EXERCISE REPORT

TEAM 15

MATHEMATICAL ANALYSIS OF RECURSIVE ALGORITHM

Lecturer: Son Nguyen Thanh MCS

Class: CS112.N21.KHTN

Member 1: Quan Vo Minh – 21520093

Member 2: Nhat Nguyen Viet – 21520378

Excercise: Team 2's Excercise

Date: 02/04/2023

1. Tower of Hanoi

a. In the original version of the Tower of Hanoi puzzle, as it was published in the 1890s by Edouard Lucas, French mathematician, the world will end after 64 disks have been moved from a mystical Tower of Brahma. Estimate the number of years it will take if monks could move one disk per minute. (Assume that monks do not eat, sleep, or die.)

The number of moves required to transfer all 64 disks is $2^{64} - 1$, as each move can only involve moving one disk and no disk can be placed on top of a smaller disk.

Assuming the monks can move one disk per minute (no eating, sleeping or dying), we can calculate the time it would take them as follows:

$$Number\ of\ moves = 2^{64} - 1$$
$$Time\ per\ move = 1\ minute$$

$$Total\ time = (2^{64} - 1) * 1\ minute = 18{,}446{,}744{,}073{,}709{,}551{,}615\ minutes$$

Assume there are 60 minutes in an hour, 24 hours in a day, and approximately 365.25 days in a year. The number of minutes in a year is:

$$60 \times 24 \times 365.25 = 525{,}600\ (\frac{minutes}{year})$$

Therefore, the years required to complete is:

$$18{,}446{,}744{,}073{,}709{,}551{,}615\ minutes\ /\ 525{,}600\ minutes/year$$
$$= 35{,}184{,}372{,}088\ years$$

b. How many moves are made by the i-th largest disk (1 <= i <= n) in this algorithm?

Algorithm step-by-step:

Assuming all n disks are distributed in valid arrangements among the pegs; assuming there are m top disks on a source peg, and all the rest of the disks are larger than m, so they can be safely ignored; to move m disks from a source peg to a target peg using a spare peg, without violating the rules:

1. Move m − 1 disks from the source to the spare peg, by the same general solving procedure. Rules are not violated, by assumption. This leaves the disk m as a top disk on the source peg.
2. Move the disk m from the source to the target peg, which is guaranteed to be a valid move, by the assumptions − a simple step.
3. Move the m − 1 disks that we have just placed on the spare, from the spare to the target peg by the same general solving procedure, so they are placed on top of the disk m without violating the rules.
4. The base case is to move 0 disks (in steps 1 and 3), that is, do nothing - which obviously doesn't violate the rules.

Recurrence relation: $T(n) = T(n-1) + 1 + T(n-1)$ with $T(0) = 0$

We see that the largest disk moves only once, the second largest disk moves twice as much and so on. Accordingly, number of moves made by largest disk to smallest disk is $1, 2, 4, \dots, 2^{n-1}$

c. Find a non-recursive algorithm for the Tower of Hanoi puzzle and implement it in the language of your choice.

Algorithm step-by-step:

1. Calculate the total number of moves required as $2^n - 1$, where n is the number of disks.
2. If the number of disks is even, swap the destination and auxiliary poles.
3. For each move i from 1 to the total number of moves:
   - If $i\%3 == 1$, move the top disk from the source pole to the destination pole.
   - If $i\%3 == 2$, move the top disk from the source pole to the auxiliary pole.
   - If $i\%3 == 0$, move the top disk from the auxiliary pole to the destination pole.

Source code: [nv259/CS112.N21.KHTN/subject02/tower_of_Hanoi.py]

2. Quicksort

T(n): time complexity of Quicksort for input size of n.

At each step, the input of size n is broken into two parts with size k and n-k

Recurrence relation: $T(n) = T(n-k) + T(k) + n$, with $T(1) = 1$

Best case:

The best case of quicksort is when the selected pivot is a mean element. So the recurrence relation becomes $T(n) = T(n/2) + T(n/2) + n$, so $T(n) = nlogn$

Average case: $T(n) = nlogn$

Proof

Worst case:

The worst case happens when we select smallest or largest element as pivot. Recurrence relation: $T(n) = T(n-1) + T(1) + n$, so $T(n) = n^2$

3. EXP

a. Design a recursive algorithm for computing 2^n for any non-negative integer n that is based on the formula $2^n = 2^{n-1} + 2^{n-1}$.

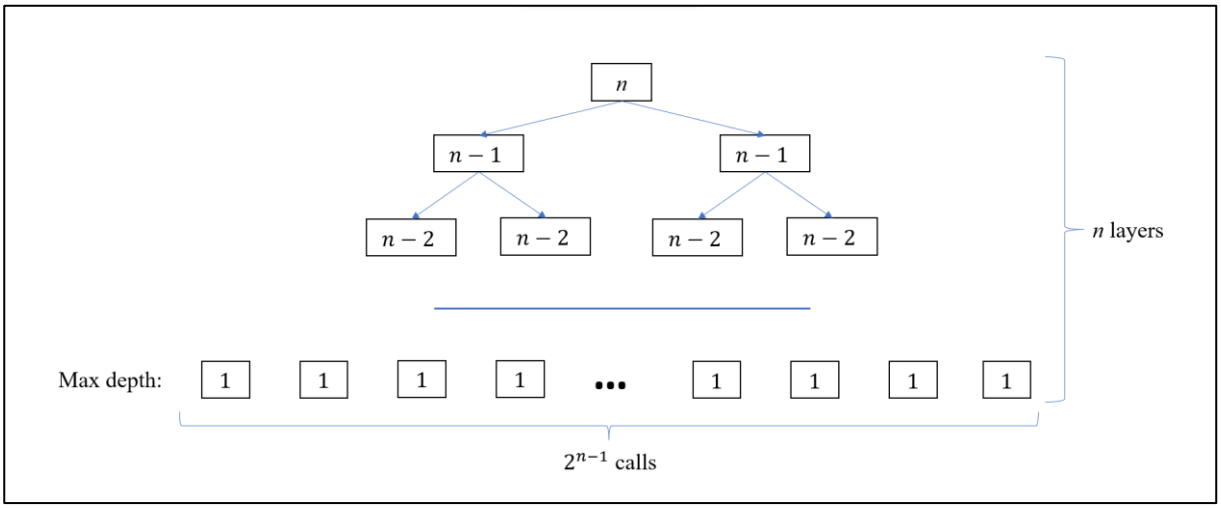
```
//Input: non − negative number n
calculate(n):
    if n == 1 then return 2
    return calculate(n − 1) ∗ calculate(n − 1)
```

b. Set up a recurrence relation for the number of additions made by the algorithm and solve it.

$T(n) = T(n-1) + T(n-1) + 1$, with $T(1) = 1$.

Using backward substitution we get: $T(n) = 1 + 2 + 4 + \cdots + 2^{n-1} = 2^n - 1$

c. Draw a tree of recursive calls for this algorithm and count the number of calls made by the algorithm.



The number of calls made: $1 + 2 + 4 + \cdots + 2^{n-1} = 2^n - 1$.

d. Is it a good algorithm for solving this problem?

No, we can change the mentioned formular to $2^n = 2 \times 2^{n-1}$. So the recurrence relation is $T(n) = T(n-1) + 1$, with $T(1) = 1$. After backward substitution we get $T(n) = n$