Problem Set 3

Question 1

(a)

(b) Let T(n) be the minimum possible number of moves required to solve the game when there are n disks. For example, T(2) = 3 and T(3) = 7.

Clearly explain why T(4) = 15, showing it is possible to solve this game in exactly 15 moves and proving why it is impossible to solve this game in 14 (or fewer) moves.

(c) Find a recurrence relation for T(n) and clearly and carefully explain why that recurrence relation holds. Then solve the recurrence relation using any method of your choice to determine a formula for T(n) that is true for all integers n ≥ 1.

NOTE :

I will provide the solution of both C (finding the recurrence relation) and B since both C and B are interdependent on each other. You can prove B using the solution for C.

Let's quantify what the Tower of Hanoi problem states before we move to this problem:

Goal:

Move all disks from a *Start* index peg to *End* index peg.

Rules:

1. You can only move one disk at a time.
2. A larger disk cannot be placed on top of a smaller disk.
3. You can use an intermediate peg (in addition to the source and destination pegs) to achieve the goal.

According to the problem $T(N)$ is the minimum number of moves required to solve Tower of Hanoi for N disks.

To prove T(4) = 15, we can prove T(N) for all positive N and then substitute the value of N as 4.

However, to carry on with this proof, we need to devise an algorithm. Hence, let's look at the cases with disks = 1, 2 and 3 first as our base cases.

T(N) where N =1, can be solved by $1 \rightarrow 3$.

T(N) where N = 2 can be solved by $1 \rightarrow 2, 1 \rightarrow 3, 2 \rightarrow 3$. This notation represents the order of moving the topmost disk on the stack from the first numbered peg to the second numbered peg after the arrow $\rightarrow$.

For T(N), N = 3 can be solved: $1 \rightarrow 3, 1 \rightarrow 2, 3 \rightarrow 2, 1 \rightarrow 3, 2 \rightarrow 1, 2 \rightarrow 3, 1 \rightarrow 3$. Here we get the total number of steps as 7.

We can observe a pattern here. The total number seems to be equal to $2^N - 1$, and the problem can be solved in this manner :

1. Move $N-1$ disks from start to the extra peg, using the destination peg as a temporary peg for the extra peg. In this case, peg 2 is the extra peg and peg 3 is the end or destination peg ($T(N-1)$ moves).
2. Move the largest disk from source to destination (+1 move).
3. Now move $N-1$ disks from extra peg to destination peg, using the start peg as a temporary peg ($T(N-1)$ moves).

Total number of moves = $2T(N-1)+1$.

Using the recurrence relation and starting from T(1), we can calculate T(2), T(3), T(4), and so on:

- $T(1) = 1$
- $T(2) = 2*T(1)+1 = 2*1+1 = 3$
- $T(3) = 2*T(2)+1 = 2*3+1 = 7$
- $T(4) = 2*T(3)+1 = 2*7+1 = 15$

In general, $T(N) = 2T(N-1)+1$ for any positive integer k.

Let's try to build an argument using substitution.

$$T(N) = 2T(N-1)+1$$

$$T(N-1) = 2T(N-2)+1$$

$$T(N-2) = 2T(N-3)+1$$

$$T(N) = 2(2(2T(N-3)+1)+1)+1 \Rightarrow 2^3T(n-3)+2*2+2+1$$

at the k^{th} value-

$$T(N) = 2^kT(N-k)+2^{k-1}+..2^3+2^2+2^1+2^0$$

to calculate the value of k, we can look at our base cases :

$$T(1) = 1 \rightarrow N-k = 1 \Rightarrow k = N-1$$

substituting this value into the equation.

$$T(N) = 2^{N-1}T(N-(N-1))+2^{N-1-1}+..2^3+2^2+2^1+2^0$$

$$T(N) = 2^{N-1}T(1)+2^{N-2}+..2^3+2^2+2^1+2^0$$

$T(N) = 2^{N-1} + 2^{N-2} + ..2^3 + 2^2 + 2^1 + 2^0$

$T(N) = 2^N - 1$

Hence from the proof, we can conclude for sure that when N = 4, T(N) = 15 and no less since this is proved for the minimum number of moves.

(d) Substitute n = log(m) into your recurrence relation for $T(n)$ above, and use the Master Theorem to prove that $T(n) = \Theta(2^n)$. Briefly explain how and why your formula in part (c) is indeed $\Theta(2^n)$.

Let's substitute $n = log(m)$ in our recurrence relation $T(n) = 2T(n-1) + 1$.

$\therefore$ The relation becomes : $T(log(m)) = 2T(log(m) - 1) + 1$

Since $log_2(2) = 1$

$$T(log(m)) = 2T(log(m) - log(2)) + 1$$
$$\Rightarrow 2T(log(m/2)) + 1$$
$$\text{we can substitute T(log(m)) to be a function f(m)}$$
$$f(m) = 2(f(m/2)) + 1$$

Applying masters theorem on the last equation we have a = 2, b = 1.

According to Case 1 of master theorem, $f(m) = O(m^{log(2)}) = O(m)$

Now, we assumed $n = log(m)$ initially. Hence $m = n^2$

$\Rightarrow T(n) = \Theta(n^2)$

We can further prove that $T(n) = \Theta(n^2)$ using the argument in part (c), where I use iterative substitution to arrive at the answer.

Credits to **Hakshay Sundar** for pointing me in the right direction with this question when I was stuck on using a transformation function incorrectly.

Question 2

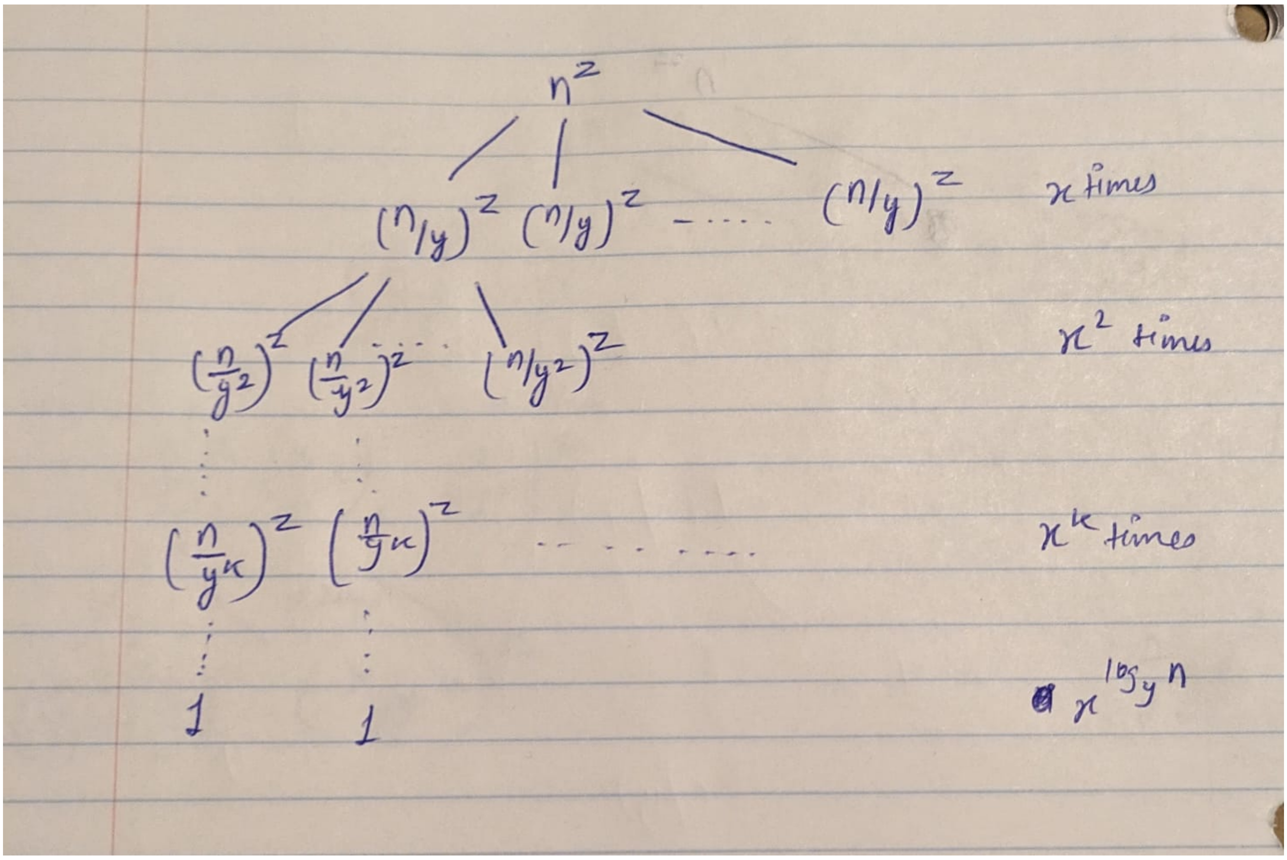
Let x, y, z be real numbers for which $T(1) = 1$ and $T(n) = xT(n/y) + n^z$.

(a) If $z < logy(x)$, prove that $T(n) = \Theta(n^{log_y(x)})$.

(b) If $z = log_y(x)$, prove that $T(n) = \Theta(n^{log_y(x)} log_y n)$.

If $z > log_y(x)$, prove that $T(n) = \Theta(n^z)$.

Let's try reducing $xT(n/y)$ through a recursion tree.



For the first level, there are x number of nodes and the sum of the nodes is simply the work done, which is $x(n/y)^z$.

For the second level, there are x nodes of each of the previous x nodes on the first level, hence the total is $x^2(n/y^2)^z$. Similarly at the k^{th} level, we have done work $x^k(n/y^k)^z$.

The grand sum will be $x(n/y)^z + x^2(n/y^2)z + \ldots + x^k(n/y^k)^z$. Since the tree branches out exponentially, we know that the total number of levels will be $k = log_y n$ where all the elements are split to 1.

This sum can be represented as : $\sum_{k=0}^{log_y n} n^z(x/y^z)^k$.

(a) Here, $z > log_y x$, we get through geometric progression,

$$n^z(a/b^z)^{log_y n}$$

This can be simplified to

$$n^z(a^{log_y n}/b^{z\log_y n})$$
$$\Rightarrow n^z(n^{log_y a}/n^z)$$
$$\Rightarrow n^{log_y a}$$

using logarithmic identities.

Hence $T(n) = \Theta(n^{log_y a})$ for (a)

(b) Here, $z = log_y x$, hence our summation becomes:

$$\sum_{k=0}^{log_y n} n^z(x/y^z)^k \Rightarrow \sum_{k=0}^{log_y n} n^{log_y x}(x/y^{log_y x})^k$$

$$= \sum_{k=0}^{log_y n} n^{log_y x}(1)^k = (1 + log_y n)n^{log_y x} = O(log_y n)n^{log_y x}$$

Hence Proved that $T(n) = \Theta(log_y n)n^{log_y x}$

(c) Here, $z < log_y x$, hence through geometric progression our summation assumes the form:

$$\sum_{k=0}^{log_y n} n^z(x/y^z)^k = (n^z)$$

Hence $T(n) = \Theta(n^z)$

(d) $T(n) = 7T(n/2) + n^2$

Strassen's algorithm takes on the form : $T(n) = 7T(n/2) + n^2$

Here, $x = 7$ and $y = 2$.

Now, $log_2(7) = 2.807..$

Hence $z = 2 < log_y x = log_2(7)$

Hence case (c) applies, so the algorithm is of $T(n) = \Theta(n^2)$ and runs in $O(n^2)$ time.