Cosc 30: Discrete Mathematics

Prishita Dharampal, Arses Prasai

Problem 1. **Asymptotic analysis.** We often use big-O notation to describe the asymptotic behavior of algorithms. Formally, the proposition $f(n) = O(g(n))$ means:

$$\exists C > 0 \, \exists N \, \forall n \geq N, \quad f(n) \leq C \cdot g(n).$$

Prove the following statements (as formal quantified propositions; hand-waving arguments based on intuition of big-O notation are insufficient):

(a)

$$\sum_{k=1}^{n} k^{30} = O(n^{31}).$$

(b)

$$\sum_{k=1}^{n} \frac{1}{k} \neq O(1),$$

formally interpreted as

$$\neg \left(\sum_{k=1}^{n} \frac{1}{k} = O(1) \right).$$

Solution.

(a) To show that

$$\sum_{k=1}^{n} k^{30} = O(n^{31}).$$

we need to identify constants $C > 0$ and N such that $\forall n \geq N, \sum_{k=1}^{n} k^{30} \leq C \cdot n^{31}$.

For all $1 \leq k \leq n$, we know that, $k^{30} \leq n^{30}$ (because $k, n > 0$).

$$\implies \sum_{k=1}^{n} k^{30} \leq \sum_{k=1}^{n} n^{30} = n \cdot n^{30} = n^{31}$$

Hence we get,

$$\sum_{k=1}^{n} k^{30} \leq n^{31}$$

So, for $C = 1, N = 1$ we know that $\sum_{k=1}^{n} k^{30}$ grows at worst as fast as n^{31} and hence, $\sum_{k=1}^{n} k^{30} = O(n^{31})$.

(b) To show that

$$\sum_{k=1}^{n} \frac{1}{k} \neq O(1),$$

we need to show that for all possible values of constants $C > 0$ and N there exists an $n \geq N$ for which

$$\sum_{k=1}^{n} \frac{1}{k} > C \cdot O(1) = C$$

So, essentially we need to show that the summation $\sum_{k=1}^{n} \frac{1}{k}$ is unbounded. Let this series be $A(n)$. We can prove that $A(n)$ is unbounded if we can find a different series $B(n) \leq A(n)$ that is unbounded.

Define $B(n)$ to be the series such that for every term $\frac{1}{k} \in A(n)$, we replace k with the least power of 2 less than or equal to it. I.e.

$$\frac{1}{1} \to \frac{1}{2^0}, \qquad \frac{1}{2} \to \frac{1}{2^1}, \qquad \frac{1}{3} \to \frac{1}{2^2}, \qquad \frac{1}{4} \to \frac{1}{2^2}, \qquad \frac{1}{5} \to \frac{1}{2^3}, \qquad \cdots$$

We can represent this formally as,

$$B(n) = \sum_{i=1}^{n} \left(\sum_{j=2^{i-1}+1}^{2^i} \frac{1}{2^i} \right) = \sum_{i=1}^{n} \frac{1}{2^i} \left(\sum_{j=2^{i-1}+1}^{2^i} 1 \right) = \sum_{i=1}^{n} \frac{1}{2^i} 2^{i-1} = \sum_{i=1}^{n} \frac{1}{2}$$

As n tends to infinity, the summation also tends to infinity, because there $\frac{1}{2}$ would be added an infinite number of times. Hence, $B(n)$ is unbounded as n tends to infinity. Since $\forall i \in \{1, \cdots, n\}, b_i \in B(n), a_i \in A(n)$, $b_i \leq a_i$, we can see that $B(n) \leq A(n)$. Thus, as $B(n), n \to \infty$ is unbounded $A(n), n \to \infty$ must also be.

Therefore, there exists no $C > 0$ such that

$$\sum_{k=1}^{n} \frac{1}{k} \leq C$$

Hence, there $\exists n \geq N$, such that $\sum_{k=1}^{n} \frac{1}{k}$ must be greater than C, for all $C > 0$.

Problem 2. No all-purpose lossless compression. The purpose of this problem is to demonstrate that, even with the materials learned so far, we can already prove a non-obvious fact in computer science.

(a) Write the following statement as a formal quantified proposition:

Every lossless compression algorithm that shortens some input text must lengthen some other input text.

[Hint: It is not sufficient to treat "lossless compression algorithm" as a basic type, for the purpose of the next subquestion.]

(b) Prove the above statement about lossless compression algorithms.

[Hint: The problem may look daunting at first because we have not seen many CS facts proved formally before. Trust the process and carry out the mechanical part of the proof; an insight should strike. Read the first hint if necessary.]

Solution.

(a) By definition, lossless compression algorithms map an input text of finite length to a representation such that it can be directly mapped back to the original input text. Hence, a lossless compression algorithm is an injective function.

Let f be a function such that, $\forall$ input text t: $f(t)$ gives us the unique resulting representation r. We can write this as: $f(t) = r$.

Then, we can state the following proposition as follows:

$\forall$ inj. functions $f, (\exists$ input text $t_i : |f(t_i)| < |t_i|) \implies \exists$ input text $t_j : (|f(t_j)| > |t_j|)$

(b) Let A be an alphabet system with k different alphabets, and let A_m be the set of all strings in the alphabet of length at most m. Let function $f : A_m \to A_m$ be the lossless compression algorithm (as defined in part (a)). So, f is injective. Then for an arbitrary string a of length m, in A_m, we know that ideally the lossless compression function would map it to a string of length strictly less than m.

Without loss of generalization, assume that $f(a) = b$, where b is a string of length n, and $n < m$.

Now, consider the function $f' : A_n \to A_n$, where A_n is the set of all strings in the alphabet of length at most n. Now, because the function is injective and we already mapped $a \to b$, the codomain A_n has at least 1 available less string. Now, because $|domain A_n| > |range A_n|$, and f' is injective, at least one string from the input cannot be mapped. So, that input would necessarily have to be mapped to a string of length strictly greater than n. This proves that every lossless compression algorithm that shortens some input text must lengthen some other input text.

Problem 3. **Forward Tower of Hanoi.** Consider the following variant of the Tower of Hanoi: the pegs are labelled 0, 1, and 2, and the goal is still to move a stack of n disks (sorted from smallest to largest) from peg 0 to peg 1, such that at any point no disk can sit on top of another disk of smaller size. However, there is one extra constraint: every disk at peg i can only be moved to the next peg $i+1 \pmod 3$. In other words, if we imagine the pegs are placed in cyclic order, then every disk can only move forward one step at a time.

1. Describe a process to solve this variant of the Tower of Hanoi puzzle. Your algorithm may be described in English or in pseudocode, but explanations are required.

2. Prove that your process correctly solves the puzzle in at most $4^n - 1$ steps.

   [Hint: If you insist on the optimal bound, you need some combinatorics. Read Pigeon 6.4.3 for inspiration.]

Solution.

1. To move n disks from peg $i-1$ to peg i, we

   (a) recursively move $n-1$ disks from $i-1$ to peg i

   (b) recursively move $n-1$ disks from i to peg $i+1$

   (c) move disk n to peg i

   (d) recursively move $n-1$ disks from peg $i+1$ to peg $i+2 \equiv i-1 \pmod 3$

   (e) recursively move $n-1$ disks from peg $i-1$ to peg i

   (f) when we have 1 disk left, we can move it to i without obstruction in at most 2 moves.

```
ForwardHanoi(n, i-1, i, i+1)
    n > 1:
        ForwardHanoi(n-1, i-1, i)
        ForwardHanoi(n-1, i-1, i+1)
        move disc n from i-1 to i
        ForwardHanoi (n-1, i+1, i-1)
        ForwardHanoi(n-1, i-1, i)

    n = 1:
        move disk from i-1 to i
```

2. Let f be a function such that $f(k)$ is the number of steps it takes to solve the hanoi tower with k disks. From the algorithm we can see that,

$$f(1) = 1, \qquad f(n) = 4(f(n-1)) + 1$$

We will prove that our algorithm solves the puzzle in at most $4^n - 1$ moves using induction.

Assume that $f(n-1) \leq 4^{n-1} - 1$, for all $n > 1$. Then for the n-th case, we get

$$f(n) = 4(f(n-1)) + 1$$

Since $f(n-1) \leq 4^{n-1} - 1$, we can re-write the above equation with an inequality:

$$f(n) \leq 4(4^{n-1} - 1) + 1 = 4^n - 3$$

And,

$$f(n) \leq 4^n - 3 \leq 4^n - 1.$$

Now for $f(1)$ we directly check,

$$f(1) = 1 \leq 4^1 - 1 = 3$$

Hence, the assumption holds, and the algorithm takes at most $4^n - 1$ steps to solve the hanoi tower.

Problem 4*. **Approximating square root.** On computers we do not usually represent square roots exactly (as solutions to quadratic equations), but as estimations with variable accuracy.
The following procedure HERON(x) computes estimations of the square root of x iteratively:

```
HERON(x):
  let $y_0$ be an arbitrary real number; set $i \leftarrow 0$
  while $y_i^2$ is too far away from $x$:
      $y_{i+1} \leftarrow (y_i + x/y_i)/2$
    $i \leftarrow i + 1$
  return yi
```

Heron's method is known at least by the first-century Egyptians and was named after the Greek mathematician Heron of Alexandria. The method converges quadratically (the number of correct digits doubles at each step) and is very efficient in practice. The goal of this exercise is to prove the correctness of Heron's method.

(a) State the convergence criteria of HERON(x) as formal quantified propositions.

[Hint: How do you know the procedure will eventually stop? There are at least two things to check.]

(b) Prove the convergence criteria you gave in the previous subquestion.

Solution.

1. **Proposition**: Let x be the arbitrary (positive) real number that we want to find the square root of, ε be the given margin of error, and y_j be the j-th iteration of the algorithm.

$$\forall \varepsilon > 0, \exists j \in \mathbb{N} \text{ such that } |x - y_j^2| < \varepsilon$$

2. To prove the above convergence criteria, we check what happens at an arbitrary step i in the loop. There are the following cases:

   Case 1: $y_i^2 > x$, and $y_i^2 - x > \varepsilon$

   If $y_i^2 > x$ then $x/y_i < y_i$, and $y_{i+1} = (y_i + x/y_i)/2 < y_i$. And

$$y_{i+1} < y_i \implies y_{i+1}^2 < y_i^2 \implies \varepsilon - (y_{i+1}^2 - x) < \varepsilon - (y_i^2 - x)$$

   That is to say that y_{i+1}^2 moves closer to x.

Case 2: $y_i^2 < x$ and $x - y_i^2 > \varepsilon$

If $y_i^2 < x$ then $x/y_i > y_i$, and $y_{i+1} = (y_i + x/y_i)/2 > y_i$. And

$$y_{i+1} > y_i \implies y_{i+1}^2 > y_i^2 \implies \varepsilon - (x - y_i^2) > \varepsilon - (x - y_{i+1}^2)$$

That is to say that y_{i+1}^2 moves closer to x^2.

Case 3: $|x - y_i^2| < \varepsilon$

That is the intended approximination and the program returns y_i.

Since, we have a while loop the loop does not terminate until we reach Case 3, and Cases 1 and 2 y_{i+1} moves closer to the intended value the convergence criteria is eventually satisfied.