CS 381 Introduction to the Analysis of Algorithms
Fall 2021 Simina Branzei and Alexandros Psomas HW 3

Due Fri October 15 at 11:59PM

1. (7 points)

   (a) Prove that $\sum_{i=1}^{n} i^4 \leq n^5$ for all $n \geq 1$.

   (b) Prove by induction that $\frac{n^5}{5} \leq \sum_{i=1}^{n} i^4$ for all $n \geq 1$.

   (c) Conclude that $\sum_{i=1}^{n} i^4$ is $\Theta(n^5)$

2. (8 points)

   Consider the following recurrence:

   - $T(1) = T(2) = T(3) = 1000$
   - For $n > 3$, $T(n) = T(\lceil \frac{2n}{3} \rceil) + T(\lceil \frac{3n}{4} \rceil) - 2n - 1$

   (a) Find a constant c such that $T(n) \geq cn^2$ for $n \geq 1$. (Hint: if you're not sure what constant to pick, try starting part b first, and come back to this part)

   (b) Prove by induction that your chosen c is correct (i.e. that $T(n) \geq cn^2$ for $n \geq 1$).

   (c) Conclude that $T(n)$ is $\Omega(n^2)$.

3. (5 + 5 = 10 points) Consider the following pseudo code for finding the length of the longest common sub sequence (LCS) of two strings X and Y.

```
LCS-LENGTH(X,Y){
    m = X.length
    n = Y.length
    for i = 1 to m{
        c[i][0] = 0;
    }
    for j = 1 to n{
        c[0][j] = 0;
    }
    for i = 1 to m{
        for j= 1 to n{
            if x[i] == y[j]{
                c[i][j] = c[i-1][j-1] + 1
            }
            else{
                c[i][j] = max(c[i-1][j],c[i][j-1]);
            }
        }
    }
    return c;
}
```

- Given an input $X = AGGTAB$ and $Y = GXTXAYB$, show the contents of the 2-Dimensional array c upon calling LCS-LENGTH(X,Y). Note that you only need to show the final contents i.e., after the pseudo code stops running.
- Given an input $X = ABCDE$ and $Y = BCE$, show the contents of the 2-Dimensional array c upon calling LCS-LENGTH(X,Y).

4. (12 + 3 = 15 points) Gotham city is in trouble again! The Riddler has placed bombs in every major part of the city and they will explode tonight unless Batman stops him. Now, the Riddler wants to challenge Batman to a battle of wits and thus he presents a way for the Batman to defuse the bombs. To do so, all Batman has to do is solve an algorithmic problem.

   Given two strings X and Y, Batman has to find the length of the longest common substring among X and Y. Batman had written a naive solution earlier which had worked for small inputs. But, the Riddler drastically increased the input size such that no algorithm that runs slower than $O(m \cdot n)$ - where m, n are the lengths of strings X, Y respectively - can find the longest common substring before the bomb explodes. Batman stands thoroughly beaten by this challenge. Can you help him to obtain an $O(m \cdot n)$ algorithm to find the length of the longest common substring?

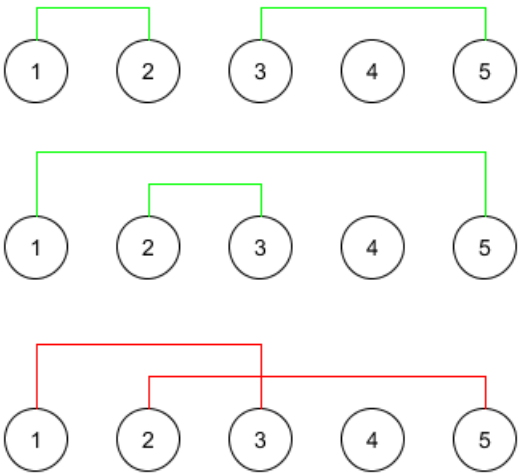
   Note: A substring is a contiguous sequence of characters in a string. For instance, "Computer Science" is a substring of "Computer Science is fun", while "Computer fun" is not. You can assume that $m, n \geq 1$.

   Example: Given X = ABABC and Y = BABCA, the longest common substring is "BABC" and its length is 4.

   - Find, describe, and prove the correctness of an algorithm to find the length of the longest common substring. It must run in $O(m \cdot n)$ time or better; faster is possible but not at all required. (12 points)
   - Analyze the algorithm and show it runs in $O(m \cdot n)$. (3 points)

   Hint: try using $OPT(i, j) =$ the length of the longest common substring of the first i characters of X and the first j characters of Y.

5. (30 points) We say that two pairs of distinct natural numbers (a,b) and (c,d) with $a < b$ and $c < d$ are "entangled" if either $a < c < b < d$ or $c < a < d < b$. For example, the pairs $(1,2)$ and $(3,5)$ are not entangled, nor are $(1,5)$ and $(2,3)$, but $(1,3)$ and $(2,5)$ are entangled.



The order in a pair doesn't matter, so $(3,1)$ and $(2,5)$ are still entangled. You are given an $n \times n$ matrix V of non-negative "pair values"; i.e. $V_{i,j}$ is the value of the pair (i,j). You cannot pair a number with itself, so $V_{i,i} = 0$ for all $1 \leq i \leq n$. The pair (i,j) is the same as the pair (j,i), so $V_{i,j} = V_{j,i}$ for all $1 \leq i < j \leq n$. Your goal is to determine the maximum value possible from forming pairs from the numbers $1,2,\ldots,n$ such that no two pairs are entangled. Each number may be paired at most once. It is not necessary to pair every number.

For example, suppose we had $n = 5$ with

$$V = \begin{bmatrix} 0 & 2 & 3 & 5 & 1 \\ 2 & 0 & 4 & 12 & 3 \\ 3 & 4 & 0 & 8 & 2 \\ 5 & 12 & 8 & 0 & 6 \\ 1 & 3 & 2 & 6 & 0 \end{bmatrix} \tag{1}$$

$(1,2),(3,4)$ is a valid pairing, and has value $2+8=10$. $(1,5),(2,3)$ is a valid pairing, and has value $1+4=5$. $(2,4)$ is a valid pairing, and has value 12. It is also the optimal pairing. $(1,3),(2,4)$ is not a valid pairing, because $(1,3)$ and $(2,4)$ are entangled. $(1,2),(2,4)$ is not a valid pairing, because 2 appears in multiple pairs.

Find an algorithm with fastest asymptotic time complexity for this problem. Then describe it, prove its correctness, and prove its asymptotic time complexity. You do not need to return the optimal pairing, only its value. You do not need to prove that your time complexity is the fastest possible. Solutions which require exponential time are too slow, and will not receive credit. You do not need to prove anything regarding space complexity.

6. (10 + 10 + 5 points) Consider the problem of giving change "optimally" in American denominations. You are given a value V and are asked how much of each American coin and bill do you give back to have the total add up to V and such that you return the fewest bills and coins possible. Assume for this problem American Denominations are $\{\$20.00, \$10.00, \$5.00, \$1.00, \$0.10, \$0.05, \$0.01\}$ (note: there are No Quarters). If no combination exists for a given value (such as $V = \$10.005$) then the algorithm should return that there is no combination possible for this value.

   For example, if the input to the algorithm was V = \$31.27 the "optimal" answer would be: one 20, one 10, one 1, one quarter and two pennies.

   - Design an efficient polynomial time greedy algorithm that provides the "optimal" number of each denomination for an input value V.
   - Prove why a greedy approach will always find a correct and optimal solution. (Hint: use the fact that every denomination in question is a multiple of the next smallest).
   - Analyze and prove the algorithm's runtime (You may assume that $V < 2^n$ is the only input passed to the algorithm and that modulus operation can be computed in constant time).

7. (5 points) Consider the problem of giving change "optimally" in n arbitrary denominations. You are given a value $V < 2^n$ and a set of denominations D_n and are asked how much of each denomination do you give back to have the total add up to the value and such that you return the fewest denominations possible, or return that there is no combination of the denominations that add up to V

   For example, if the input to the algorithm was $V = 3.32$ and $D_n = \{1.27, 0.67, 0.13\}$ the "optimal" answer would be: two 1.27s and six 0.13s.

   Either design/modify an efficient polynomial time greedy algorithm to solve this problem and prove it's correctness and runtime, or justify why no polynomial time greedy algorithm can correctly solve this problem for all cases. (You may assume that $V < 2^n$ is the only input passed to the algorithm and that modulus operation can be computed in constant time).