# Bài toán nT(n) = (n-2)T(n-1) + 2, for n > 1 with T(1) = 1

2T(2) = (2-2)T(1) + 2 = 2 => T(2) = 1.

3T(3) = (3-2)T(2) + 2 = 3 => T(3) = 1.

4T(4) = (4-2)T(3) + 2 = 4 => T(4) = 1.

5T(5) = (5-2)T(4) + 2 = 5 => T(5) = 1.

=> T(n) = 1.

# 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

**Solution:**

1. In this problem we solved recurrence relation for tower of Hanoi. Supose that:

So the number of the years can be can be estimate as follow formula (With Y(n) is number of the years it will take if time to move one disk is 1 minute).

1. The number of moves are made by the ith largest disk (1 ≤ i ≤ n) in this algorithm can be derived by considering the recursive structure of the Tower of Hanoi algorithm. When we move a disk from one peg to another, we need to first move all the disks above it to a spare peg, then move the disk itself to the destination peg, and finally move all the disks above it from the spare peg to the destination peg. This means that the ith largest disk will be involved in a sequence of moves, as it will need to be moved from one peg to another times in order to complete the algorithm.

Another solution:

The 1th largest disk just make one move from source peg to des peg. 2th make 2 move from source to spare peg, and spare peg to source, 3th make 4 move repeat the same thing like 2th peg 2 times. And so on. We have number sequence is 1, 2, 4, 8, .... So the exact formula for problem is .

1. a nonrecursive algorithm for the Tower of Hanoi puzzle and implement it in C++.

Algorithm:

Step 1. Put n disk into stack, represent for n disks is in source peg in initial.

Step 2. Loop, if we can move top disk direct to des peg, do it. If stack is empty goto step 5.

Step 3. If we can’t move direct, it mean we must solve three subproblem first.

* Move n-1 disk from source to spare.
* Move 1 disk from source to des.
* Move n-1 disk from spare to des.

Step 4. End.

Implement. Use c++, stack, pair<int, char> - represent for the number of disk in <char> peg.

#include <iostream>

#include <stack>

using namespace std;

// Function to perform the Tower of Hanoi algorithm

void towerOfHanoi(int n, char source, char auxiliary, char destination) {

    stack<pair<int, char>> s;

    s.push(make\_pair(n, source));

    while (!s.empty()) {

        int disks = s.top().first;

        char from\_peg = s.top().second;

        s.pop();

        if (disks == 1) {

            cout << "Move disk 1 from peg " << from\_peg << " to peg " << destination << endl;

        }

        else {

            char to\_peg = destination;

            destination = auxiliary;

            auxiliary = to\_peg;

            s.push(make\_pair(disks-1, auxiliary));

            s.push(make\_pair(1, from\_peg));

            s.push(make\_pair(disks-1, destination));

        }

    }

}

int main() {

    int n;

    cout << "Enter the number of disks: ";

    cin >> n;

    towerOfHanoi(n, 'A', 'B', 'C');

    return 0;

}

# 2. QuickSort

Recurrence relation for quick sort is T(n) = T(k) + T(n – k - 1) + n, T(0) = T(1) = 1 (1)

where k is the index of the pivot element (in sorted array) chosen by the algorithm.

* Best case: k is median, it mean that T(k) = T(n-k-1) = T(n/2). (Note the we solve with n = ). Recurrance relation become T(n) = 2T(n/2) + n, solve like merge sort. T(n) = nlogn. Time complexity is (nlogn).
* Worst case: k is n-1 or k = 0, it mean that T(k) + T(n-k-1) = T(n-1). Recurrance relation become T(n) = T(n-1) + n, use telescope T(n) = . Time complexity is O(n^2).
* Average case quite hard:

If k is a random pivot we can calcute average by . both is the same, we can rewite (1) like this:

Multiply both side by n

Subtract same formula for n-1:

Collect:

Divede by n(n+1)

Ignore , telescope

Now this time for estimate time complexity:

Time complexity for this case is O(nlogn).