ANALYSIS OF
RECURSIVE ALGORITHMS

Lê Chí Cường – 21520012. Đoàn Nguyễn Trần Hoàn - 21520239

GROUP 13 KHTN2021

1. Tower of Hanoi

1. 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.)
2. How many moves are made by the ith largest disk (1 ≤ i ≤ n) in this algorithm?
3. Find a nonrecursive algorithm for the Tower of Hanoi puzzle and implement it in the language of your choice

Answer:

1. The minimum moves to solve Tower of Hanoi problem with 64 disks are $2^{64} - 1$ moves. Thus, it takes $2^{64} - 1\ minutes \approx 584942400000\ years.$
2. The number of moves are made by the 1st largest disk is 1 (from the first stack to the 3rd stack). Before that, the 2nd largest need to move 1 time (from first stack to 2nd stack), and after the largest disk move to the 3rd stack, the 2nd largest dist need to move one more time (from 2nd stack to 3rd stack) to complete the problem.

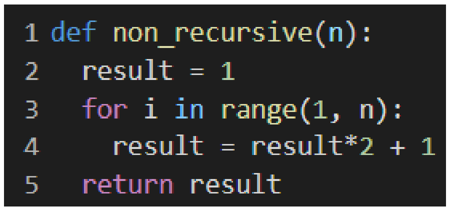
   Generally, when the ith largest disk move 1 time, the (i+1)th largest disk need to move at least 2 time. Hence, we have the following recurrent formula: $M(i+1) = 2 * M(i);\ M(1) = 1.$ (where *M(i)* is the least moves are made by the ith largest disk)

   Solve above formula, we have: $M(i) = 2^i$

3. None-recursive algorithm:

```
result ← 1
For i ← 2 to n:
    result ← 2*result + 1
```

   Implement in python:



```
def non_recursive(n):
  result = 1
  for i in range(1, n):
    result = result*2 + 1
  return result
```

2. QuickSort

Quicksort is one of the fastest sort-algorithm. Below is the example quicksort code.

```
def QuickSort(arr):
  if len(arr) <= 1:
    return arr
  else:
    pivot = arr[0]
    left = [x for x in arr[1:] if x <= pivot]
```

```
right = [x for x in arr[1:] if x > pivot]
return QuickSort(left) + [pivot] + QuickSort(right)
```

Set up a recurrence relation, with an appropriate initial condition, for the number of times the basic operation is executed for Quicksort algorithm. And solve it for the best case, worst case and average case, then conclude the time complexity for each case.

Answer:

- Assume that there are k numbers not greater than pivot. Then, we have the following recurrence relation: $T(n) = T(k) + T(n-k-1) + cn; T(0) = T(1) = 1$ (we need to add *n* for merging step).
- The worst case occurs when the partition process always picks the greatest or smallest element as the pivot. Thus, the recurrence relation become: $T(n) = T(n-1) + cn; T(0) = T(1) = 1$. Solve that relation, we have: $T(n) = \frac{cn(n-1)}{2} \in O(n^2)$
- The best case occurs when the partition process always picks the middle element as the pivot. The following is recurrence for the best case. Thus, the recurrence relation become: $T(n) = 2T\left(\frac{n}{2}\right) + cn; T(0) = T(1) = 1$. Solve that relation, we have: $T(n) \in \Omega(n.\log(n))$
- The average case is $\Theta(n.\log(n))$

3. EXP

a. Design a recursive algorithm for computing 2^n for any nonnegative integer n that is based on the formula $2^n = 2^{n-1} + 2^{n-1}$.
b. Set up a recurrence relation for the number of additions made by the algorithm and solve it.
c. Draw a tree of recursive calls for this algorithm and count the number of calls made by the algorithm.
d. Is it a good algorithm for solving this problem?

Answer:

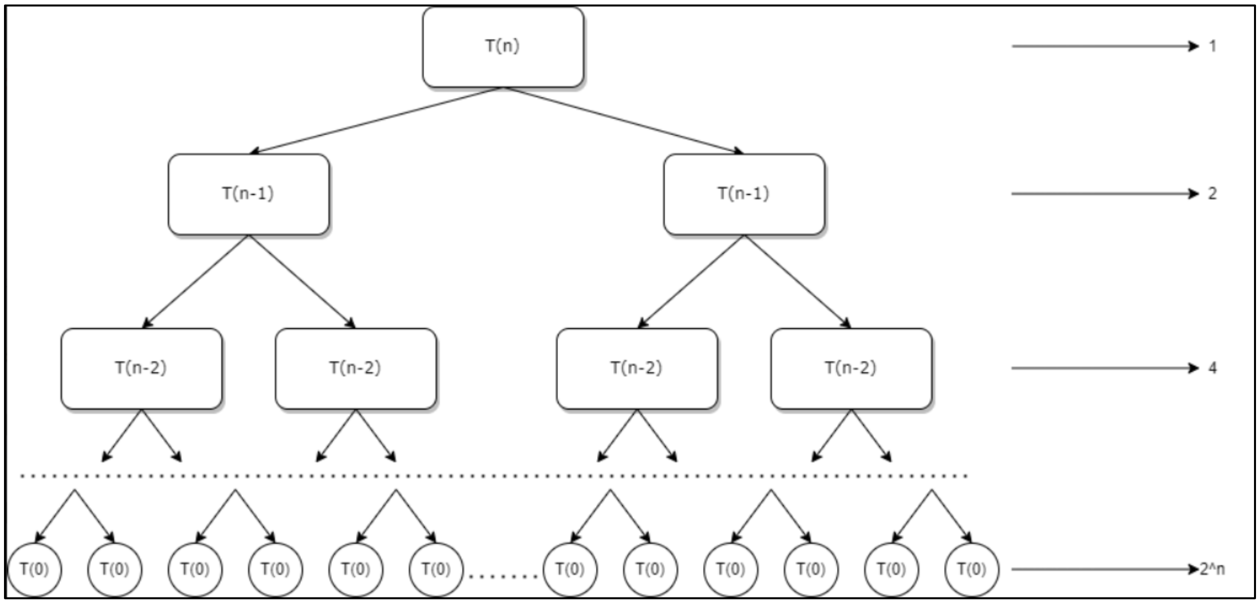
a. Psuedo code for algorithm:

```
Function power(n):
    If n <= 0:
        Return 1
    Else:
        Return power(n-1)*2
```

b. Recurrence relation: $T(n) = 2T(n-1) + 1; T(0) = 1$
Solve:

$$T(n) = 2T(n-1) + 1 = 4T(n-2) + 2 + 1 = 8T(n-3) + 4 + 2 + 1$$
$$= \cdots = 2^n T(0) + 2^{n-1} + \cdots + 2 + 1 = 2^n + 2^n - 1 = 2^{n+1} - 1$$

c. Recursive tree:



Number of calls: $1 + 2 + 4 + \cdots + 2^n = 2^{n+1} - 1$

d. It isn't a good algorithm for solving this problem, because there are many repeated steps.