Recursion

Unit 6

Recursion

The function that calls itself is known as recursion.

```
void function reuse(){
        ……..
        recurse();
        ……..
}
```

Recursion	Iteration
Recursion is like piling all of those steps on top of each other and then quashing them all into the solution	In iteration, a problem is converted into a train of steps that are finished one at a time, one after another
In recursion, each step replicates itself at a smaller scale, so that all of them combined together eventually solve the problem.	With iteration, each step clearly leads onto the next, like stepping stones across a river
The not all-recursive problem can be solved by iteration	Any iterative problem is solved recursively
Slow Performance	Fast Performance
Recursion is always applied to functions	Iteration is applied to iteration statements or “loops”.
Small line of code	Large Line of code

Advantages of Recursion

. The main benefit of a recursive approach to algorithm design is that it allows programmers to take advantage of the repetitive structure present in many problems.

ii. Complex case analysis and nested loops can be avoided.

iii. Recursion can lead to more readable and efficient algorithm descriptions.

iv. Recursion is also a useful way for defining objects that have a repeated similar structural form.

Disadvantages

i. Slowing down execution time and storing on the run-time stack more things than required in a non recursive approach are major limitations of recursion.

ii. If recursion is too deep, then there is a danger of running out of space on the stack and ultimately program crashes.

iii. Even if some recursive function repeats the computations for some parameters, the run time can be prohibitively long even for very simple cases.

Some examples of recursion

- Factorial of a number
- Fibonacci series
- Tower of Hanoi

Tower of Hanoi

It is one of the main application of recursion whose objective is to transfer all disks from origin pole to destination pole using intermediary pole as a temporary storage

Condition:

Move one disk at a time

Each disk must be placed around the pole

Never place larger disk on top of smaller disk

Algorithm (TOH)

To move n disks from A to C, using B as auxiliary

- If *n=1*, move the single disk from A to *C* and stop.
- Move the top *n-1* disks from *A* to *B*, using *C* as auxiliary.
- Move the remaining disk from *A* to *C*.
- Move the *n-1* disks from *B* to *C*, using A as auxiliary.

Algorithm for TOH

Let’s consider move ‘n’ disks from source peg (A) to destination peg (C), using intermediate peg (B) as auxiliary.

1. Assign three pegs A, B & C
2. If n==1

Move the single disk from A to C and stop.

3. If n>1

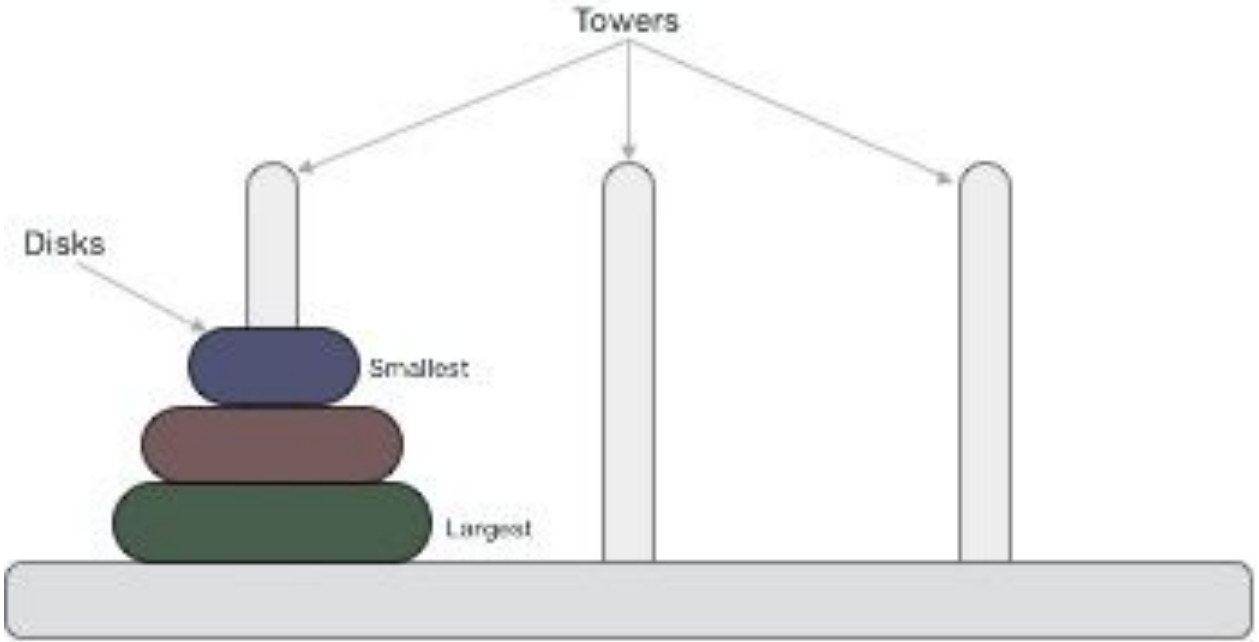
a) Move the top (n-1) disks from A to B.

b) Move the remaining disks from A to C

c) Move the (n-1) disks from B to C

4. Terminate

Problem is to transfer the disks from left most tower to right most tower with the help of middle tower(intermediary tower)



Following are the steps to solve the TOH problem for 3 disks

Move disk 1 from tower A to tower C

Move disk 2 from tower A to tower B

Move disk 1 from tower C to tower B

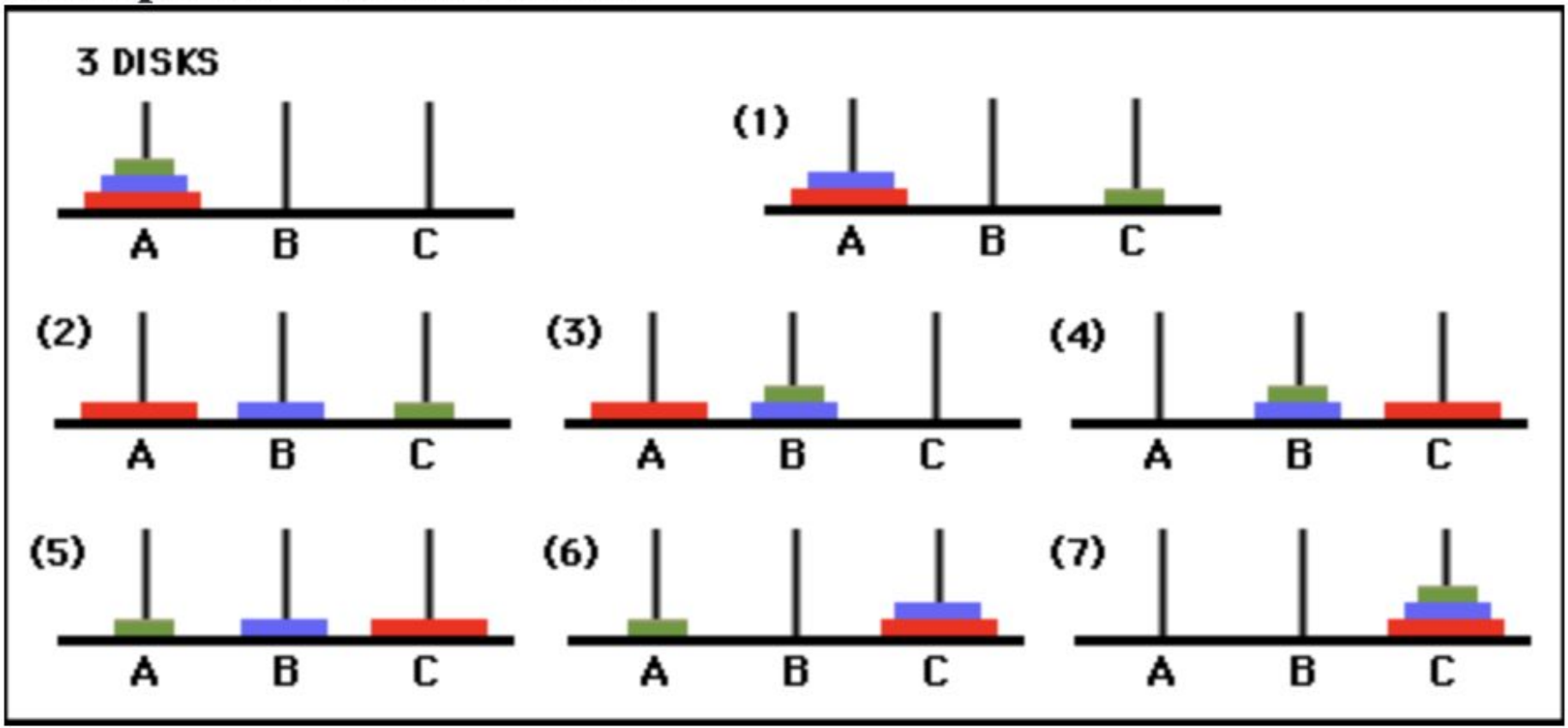
Move disk 3 from tower A to tower C

Move disk 1 from tower B to tower A

Move disk 2 from tower B to tower C

Move disk 1 from tower A to tower C

Example for 3 disks: 7 moves



Pseudocode for TOH

```
void towers(int n,char fromtower,char totower,char auxtower) { /* If only 1 disk, make the move and return */
if(n==1) {
printf("\nMove disk 1 from tower %c to tower %c",fromtower,totower);
return;
}
/* Move top n-1 disks from A to B, using C as auxiliary
towers(n-1,frompeg,auxtower,totower);
/* Move remaining disks from A to C */
printf("\nMove disk %d from tower %c to tower %c",n,fromtower,totower);
/* Move n-1 disks from B to C using A as auxiliary */
towers(n-1,auxtower,totower,fromtower);
}
```