Lab 1 Report – Recursive Function Experiments

Title: Lab 1 – Report
Date: 01 September 2023
[redacted]

Just trying out Swift here for the first time so the code may not be pretty

1. Introduction

Objective:

For this lab report we will be taking a look at the **Extended Euclid Algorithm** and the **Hanoi Tower Algorithm with Adjacency Constraint**.

> *In addition to this I am going to extend the Euclid algorithm to include computation of A, B with rational numbers that approximate the set of irrationals and reals by continued fraction expansion

Problem Statement:

We will tackle multiple problems:

1. For the Euclid Algorithm we will be extending the algorithm so that each computation includes the identity $ax + by = gcd(a, b)$. Take note that in the case of co−prime integers where $gcd(a, b) = 1$ which allows us to find the *modular multiplicative inverse* when the a and b are co−prime.
2. We extend this notion the rational numbers instead of just integers to allow for an extended domain of computation that approximates the real domain for application to real world ratio problems. We include the $lcm(a, b)$ equivalent method as well to give us a complete set of tools for application of the gcd.
3. For the Hanoi Tower Variation we add the constraint to the original Hanoi problem (see here), that given pegs `(A,B,C)` we area not allowed to move any disks directly from `A -> C`.

2. Results

All Goals Achieved: [YES]

Extended Euclid Algorithm

> For the Euclid Algorithm we were able to satisfactorily introduce an extension which decomposes the relation of `a` and `b` as a linear combination with `x` and `y` of the `GCD(a,b)`. We can see that the extended Euclidian algorithm introduces no additional complexity to the original GCD since there are no additional branches which enter loop processing nor are there any additional recursive calls. Which leads us to the hypothesis that $O(n)_{extended} = O(n)_{originalgcd} + C$

> In addition to this we introduced our own utility functions to the GCD procedure to include *rational approximation* along with the rational calculation and LCM relation. Using rational approximation we extend GCD past the integer domain which allows us to apply GCD to continuous value problems which relate to ratio problems such as frequency division and spatial division. We show that the *continued rational expansion algorithm* is related to the Euclid algorithm.

Towers of Hanoi

> The original Towers of Hanoi problem is a straightforward recursion problem which subdivides the concerns of the movement of individual disks to the movement of the disks which lie above. The recursion splits into two subproblems of equal size for each $(n-1)$ case. Also we account for a single move between. If we could visually display the call graph as splitting in the recursion we could get a good idea of the runtime of the algorithm which is deterministic and set at $O(n) = 2^n - 1$.

> The variation of the algorithm which disallows non−adjacent moves. Add's more complexity to the algorithm and can be difficult to implement correctly. The adjacency requirement is difficult to analyze, by enforcing adjacent moves only each time we request to move a disk at a certain depth, along with its disks above, we introduce another swap subproblem for each level. We can break this problem up by saying that instead of moving all disks from `A` to `C` that all concrete specified moves require adjaceny moves only. However in this we add another recursive depth call which evaluates to 3 recursion calls per depth entry vs two.

> We were able to successfully implement the adjacency recursion problem and simplify by ensuring stack calls enforced object tracking

All Tests Passed: [YES]

Overview of results

3. Algorithm Framework

- **Algorithm Method Chosen + Rationale For Selection**

  Towers of Hanoi Adjacency Problem

  > We need implement the Towers of Hanoi algorithm and Towers of Hanoi with adjacency restriction by means of multiple recursion.
  > By relating the recursive structure of this problem to a case of the permutation of tree paths, where all possible paths are taken with a binary tree, we should get an idea for the actual logical runtime of the original problem and more general recursive call impacts in general.
  > In the case of the adjacency problem we extend this permutation to have its base order, as in the k in k^n, where k represents number of recursive paths met at any layer in the algorithm.

Extended Euclidian Algorithm

> We solve the Extended Euclidian by means of single recursion where the progressive case is determined by the termination of successive division operations.

> In the case of the extension, the parameters s, t are well defined at any step in the computation and therefore do not add any complexity.

$R_{k-2} = q_k r_{k-1} + r_k$ [1]

> This relation explains how when $r_k = 0$ then r_{k-1} is the common divisor. In this case q_k can be disregarded. This relationship can be mapped into a recursion by successive modulus division

```
func gcd(a,b){
        if a == 0{
                return b  //R_{k-1}
        }else{
                return gcd(b%a,a)
        }
}
```

By this very fact \ we can find r_{k-1} by deduction when $r_k = 0$

> For extended euclidian algorithm all we do is consider the quotient q_k and we follow the same euclidian algorithm. The only difference is that we also consider:

$s_{i+1} = s_{i-1} - q_i s_i$
$t_{i+1} = t_{i-1} - q_i t_i$

> However we can solve for s, t with:

$$t_{k+1} = \frac{a}{\gcd(a,b)}$$

$$s_{k+1} = \frac{b}{gcd(a,b)}$$

Continued Fractional Expansion

> We also extend the Euclidian Algorithm easily to handle rationals. In order to extend reals into the problem set we need a way to approximate irrationals. This is done by CFE.
> CFE is a variation of the Euclidian Algorithm as we successively generate quotient remainders that approximate the original real number, this is also a variation of the process used for "long division" which can turn a rational representation into its decimal representation.

Key steps

Extended Euclidian Algorithm

1. Select initial a, b for Euclidian solution
2. Check if b which is the remainder input for successive recursion is 0. If so a must be our common divisor.
3. Otherwise continue successive divisions and divide the remainder out by the current divisor A.

4.

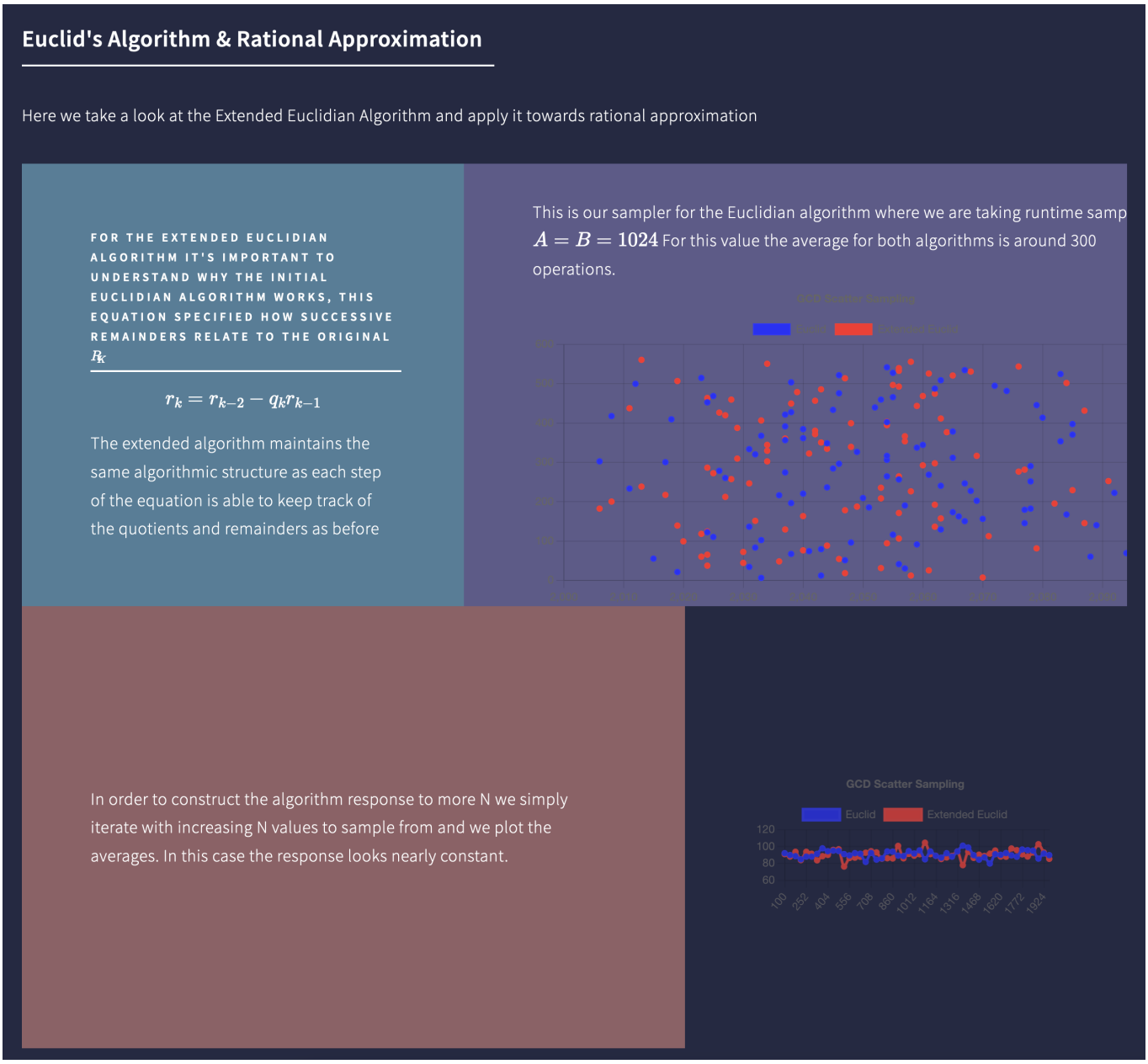
Towers of Hanoi Variation:

1. Represent Disks and Pegs with Stack Data Structure
2. Compute Hanoi on some starting configuration from a source to destination
3. Check if the Hanoi source and destination are adjacent
4. If Yes then use classical movement pattern to move disks
5. Else implement movement pattern where all base case encoded movements ensure only adjacent pegs are used (3 recursions)

4. Complexity Analysis

Extended GCD

- The initial recursion looks deceptively simple as successive modulations involves a single recursion call. However the dependency on the modulus remained introduces complexity to calculation since there's no fixed `N` and given any modulus its hard to say if we progressed closer or farther to the solution.
- In fact trying to get an approximate look at the run−time requires a statistical rather than purely logical inductive approach.
- This statistical relevance is shown on the attached **Scatter plot**
- This lack of definition could force us to say that for an average case , looking at random sets of numbers, that we would decrease the successive remainders by 1/2. This implies an average case approximately of $O(n) = log(N)$ since every case should on average over the law of large numbers decrease by at least $\frac{1}{2}$ the complexity of the original complexity N.
- In this average case however we can define this original complexity as dropping the successive quotients of both a and b. So $O(n) = \$log(a) + log(b) = log(N)$
- However for worst case we can just assume the each remainder is only reduced by 1 so $O(n) = a + b = N$

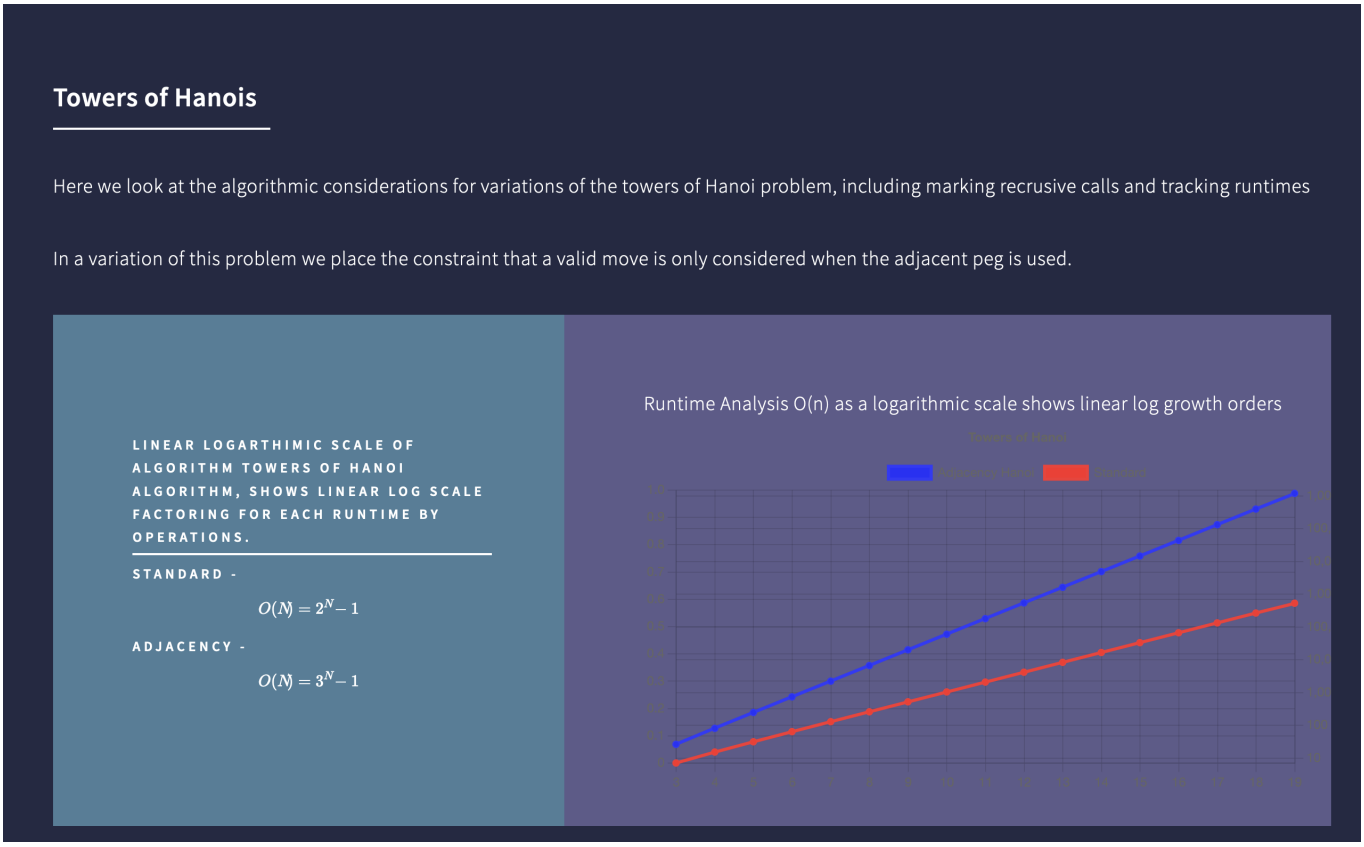


While numerical analysis on its own is no substitute for logical analysis we can at least see that in terms of A, B that the runtime is nearly constant or at least very good by sampling and reconstruction.

Towers of Hanoi

Much of what is to be said about the complexity of Towers of Hanoi adjacency I mentioned previously. However the key points are:

- We have a definite progression of $N - 1$ for each Hanoi recursion case.
- The adjacency hanoi problem includes two paths: 1) Classic Hanoi with two recursions. 2) A 3 way recursive call chain
- These call chains permutate eachother, each being called N times
- In the worst case with a $3 - way$ chain then we have $O(n) \approx 3^n$
- We can visualize this as the permutated paths through a ternary tree structure or in the case of a binary tree the original hanoi problem.



Here on a logarithmic graph in relation to the number of operations we can see the function is tailored towards exponential execution although we can't prove via this method.

5. Lessons Learned & Feedback

- **Challenges Faced**
  - Towers of Hanoi algorithm refused to work for and the recursive steps are nearly impossible to trace. There needs to be a better way to trace and visualize recursive algorithms, tree structures may be the best way.
  - GCD computing the rational approximations through continued fraction expansion was non–intuitive. I should have just taken the 10^x integer representation. Although we can get the best approximations with the continued expansion.
- **Insights Gained**
- Implementing the rational approximation algorithm through continued rational expansion. This was un–intuitive as I first was looking for a good way to approximate but now it makes more sense, we can take the integer portion of a irrational number and separate it from its decimal component. **THIS EXACTLY PARALLELS THE EUCLIDIAN ALGORITHM**
- When we start with an irrational number we obtain successive quotients in much the same way until we reach a case where the remainder of the quotient is 0. At this point we can return and sum the fractional components and take their inverse
-
- **Feedback**

Concluding Remarks

This was challenging and I learned alot about the issues dealing with quantifying recursive techniques I appreciate the notes and lectures. I was intrigued by alot of the questions that arrive out of basic algorithmic problems and their relatedness for example recursion to tree paths and the GCD in terms of fractional representation.

ChatGPT Usage: ChatGPT was used in a conversational and informational manner to confirm ideas I've intuited already about the problems posed. I did ask for pseudo code when dealing with rational GCD as I felt that this could help iterate towards a solution quicker. Notably the pseudo code was in fact usually inadequate. In particular I used chat GPT to help me clarify about dealing with GCD in the real domain and applications. I have posted the chat discussion link below.

6. References and Use of Tools

[1]– https://en.wikipedia.org/wiki/Euclidean_algorithm
[2]– ChatGPT discussion on GCD with irrational numbers for applications of rational approximations
https://chat.openai.com/c/4c56d02c-d70e-4dc4-8cba-00b758d8d0eb
[3]– https://en.wikipedia.org/wiki/Continued_fraction