Class: CS112.N21.KHTN

Member:

Nguyễn Trường Thịnh, 21520110

Huỳnh Phạm Đức Lâm, 21521050

Exercise: Team 2 (Phân tích độ phức tạp thuật toán đệ qui)

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 ($1 \le i \le n$) 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.

Number of moves: 2^64-1.

Time of move: (2^64-1)*1(minute).

1 year= 60*24*365 minutes.

=> Number of year to run this algorithm: (2^64-1)/(60*24*365)=35 tỷ năm.

2.

T(n)=T(n-1)+1+T(n-1) with T(0)=0.

assume that n-1 largest disk on 1th column(except the largest disk). We move it to 2th comlumn with T(n-1). Then move largest disk to 1th column. Finally move n-1 disk initial back with T(n-1).

3.

Create 3 stack: source, destination, auxiliary.

calculate total number of moves: 2^n-1.

If n is even, swap destination and auxiliary.

Run i from 1 to total number of moves:

    if i%3==1: move top disk from source to destination.

    if i%3==2: move top disk from source to auxiliary.

if i%3==0: move to disk from auxiliary to destination.

2. QuickSort

1. 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:

T(n): Time complexity solving Quicksort problem with input size of n.

Quicksort algorithm divide n into 2 subaray. Assign n and n-k.

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

Average case: O(nlogn)

Best case: O(nlogn)

Worst case: O(n^2)

3. EXP

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

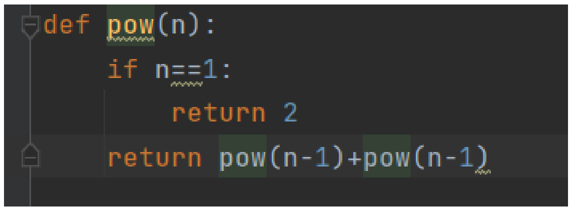
b. Set up a recurrence relation for the number of additions made by the algorithm and solve it.

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.



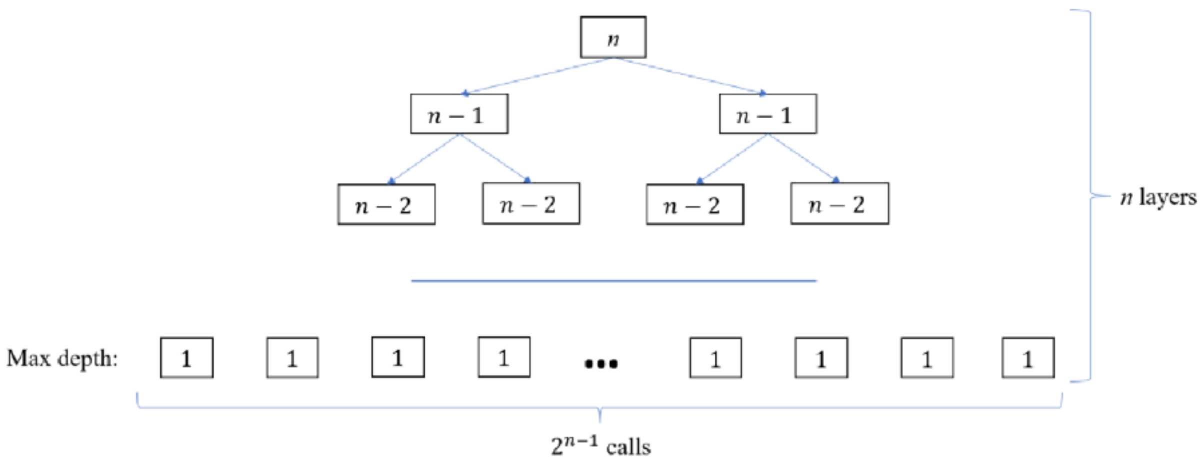
```
def pow(n):
    if n==1:
        return 2
    return pow(n-1)+pow(n-1)
```

b.

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

T(n)=2^n-1.

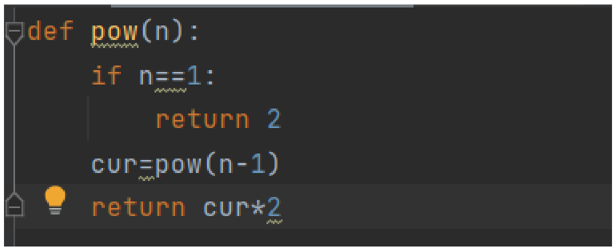
c.



d.

I suggest a algorithm having time complexity is O(n).

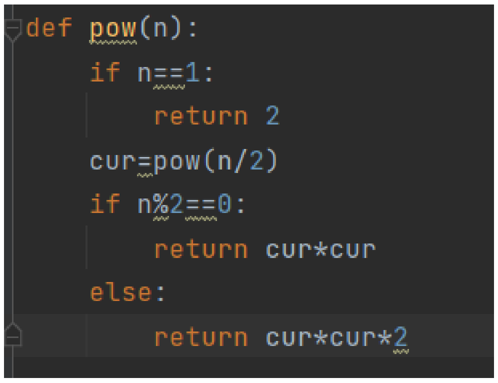
recurrence relation: T(n)=T(n-1)+1 with T(1)=1.



```
def pow(n):
    if n==1:
        return 2
    cur=pow(n-1)
    return cur*2
```

Or a algorithm having time complexity is O(logn).

recurrence relation: T(n)=T(n/2)+1 with T(1)=1.



```
def pow(n):
    if n==1:
        return 2
    cur=pow(n/2)
    if n%2==0:
        return cur*cur
    else:
        return cur*cur*2
```