Nhóm 3:

Mai Nhất Tùng - 21520523

Hà Trọng Tài - 21520436

Nguyễn Ngọc Nhung – 21521248

BÀI LÀM

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

1.

Xét bài toán với trên với n cái đĩa. Ta gọi Un là số bước tối thiểu để chuyển n cái đĩa từ cột này sang cột khác. Ta có:

- Để chuyển n đĩa từ cột 1 sang cột 2 ta cần chuyển đĩa lớn nhất sang cột 2 vậy phải chuyển n-1 đĩa trên cùng sang cột 3 tốn ít nhất U_{n-1} lần chuyển.
- Tiếp theo chuyển đĩa lớn nhất sang cột 2 tốn 1 lần chuyển.
- Cuối cùng chuyển n-1 đĩa từ cột 3 về cột 2 tốn ít nhất U_{n-1} lần chuyển.

Vậy ta có công thức truy hồi như sau $U_n = 2U_{n-1} + 1$ với $U_1 = 1$, ta có:

$$U_n + 1 = 2(U_{n-1} + 1) = \dots = 2^{n-1}(U_1 + 1) = 2^n$$

$$\Rightarrow \quad U_n = 2^n - 1$$

Vậy để chuyển được 64 đĩa từ cọc nguồn đến cọc dích phải mất ít nhất là $2^{64} - 1$ phút tương đương với 35100 tỷ năm.

Mã giả:

```
def move(n, source, dest, inter):
        if n == 1:
                move disk from source to dest
        else:
                move(n-1, source, inter, dest)
                move nth disk from source to dest
```

```
move(n-1, inter, dest, source)
```

2.

Số lần di chuyển các đĩa.

- Ta nhận thấy theo thuật toán trên thì đĩa lớn nhất chỉ bị di chuyển đúng một lần.
- Dựa vào thuật toán trên ta dễ dàng thấy được move(n-1) luôn được tuy hồi gấp đôi move(n)
- Vậy số lần di chuyển của các đĩa từ lớn tới nhỏ là: 1, 2, 4, 8, …, 2^n

3.

Ta có thể sử dụng thuật toán ngắn gọn sau:

```
col = ["source", "inter", "dest"]
if n%2==0:
  col[1], col[2] = col[2], col[1]
for i in range(1,2**n - 1):
   print(f"Move a disk from {col[(i & i-1) % 3]} to {col[(i | i-1) + 1) % 3]}")
```

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.

Độ phức tạp.

- T(1) = T(0) = 1, T(n) = T(k) + T(n-k-1) + n-1
- Worst case: trường hợp mà dãy số sắp xếp từ lớn tới bé khi đó mỗi lần chia theo pivot không có tác dụng do đó ta có:
  T(n) = T(n-1) + T(0) + n-1 = T(n-1) + n = … = 1 + 2 + … + n
  $\boldsymbol{T(n) = \frac{n(n+1)}{2}}$
- Average case: trường hợp pivot ở vị trí ngẫu nhiên (random pivot)
  $T(n) = T(k) + T(n-k-1)$
  $= \frac{1}{n}\sum_{k=1}^{n-1}\big(T(k) + T(n-k)\big)$
  $= \frac{2}{n}\sum_{k=1}^{n-1} T(k)$

⇨ $n * T(n) = 2\sum_{k=1}^{n-1} T(k)$ (1)

$(n-1) * T(n-1) = 2\sum_{k=1}^{n-2} T(k)$ (2)

Lấy (1) – (2), ta được:

$$n * T(n) - (n-1) * T(n-1) = 2 * T(n-1) + cn + c(n-1)$$

⇔ $n * T(n) = (n+1) * T(n-1) + 2 * cn$

Chia 2 vế cho n*(n+1)

⇨ $\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2c}{n+1}$

$\frac{T(n-1)}{n} = \frac{T(n-2)}{n-1} + \frac{2c}{n}$

$\frac{T(n-2)}{n-1} = \frac{T(n-3)}{n-2} + \frac{2c}{n-1}$

…

$\frac{T(1)}{2} = \frac{T(0)}{1} + \frac{2c}{2}$

⇨ $\frac{T(n)}{n+1} = \frac{T(0)}{1} + 2c * (\frac{1}{n+1} + \frac{1}{n} + \cdots + \frac{1}{2})$

⇔ **T(n)** $\approx \boldsymbol{2c * \log(n) * (n+1)}$

- Best case: trường hợp tốt nhất là pivot luôn nằm ở trung vị của dãy. Khi đó ta có:
  T(n) = 2T(n/2) + n-1
  T(n) $\approx$ ***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?

a)

- Ý tưởng: Ta có $2^n = 2^{n-1} + 2^{n-1}$. Xây dựng hàm đệ quy trả về powerOfTwo(n-1) + powerOfTwo(n-1). Nếu n = 0 thì trả về 1.

Mã giả:

```
Function powerOfTwo(int n)
{
    if ( n == 0 ) return 1
    return powerOfTwo(n-1) + powerOfTwo(n-1)
}
```

b)

T(n) = T(n-1) + T(n-1) + 1 = 2 * T(n-1) + 1 (với T(0) = 0)

T(n) = 2 * T(n-1) + 1

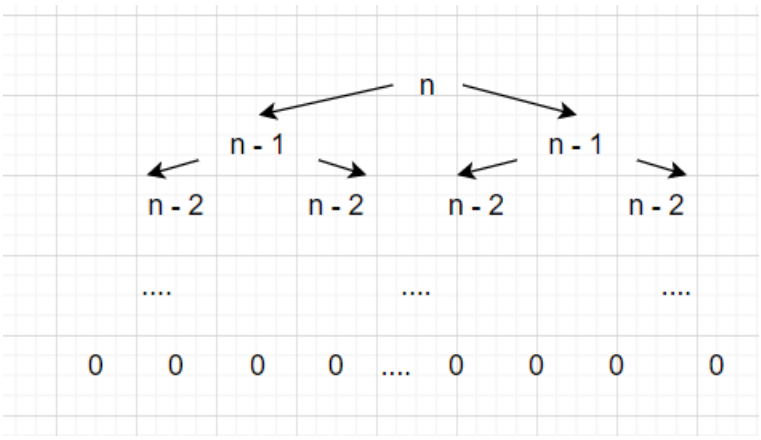
= 2^2 * T(n-2) + 2 + 1

…

= 2^n * T(0) + 2^{n-1} + … + 1

= 2^n - 1

c)



- Tổng số lần gọi hàm là $2^n + 1$, trong đó gọi đệ quy 2^n và 1 lần gọi đầu

d)

- Với số lần gọi hàm đệ quy là $2^n + 1$ và độ phức tạp $O(2^n)$. Thuật toán sẽ không phù hợp với giá trị n quá lớn
- Thay vào đó chúng ta có thể thực hiện dời bits cho bài toán tính giá trị cho 2^n này, $2^n = 0 << (n+1)$ (dời bit sang trái n+1 đơn vị).