       5000        0.000409706
2       6000        0.000492347
2       7000        0.000549307
2       8000        0.00062363
2       9000        0.000698639
2       10000       0.000826357
4       1000        9.2116e-05
4       2000        6.4846e-05
4       3000        9.1344e-05
4       4000        0.000131367
4       5000        0.00015763
4       6000        0.000193588
4       7000        0.000222309
4       8000        0.000253609
4       9000        0.000278153
4       10000       0.00030903
6       1000        5.9345e-05
6       2000        3.9127e-05
6       3000        5.6272e-05
6       4000        7.663e-05
6       5000        9.5812e-05
6       6000        0.000109005
6       7000        0.000135237
6       8000        0.000144253
6       9000        0.000170365
6       10000       0.000182779
```

- Attempt 4



```
NumOfThreads    NumOfSamples    Duration
2       1000        0.000329693
2       2000        0.00065684
2       3000        0.000912641
2       4000        0.00131762
2       5000        0.00165088
2       6000        0.00196123
2       7000        0.00229281
2       8000        0.00255576
2       9000        0.00300331
2       10000       0.00332009
4       1000        0.000536411
4       2000        0.00101339
4       3000        0.00157029
4       4000        0.00209345
4       5000        0.00251084
4       6000        0.00302316
4       7000        0.00350334
4       8000        0.00405255
4       9000        0.00472068
4       10000       0.00530636
6       1000        0.000746988
6       2000        0.00136647
6       3000        0.0020451
6       4000        0.00276147
6       5000        0.00341023
6       6000        0.00414088
6       7000        0.00477653
6       8000        0.00545094
6       9000        0.00614561
6       10000       0.00679633
```