**4- Tower Of Hanoi**

**Problem Description**

There are eight disks of different sizes and four pegs. Initially, all the disks are on the first peg in order of size, the largest on the bottom and the smallest on the top. It’s required to use divide and conquer algorithm to transfer all the disks to another peg by a sequence of moves.

The constraints are that only one disk can be moved at a time, and it is forbidden to place a larger disk on top of a smaller one.

Also it’s required to use dynamic programming algorithm to transfer all the disks to another peg in 33 moves

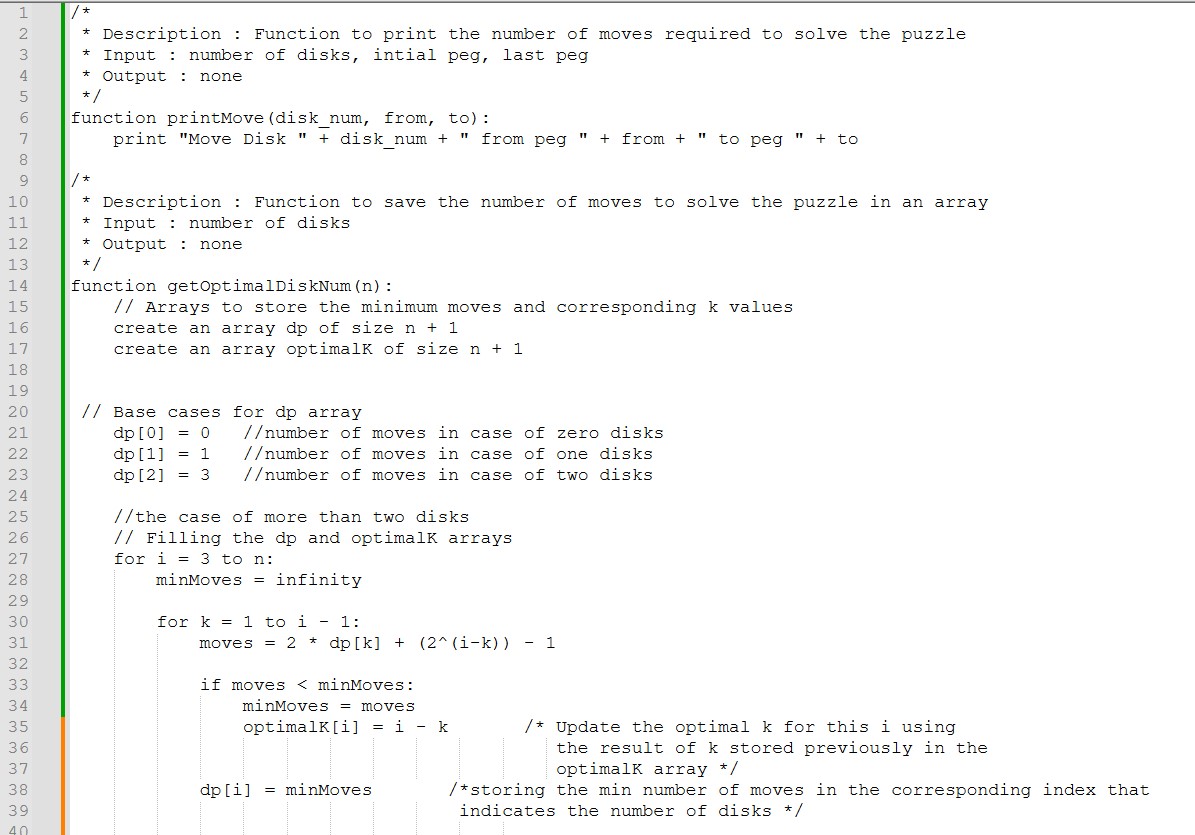
**Detailed Assumptions**

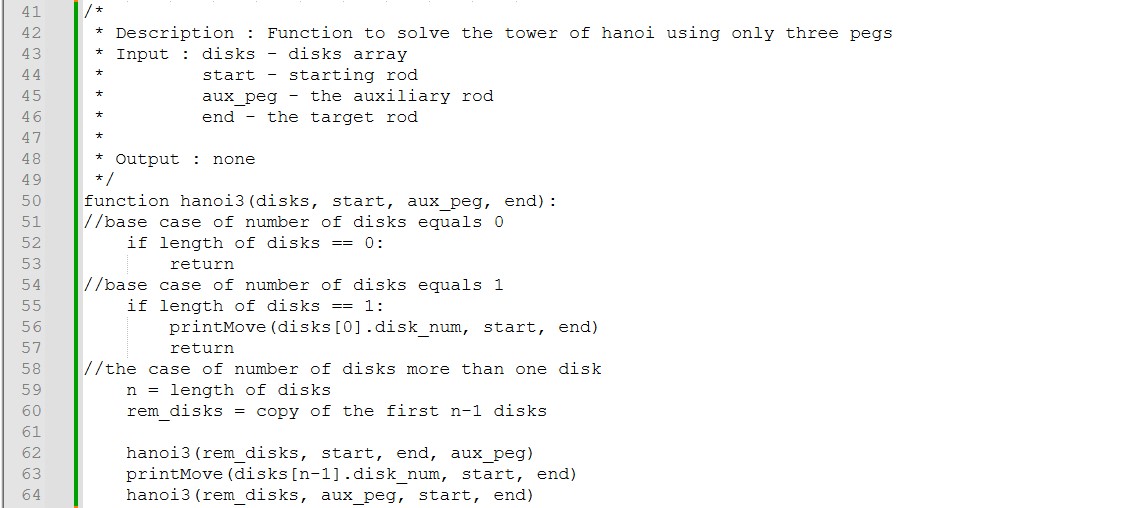
* The disks are labeled with consecutive integers starting from 1. Each disk is represented by an object of Disk class, which has a disk\_num attribute.
* The moves variable is used to keep track of the total number of moves performed during solving the Tower of Hanoi Problem.
* The Tower of Hanoi problem is being solved optimally, it aims to minimize the number of moves required to transfer the disks from the starting peg to the end peg.
* The line that assigns k to the optimal value of k that is resulted in the function of getOptimalDiskNum is commented in the code if divide and conquer method is used an the value of k will be assigned to n/2 where n is the number of disks
* The line that assigns k to n/2 is commented in case of using dynamic programming and the value of k will be assigned to the value resulted in OptimalK array in the getOptimalDiskNum function

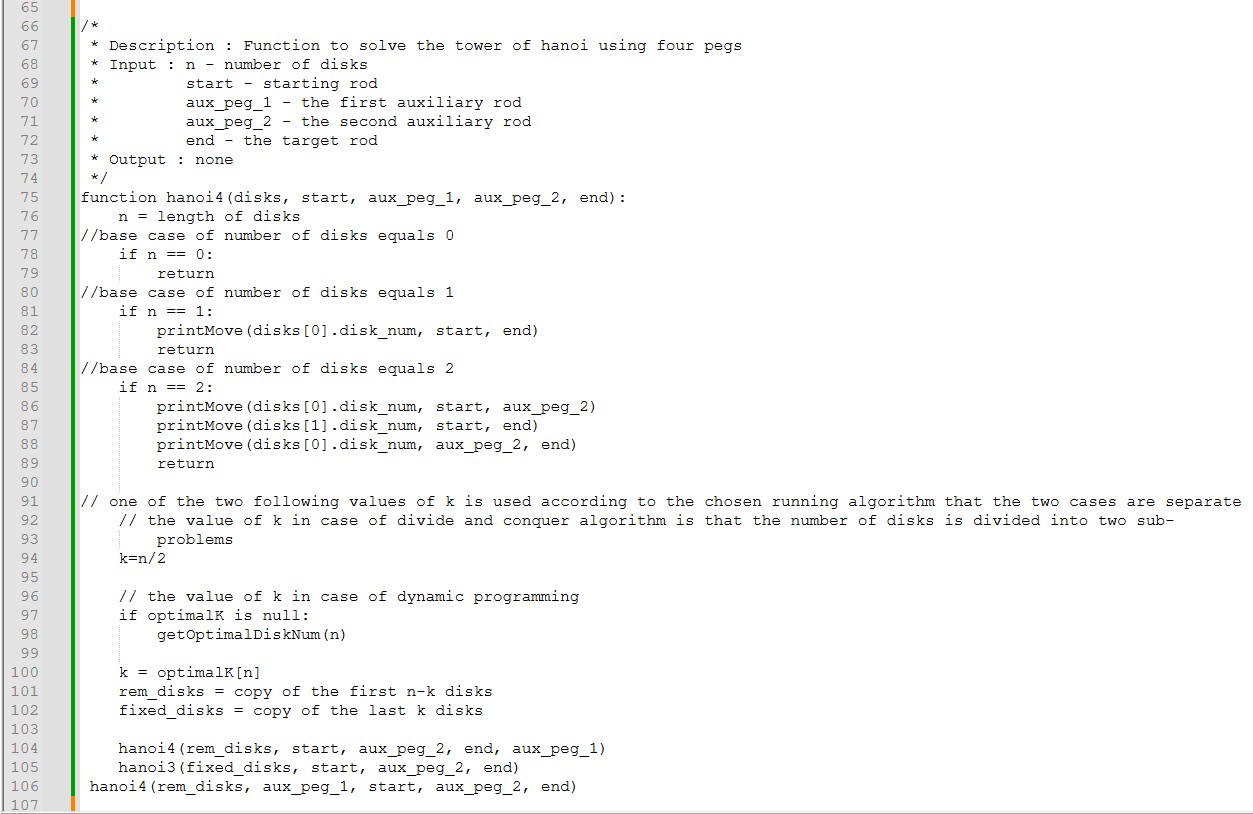
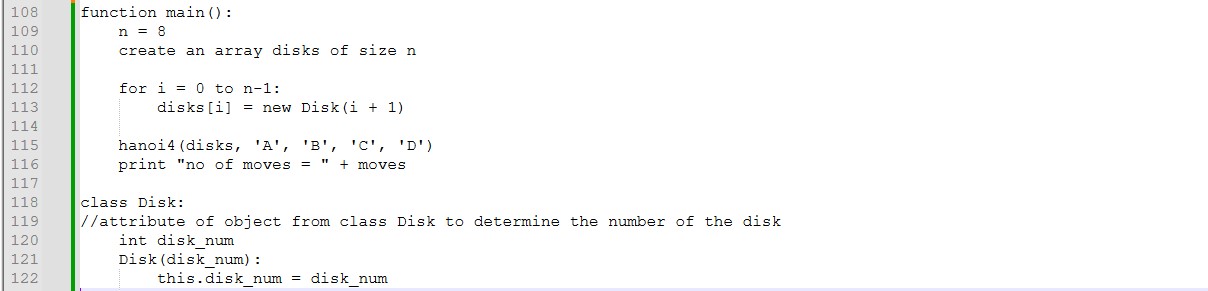
**Detailed Solution (pseudocode and steps description)**

1. Initialize an array dp of size n+1 to store the number of moves required to transfer the disks from the starting peg to the end peg
2. Initialize an array optimalK of size n+1 to store to optimal value of k that will be used to divide the disks in dynamics programming technique while in divide and conquer the disks will be divided into halves
3. The code keep divide the disks into sub-problems until reaches the base case to transfer the disks to another peg

Both divide and conquer algorithm and dynamic programming algorithm are implemented in the following pseudocode







**Complexity Analysis**

1. getOptimalDiskNum function:

* Time Complexity: O(n^2)
* Space Complexity: O(n)

1. hanoi3 function:

* Time Complexity: O(2^n)
* Space Complexity: O(n)

1. hanoi4 function:

* Time Complexity: O(2^n)
* Space Complexity: O(n)

1. main function:

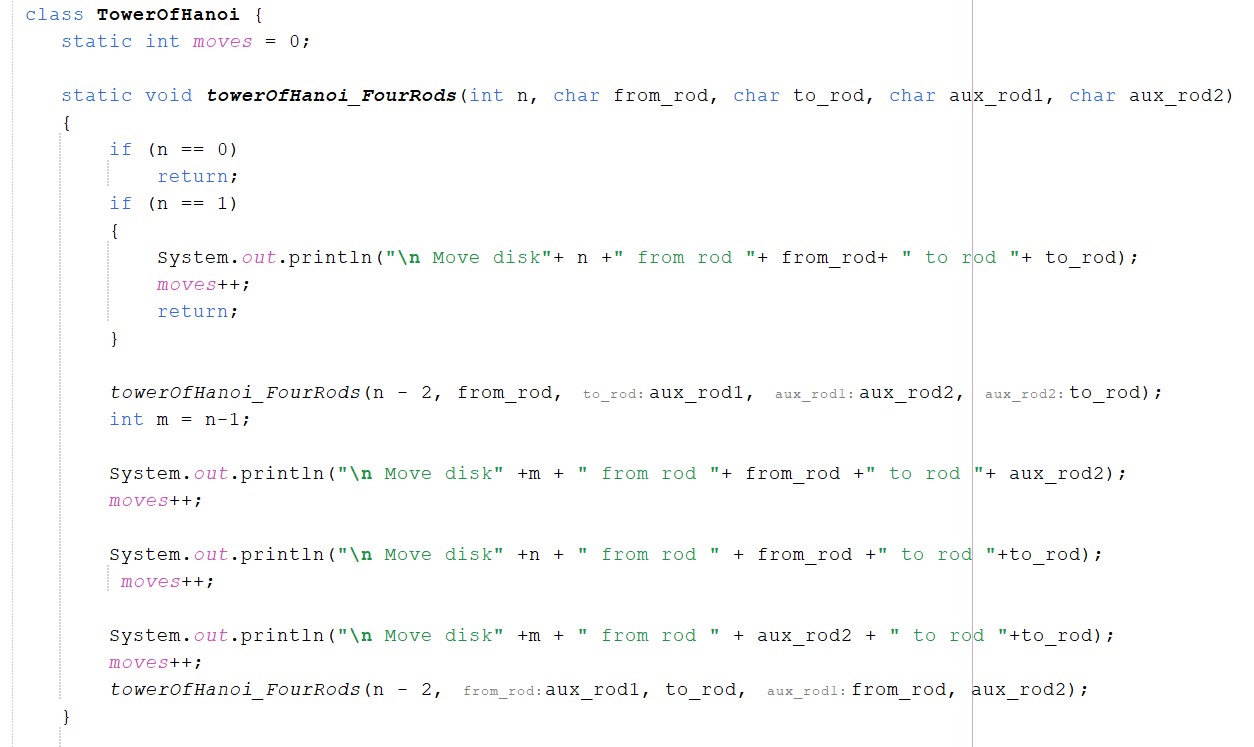
* Time Complexity: O(2^n)
* Space Complexity: O(n)

The resulted Time Complexity: O(2^n)

The resulted Space Complexity: O(n)

**Comparison**

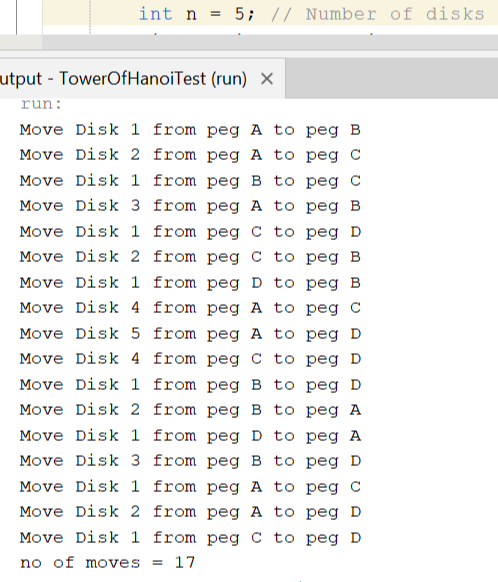
The solution using decrease and conquer:



|  |  |  |
| --- | --- | --- |
|  | Divide and Conquer | Decrease and Conquer |
| Advantages | Efficient for large problem size in case the number of disks increase | Simplicity as it reduces the problem by one until reaches the base case  Lower Memory requirements as it often operate on the problem in place |
| Disadvantages | Overhead of combining sub-problems  Increase memory usage | May not be efficient for large problems |
| Number of moves | Transfers 8 disks in 33 moves | Transfers 8 disks in 41 moves |

**Sample of Output**

- **Case of five disks**

The five disks are divided into two sub-problems of size three and two disks based of the value of optimal k

The upper three disks are ordered using the four rods in the first call of function hanoi4

The lower two disks are ordered using the remaining three rods in the call of function hanoi3

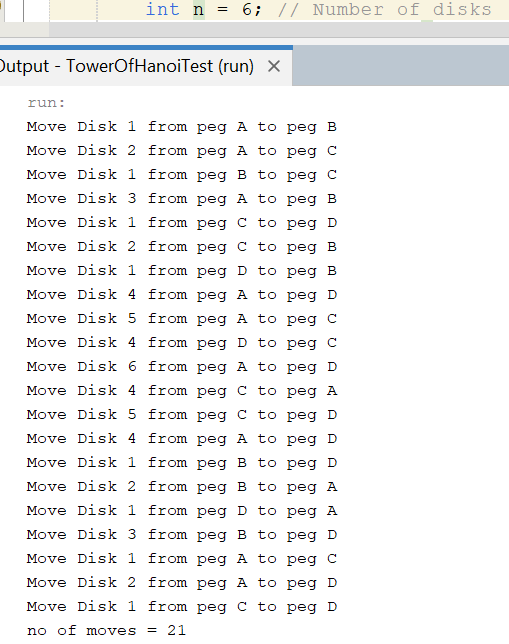
The upper three disks move back again to be ordered above the bottom two disks using the four rods in the second call of function hanoi4

The number of moves is calculated based on dynamic programming that stores the result of the base case which are 0,1, and 2 disks in

Figure(): Output of 5 disks

0, 1, and 3 moves then storing the number of moves in the dp array to be used in the next number of disks

-**Case of six disks**

The six disks are divided into two sub-problems of equal size of three disks each based of the value of optimal k

The upper three disks are ordered using the four rods in the first call of function hanoi4

The lower three disks are ordered using the remaining three rods in the call of function hanoi3

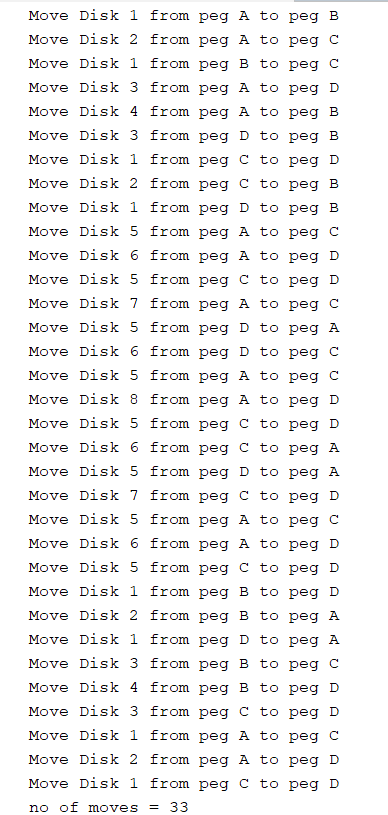
The upper three disks move back again to be ordered above the bottom three disks using the four rods in the second call of function hanoi4

The number of moves is calculated based on

dynamic programming that stores the result of the base cases which are 0,1, and 2 disks in

0, 1, and 3 moves then storing the number of Figure(): Output of six disks

moves in the dp array to be used in the next number of disks

-**Case of eight disks**

The six disks are divided into two sub-problems of equal size of four disks each based of the value of optimal k

In the first call of hanoi4

The upper four disks are ordered by recursively calling hanoi4 and divide the four disks into two sub-problems of the same size of two disks each based on the value of optimal k

The upper two disks (disk 1 and 2) ordered on peg C using the four rods in the call of function hanoi4 while the lower two disks (disk 3 and 4) are ordered on peg B using the three rods in the call of function hanoi3 then the upper two disks are ordered above the lower two disks on peg B using the four rods in the call of function hanoi4

The lower four disks are ordered by calling hanoi3 and recursively reduce the problem by one into sub-problem until it reaches the base case which is two disks (disk 5 an disk 6) then order disk 7 then order disk 8 on peg D

In the second call of hanoi4Figure(): Output of eight disks

The upper four disks are ordered by recursively calling hanoi4 and again divide the four disks into two sub-problems of the same size of two disks each based on the value of optimal k

The upper two disks (disk 1 and 2) ordered on peg A using the four rods in the call of function hanoi4 while the lower two disks (disk 3 and 4) are ordered on peg D using the three rods in the call of function hanoi3 then the upper two disks are ordered above the lower two disks on peg D using the four rods in the call of function hanoi4.

**Conclusion**

In Conclusion, using Divide and Conquer with the Dynamic Programming approach to divide the problem to two sub-problems and calculate the minimum number of moves required for different number of disks and storing them in the memory leads to optimizing the solution that offers a modified approach to solving the Tower of Hanoi Problem using four pegs, allowing more efficient solution compared to the traditional approach.

**References**

The Four- Peg Tower of Hanoi Puzzle by Richard Johnsonbaugh.