CSCI 5454 – Fall 2024
Design and Analysis of Algorithms

Homework 1

Instructions:

- **Topics:** Analysis of algorithms, the Euclidean algorithm and its analysis.
- **Due date:** This homework is due by 11:59PM Mountain Time on **Friday, September 6th**. Late assignments will not be accepted. Your lowest two homework scores for the semester will be dropped.
- You are welcome and encouraged to discuss the problems with classmates, but **you must write up and submit your own solutions and code**. You must also write the names of everyone in your group on the top of your submission.
- The primary resources for this class are the lectures, lecture slides/notes, the CLRS and Erickson algorithms textbooks, the teaching staff, your collaborators, and the class Piazza forum. We strongly encourage you only to use these resources. If you do use another resource, make sure to cite it and explain why you needed it. **Using generative AI tools (e.g., ChatGPT), Q&A forums (e.g., Stack Exchange, Quora), or cheating repositories (e.g., Chegg, CourseHero) is not allowed.** See the syllabus for a more detailed explanation.
- There are three questions, worth 50 points in total.
- You must justify all of your answers unless specifically stated otherwise.
- We strongly encourage (but do not require) you to write your solutions in LaTeX, and have provided a skeleton file.[1] However, if your homework is illegible then we reserve the right not to grade it.

[1] If you have not used Latex before, it's easy to get started using Overleaf. See their 30-minute tutorial: `https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes`.

Questions

Question 1. (Continued fractions, 25 points.)

In this problem, we will derive an algorithm for continued fraction approximations. Let r be a real-number such that $0 < r \leq 1$. A continued fraction expansion of r has the form:

$$\cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots + \cfrac{1}{a_n + \ddots}}}}},$$

where $a_1, \ldots, a_n, \ldots$ is an infinite sequence of positive integers.

For a rational number $\frac{a}{b}$ where $a < b$ are positive integers, the continued fraction expansion is always finite:

$$\cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots + \frac{1}{a_n}}}}}.$$

Note that a, b and $a_1, \ldots, a_n$ are all positive integers.

For example, the continued fraction approximation of $\frac{2}{7}$ is $\frac{1}{3+\frac{1}{2}}$. Thus in this case $n = 2$, $a_1 = 3, a_2 = 2$. As another example, take the number $19/29$. Its representation as a continued fraction is $\frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{9}}}}$. Thus, $n = 4$ and $a_1 = 1, a_2 = 1, a_3 = 1, a_4 = 9$. As a final example, suppose we have a number $a = 0.707106781187 = \frac{707106781187}{10^{12}} \approx \frac{1}{\sqrt{2}}$, a continued fraction representation is of the form:

$$\cfrac{1}{1+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{1+\cfrac{1}{2+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{3+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{9+\cfrac{1}{1+\cfrac{1}{3+\cfrac{1}{17+\cfrac{1}{3+\frac{1}{9}}}}}}}}}}}}}}}}}}}}}}}}}}}}}$$

a. (5 points.) For any positive integers $a < b$, we wish to write a, b as

$$\frac{a}{b} = \frac{1}{q + \frac{r}{a}}$$

where q, r are also positive integers and $r < a$. Work out what q, r must be in terms of a, b.

b. (10 points.) Suppose we are given two positive integers a, b such that $0 < a < b$. Write the pseudocode that will output an array of the form $[a_1, \ldots, a_n]$ wherein $a_1, \ldots, a_n$ denote the numbers associated with the continued fraction. You can write a python function that takes in a, b as inputs and output a list $[a_1, \ldots, a_n]$. Write a brief 2-3 line explanation as to why your code will work. **Note:** Please be brief in your response. Overly long or complicated responses will receive negative points.

c. (5 points.) Write pseudocode that given a list $[a_1, \ldots, a_n]$ for $n \geq 1$, representing a continued fraction, returns a pair of integers (a, b) such that

$$\frac{a}{b} = \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots + \frac{1}{a_n}}}}} .$$

Assume $a_1, \ldots, a_n$ are positive integers.

d. (5 points.) A continued fraction approximation of a number $\frac{a}{b}$ is obtained by first computing its continued fraction representation $[a_1, \ldots, a_n]$. We then consider a series of fractions:

$$\frac{1}{a_1}, \frac{1}{a_1 + \frac{1}{a_2}}, \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3}}}, \cdots, \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots + \frac{1}{a_n}}}}}$$

The idea is that each fraction provides a more refined approximation to $\frac{a}{b}$. For instance, take $\frac{a}{b} = \frac{19}{29}$, whose continued fraction is given as $[1, 1, 1, 9]$. Thus, we can write a series of approximations:

$$\frac{1}{1} = 1, \frac{1}{1+1} = \frac{1}{2}, \frac{1}{1 + \frac{1}{1+2}} = \frac{2}{3}, \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{9}}}} = \frac{19}{29} .$$

Implement a small python program that will compute continued fraction approximations for given fractions $\frac{a}{b}$. Your program will take as an input a, b and output as follows.

Examples: For $a = 19, b = 29$,

```
1/1 = 1.0
1/2 = 0.5
2/3 = 0.6666666666666666
19/29 = 0.6551724137931034
```

For $a = 419, b = 1008$,

```
1/2 = 0.5
2/5 = 0.4
5/12 = 0.4166666666666667
32/77 = 0.4155844155844156
37/89 = 0.4157303370786517
69/166 = 0.41566265060240964
175/421 = 0.4156769596199525
419/1008 = 0.4156746031746032
```

Show the outputs for fractions: $\frac{11}{39}$, $\frac{113}{312}$ and $\frac{14159265359}{10^{11}}$. There is no need to include your code. Outputs should be in the format shown above.

Note: Continued fractions are useful. In many cases when we use numerical algorithms, the result comes out to be something like 0.49999998, which is close to a rational number with a small denominator. A continued fraction scheme can easily "recognize" that such a number is close to $\frac{1}{2}$.

Question 2. (Subarray sum queries, 10 points.)

We are given an array $A[0, \ldots, n-1]$ of n numbers. We would like to support "sub-array sum" queries, i.e., queries of the following form:

- **Input:** Indices l, u of A, where $0 \leq l \leq u \leq n-1$.
- **Output:** The sum $s = \sum_{j=l}^{u} A[j]$ of all entries of the sub-array $A[l, \ldots, u]$ of A (inclusive of l and u).

For this question, we will assume that arbitrary numbers fit in 1 array cell, and that arithmetic operations with arbitrary numbers can be performed in $O(1)$ time. For each part, write pseudocode for the preprocessing step that fills in the array B, and separate pseudocode that uses the information in arrays A, B to answer queries. For simplicity, you may assume that $\sqrt{n}$ is a whole number.

a. (5 points.) Suppose you are provided with an additional array $B = B[0, \ldots, \lceil\sqrt{n}\rceil - 1]$ of size $\lceil\sqrt{n}\rceil$. Describe how to preprocess A and store the resulting information in B so that it's possible to answer each sub-array sum query about A in $O(\sqrt{n})$ time (where the query algorithm is allowed to use both A and B).

b. (5 points.) Suppose that B can store n numbers. Describe a scheme that will answer each query in $O(1)$ time.

Question 3. (Subarray max queries, 15 points.) Now consider the following modification to Question 2, where instead of asking for the sum of all elements in a subarray $A[l, \ldots u]$, we would like to know the maximum. I.e., now the problem is as follows:

- **Input:** Indices l, u of A, where $0 \leq l \leq u \leq n-1$.
- **Output:** The maximum $M(l, u) = \max_{j \in \{l, \ldots, u\}} A[j]$ of all entries of the sub-array $A[l, \ldots, u]$ of A (inclusive of l and u).

a. (3 points.) Suppose that we build a table B where $B[i, j]$ stores the maximum value of all array elements $A[i], \ldots, A[i+j]$. Note that $0 \leq i \leq n-1$ and $0 \leq j \leq (n-1) - i$. Write down pseudocode that will fill up the table B in $O(n^2)$ time.

(**Hint**: First figure out how to fill $B[i, 0]$ for each i. Next, figure out how to fill out $B[i, j+1]$ given $B[i, j]$. This should remind you of memoization in dynamic programming.)

b. (7 points.) Storing the entire table B is quite wasteful. In this problem, we will instead create a table $C[i, j]$, where $C[i, j] := \max_{k \in \{i,\ldots,i+2^j-1\}} A[k]$. In other words,

$$\begin{aligned} C[i, 0] &= \max(A[i]) \ , \\ C[i, 1] &= \max(A[i], A[i+1]) \ , \\ C[i, 2] &= \max(A[i], A[i+1], A[i+2], A[i+3]) \ , \\ &\vdots \end{aligned}$$

We will not bother filling $C[i, j]$ if $i + 2^j > n$.

i. What is the size of the table C? Write your answer in the form $\Theta(f(n))$ for some f.

ii. Write a recurrence for $C[i, j+1]$ in terms of $C[i, j]$ and $C[i + 2^j, j]$.

iii. Write down pseudocode for filling up the array C.

c. (5 points.) Write pseudocode to compute $M(l, u)$ in $O(1)$ time using the table C.

(**Hint**: You should be able to express $M(l, u)$ as maximum of just two entries in C. Suppose you are given an array with $n = 53$, $l = 18$, $u = 29$, we could write $M(18, 29) = \max(C[18, 3], C[22, 3])$. Think about other cases and work out a general scheme.)