Course: ESO207A – Data Structures and Algorithms

Indian Institute of Technology Kanpur

Programming Assignment 1 : *Does the efficiency of algorithms really matter ?*

Important guidelines for submission

- 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.
- Refrain from collaborating with the students of other groups or your friends. 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.
- **The assignment is to be submitted on Gradescope. You must submit one C file named as *assign1.c*. This is EXTREMELY IMPORTANT. The code is checked with an autograder, and a different file name will fail to pass the test cases.**
- **DEADLINE** for this assignment submission is 11 PM Saturday (18th Jan).
- In case of any issue related to this assignment, send an email at shantanu@cse.iitk.ac.in

The Objective of the Assignment

The followings are the objectives of this assignment.

1. To investigate whether the efficiency of an algorithm really matters in real world ?
2. To verify how well the RAM model of computation captures the time complexity of an algorithm.

Tasks to be done

- n^{th} **Fibonacci Number** You have to implement two functions named

  1. *int RFib(int n)* : Calculates the n^{th} Fibonacci number modulo 2025 recursively and return it.
  2. *int CleverFib(long int n)* : Calculates the n^{th} Fibonacci number modulo 2025 by repeated squaring of a cleverly defined 2x2 matrix as shown in the lectures and return the value.

  While the test cases will check the correctness of your implementation, the idea of the assignment is for you to verify the efficiency of algorithms. Analyze and compare the asymptotic run-times for the above three functions by experimentally observing the different values of n that run within the time limit.

Time →	0.001 sec	0.1 sec	1 sec	5 sec	60 sec	600 sec
RFib						
CleverFib						

  Note: Only the test cases are graded. The above exercise is designed so that you the understand the importance of the efficiency of algorithms; it is not graded.

- **Greatest Common Divisor:** You have to implement two functions to compute the GCD of two numbers a and b, named

  1. *int TrivialGCD(int a, int b)* : Trivial algorithm to calculate the greatest common divisor of a and b. You run a loop from $\min(a, b)$ to 1, checking if the loop iterator divides both a and b. You return the highest such value.
  2. *long int EuclidGCD(long int a, long int b)*: Implement Euclid's algorithm to calculate the greatest common divisor of a and b, and return it. You can read about the Euclidean GCD algorithm here. (Euclidean GCD).

Test Cases

You will be evaluated on several hidden test cases. These test cases will not be revealed to you. For each hidden test case, you need to match the expected output. Then only will you get the full marks. You will not see your score until the deadline is over when we grade your code using autograder, and publish the result. This assignment has 10 hidden test cases with 10 marks for each. 6 test cases on Fibonacci numbers and 4 on GCD. You may resubmit as many times as you want, but only the final submission will be graded.

Important Points

1. Your function names should be exactly as stated above. Without it the test cases will not pass. You can create any helper function you require, but **DO NOT create a main function**. Your code should only contain the 4 required functions and any other helper functions you may require.

2. **(Only for the non-graded exercise; do not include it in your code)**

   For calculating the actual running time in executing a function, you may use the clock() function in C. For this, you need to include library time.h library. The following code shows how to use it to find the time taken to execute a part of the code:

```
#include <stdio.h>
#include <time.h> // for clock_t, clock()

void func(){
    printf("Hello World");
}

int main()
{
    clock_t start_t, end_t;
    double total_t;
    start_t = clock();
    func();
    end_t = clock();
    total_t = (double)(end_t - start_t) / CLOCKS_PER_SEC;
    // CLOCKS_PER_SEC is a constant defined in time.h and its value is 10^6.
    printf("Total time taken: %f\n", total_t );
    return 0;
}
```

   clock() returns the actual time in microseconds. It might be the case that some algorithms are so fast that for small values of n, you won't be able to find the precise time taken during its execution using the above method. However, you are strongly advised to use your creative skills to deduce the precise time using the same method. The following idea might give you the right direction: how to measure thickness of a page of a book consisting of 1000 pages using an inch tape?