**ECE374 Assignment 4**

Due 03/06/2023

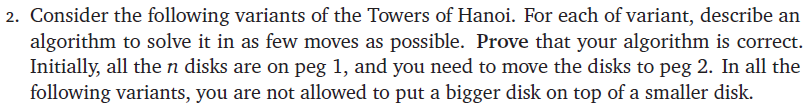
**Group & netid**

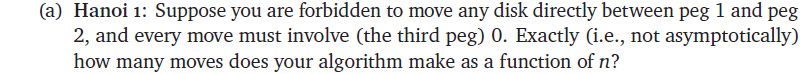
*Chen Si chensi3*

*Jie Wang jiew5*

*Shitian Yang sy39*

**Problem 2**



(a) Solution:

Assume the basic MOVE function is defined as *MOVE (from, to)*, which moves the top one disk from peg “*from*” to peg “*to*”.

HANOI\_1 (n):

hanoi\_1\_helper (n, 1, 0, 2)

hanoi\_1\_helper (n, from, via, to):

if (n=1):

MOVE (from, via)

MOVE (via, to)

else if (n>1):

hanoi\_1\_helper (n-1, from, via, to) # (ABC, \_, \_)🡪(C, \_, AB)

MOVE (from, via) # (C, \_, AB)🡪(\_, C, AB)

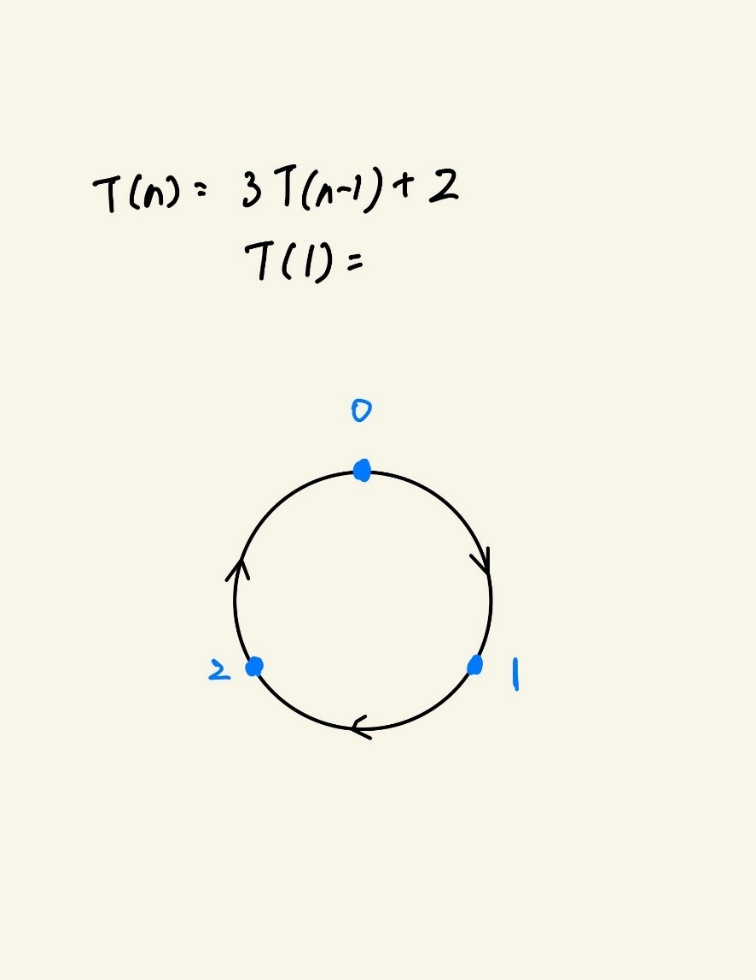
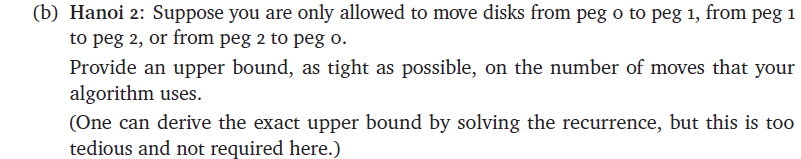
hanoi\_1\_helper (n-1, to, via, from) # (\_, C, AB)🡪(AB, C, \_)

MOVE (via, to) # (AB, C, \_)🡪(AB, \_, C)

hanoi\_1\_helper (n-1, from, via, to) # (AB, \_, C)🡪(\_, \_, ABC)

Therefore, we have the running time as T(n) = 3\*T(n-1) + 2, with T (1) = 2

So, the steps of dealing with n disks is .

(b) Solution:

Intuition:

Consider the setup of these three pegs as left image.

Let Q(n) be the min number of steps to move n disks to the next peg, T(n) be the min number of steps to move n disks to the second next peg.

We have:

(1) Q(n)

To move n pegs to the next peg (e.g. 0🡪1), we need to:

(a) Move n-1 disks from 0 to 2 🡪 T(n-1);

(b) Move the largest disk from 0 to 1 🡪 1;

(c) Move n-1 disks from 2 to 1 🡪 T(n-1)

Therefore,

(2) T(n)

To move n pegs to the second next peg (e.g. 0🡪2), we need to:

(a) Move n-1 disks from 0 to 2 🡪 T(n-1);

(b) Move the largest disk from 0 to 1 🡪 1;

(c) Move n-1 disks from 2 to 0 🡪 Q(n-1);

(d) Move the largest disk from 1 to 2 🡪 1;

(e) Move n-1 disks from 0 to 2 🡪 T(n-1);

Therefore,

Thus, the algorithm is:

T (n, from, via, to):

if (n=1):

MOVE (from, via)

MOVE (via, to)

else:

T (n-1, from, via, to)

MOVE (from, via)

Q (n-1, to, via, from)

MOVE (via, to)

T (n-1, from, via, to)

HANOI\_3 (n):

T (n, 1, 0, 2)

Q (n, from, via, to):

if (n=1):

MOVE (from, to)

else:

T (n-1, from, to, via)

MOVE (from, to)

T (n-1, via, from, to)

The run-time could be obtained by solving the equation:

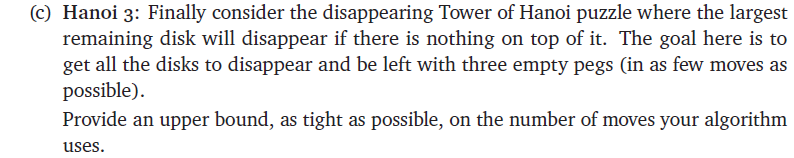
Solving it, and we get:

,

Therefore, .

As is always between -1 and 1, we could treat it as a constant

Thus, .

(c) Solution:

Intuition: It is easy to point out that, the traditional Hanoi can solve the problem, with a time complexity of , **which remove one largest disk each time.**

However, it is not the most efficient way to solve the problem.

Since there are 3 pegs, we can at most remove two largest disks every time, e.g., the 1st largest one is removed at peg1, the 2nd largest one is removed at peg2, all others are left at peg3.

With this intuition, we can design the algorithm as follows:

HANOI\_3 (n):

from = 1

via = 0

to = 2

while (n > 2):

hanoi\_3\_helper (n-2, from, via, to) # (ABCDEF, \_, \_)🡪(EF, \_, ABCD)

MOVE (from, via) # (EF, \_, ABCD)🡪(F, E, ABCD)

DISAPPEAR (from) # (F, E, ABCD)🡪 (\_, E, ABCD)

DISAPPEAR (via) # (\_, E, ABCD)🡪 (\_, \_, ABCD)

n = n – 2

SWAP (from, via)

if (n = 2):

MOVE (from, via) # (AB, \_, \_)🡪 (A, B, \_)

DISAPPEAR (from)

DISAPPEAR (via)

# If n=1, it would directly disappear

if (n=1):

DISAPPEAR (from)

# Traditional Hanoi Tower Movement

Hanoi (n, from, via, to):

if (n=1):

MOVE (from, to)

else if (n>1):

Hanoi (n-1, from, to, via) # (ABC, \_, \_)🡪(C, AB, \_)

MOVE (from, to) # (C, AB, \_)🡪(\_, AB, C)

Hanoi (n-1, via, from, to) # (\_, AB, C)🡪(\_, \_, ABC)

The time complexity of this algorithm is:

,

Where **Hanoi(n-2)** is the complexity of traditional Hanoi Tower

Referring to Saad’s paper on the fifth variation of the Tower of Hanoi, we can get:

文本, 信件

描述已自动生成

***Exploding Questions***

Therefore, the time complexity of this algorithm has an upper bound of .

Reference:

Saad Mneimneh (2018), *Simple Variations on the Tower of Hanoi to Guide the Study of Recurrences and Proofs by Induction*