CMSC 132, Object-Oriented Programming II **Summer 2018**

Lecture 10:

Lecturer: Anwar Mamat

Disclaimer: *These notes may be distributed outside this class only with the permission of the Instructor.*

10.1 RECURSION

Recursion in computer science is a method where the solution to a problem depends on solutions to smaller instances of the same problem.

Listing 1: General format of many recursive algorithms

```
if(some condition for which the answer is known){
        return solution; // base case
}else{
        recursive function call //smaller version of the same problem
}
```

10.2 Examples

10.2.1 Factorial

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ (n-1)! \times n & \text{if } n > 0 \end{cases}$$

Listing 2: Factorial

```
public int fact(int n){
        if(n == 0) return 1;
        return n * fact(n-1);
    }
```

10.2.2 GCD

$$gcd(a,b) = \begin{cases} a & \text{if } b \text{ is } 0 \\ gcd(b, a\%b) & \end{cases}$$

Listing 3: Factorial

```
public int gcd(int a, int b){
        if(b == 0){return a;}
        return gcd(b, a%b);
}
```

10.2.3 Print a linked list

Listing 4: Print linked list

```java
public void print(Node h){
        if(h == null){return;}  //base case
        System.out.print(h.data+",");
        print(h.next);
    }
```

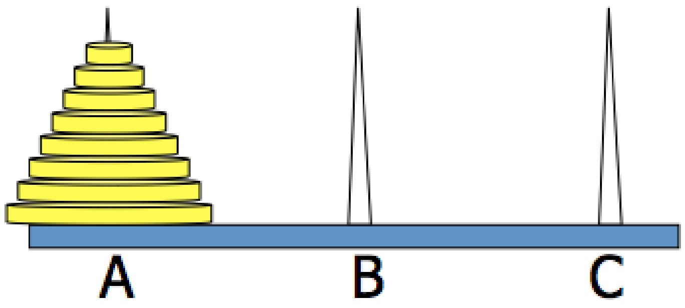
Listing 5: Print linked list in reverse order

```java
public void print(Node h){
        if(h == null){return;}  //base case
        print(h.next);
        System.out.print(h.data+",");
}
```

10.2.4 The Towers of Hanoi

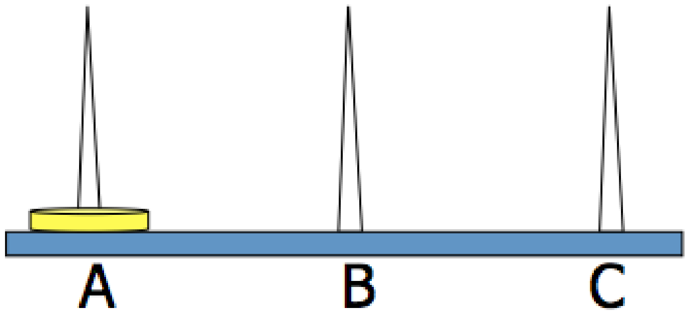
Legend has it that there were three diamond needles set into the floor of the temple of Brahma in Hanoi. Stacked upon the leftmost needle were 64 golden disks, each a different size, stacked in concentric order, as shown in Figure 10.2. The monks were to transfer the disks from the first needle A to the second needle B, using the third C as necessary. But they could only move one disk at a time, and could never put a larger disk on top of a smaller one. When they completed this task, **the world would end!**

Figure 10.1: Towers of Hanoi



10.2.4.1 Base case: one disk only

Figure 10.2: One Disk

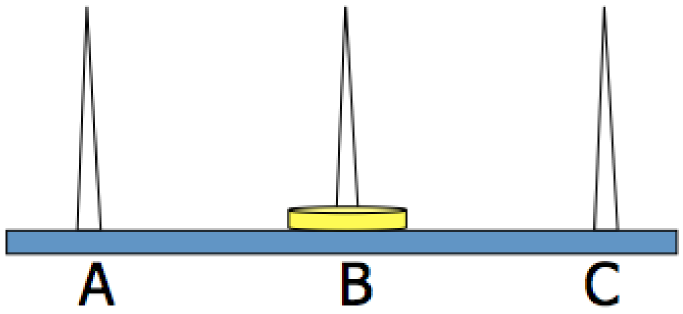


If there is only one disk, it is trivial. We just move the disk from A to B as shown in Figure 10.3.

Listing 6: One Disk

```
1. Move from A to B.
```

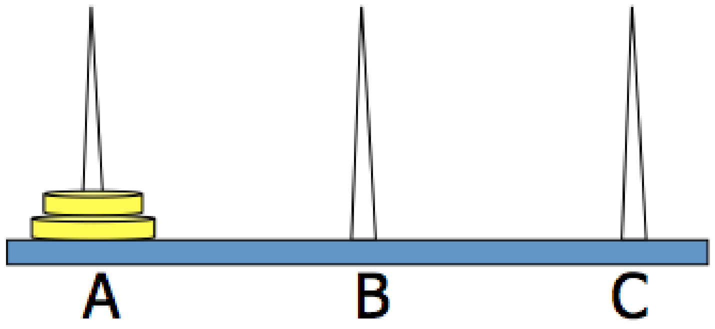
Figure 10.3: One Disk



10.2.4.2 Two Disks

You know how to move one disk. Now we move two disks. We have two disks as shown in Figure 10.4.

Figure 10.4: Two Disks



Here are the steps:

Listing 7: Two Disks

```
1. Move from A to C.
2. Move from A to B.
3. Move from C to B.
```

Figure 10.5: Two Disks: Move from A to C

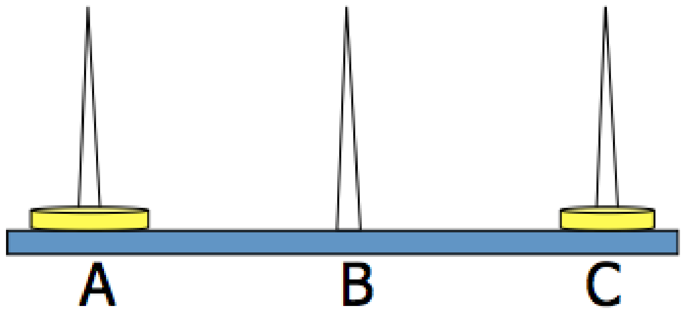


Figure 10.6: Two Disks: Move from A to B

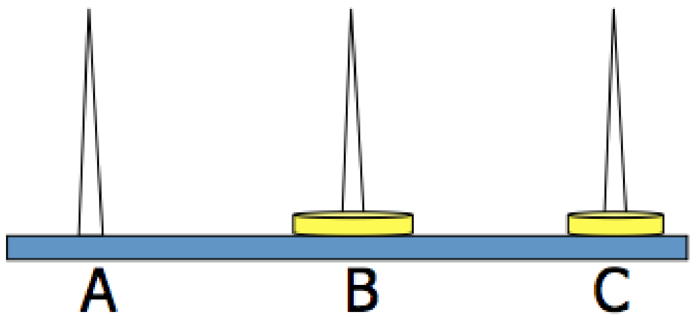
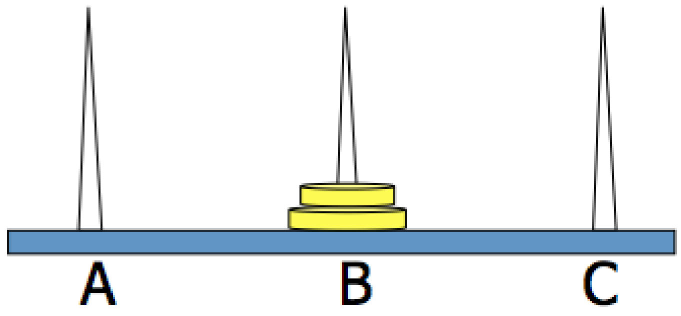


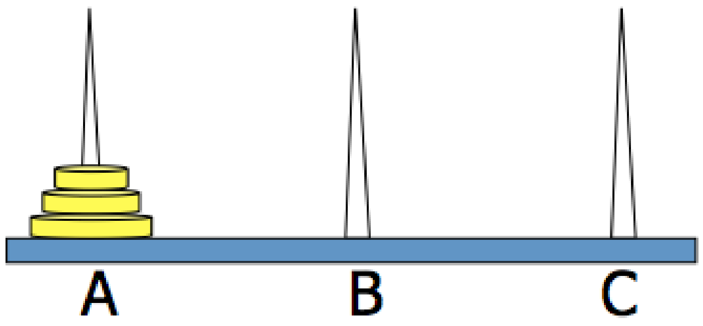
Figure 10.7: Two Disks: Move from C to B



10.2.4.3 Three Disks

You know how to move two disks, right? We just saw it. Now, we move three disks, as shown in Figure 10.11. Here are the steps:

Figure 10.8: Three Disks



Listing 8: Three Disks

```
1. Move 2 disks from A to C.
2. Move 1 disk from A to B.
3. Move 2 disk from C to B.
```

How do you move two disks at once in step 1? Yes, we can move 2 disks using the method described in section 10.2.4.2.

Figure 10.9: Three Disks

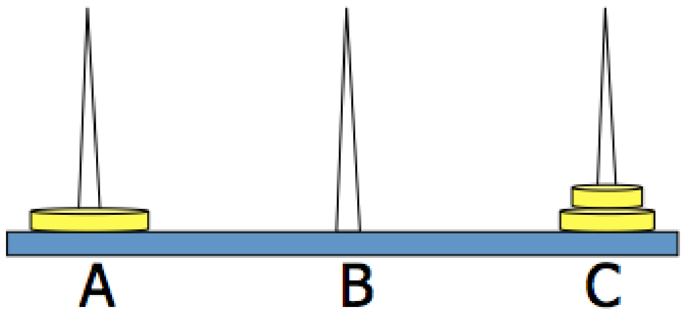


Figure 10.10: Three Disks

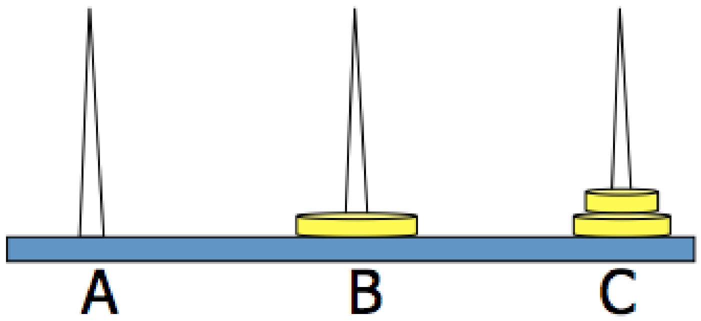
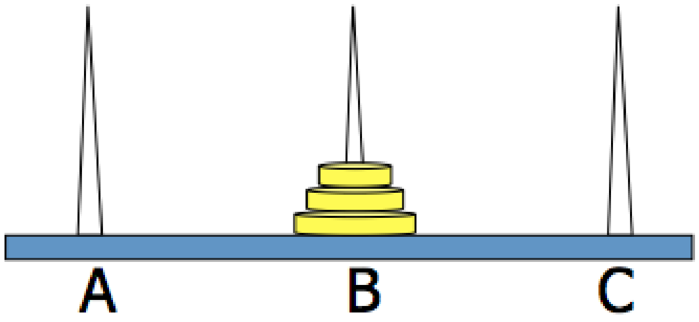


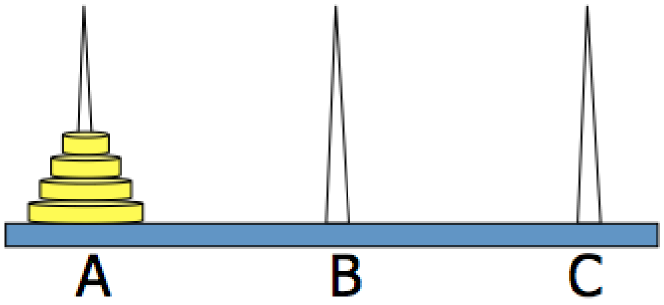
Figure 10.11: Three Disks



10.2.4.4 Four or more Disks

You know how to move three disks. Now, we move four disks, as shown in Figure 10.13.

Figure 10.12: Four Disks



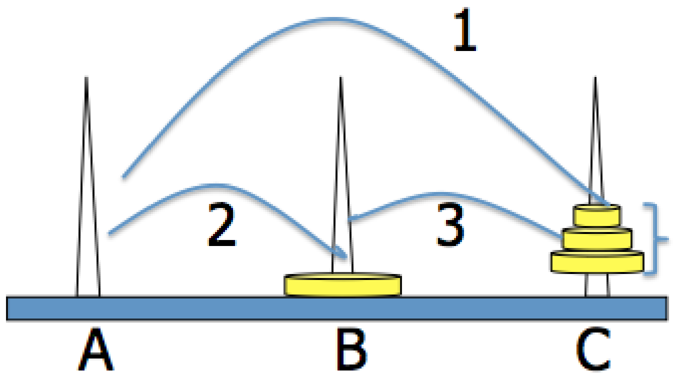
Here are the steps:

Listing 9: Four Disks

```
1. Move 3 disks from A to C.
2. Move 1 disk from A to B.
3. Move 3 disk from C to B.
```

How do you move three disks from A to C? We can do that using the exact same method we described in section 10.2.4.3.

Figure 10.13: Four Disks



If we generalize the method, in order to move n disks from A to B, the steps are:

Listing 10: n disks

```
1. Move n-1 disks from A to C.
2. Move 1 disk from A to B.
3. Move n-1 disk from C to B.
```

Here is the code for the Towers of Hanoi.

Listing 11: Towers of Hanoi

```java
import java.util.Scanner;
public class TowersOfHanoi {
  public void solve(int n, String A, String C, String B) {
       if (n == 1) {
           System.out.println(A + "->" + B);
       } else {
           solve(n - 1, A, B, C);
           System.out.println(A + "->" + B);
           solve(n - 1, C, A, B);
       }
   }

   public static void main(String[] args) {
       TowersOfHanoi towersOfHanoi = new TowersOfHanoi();
       System.out.print("Enter␣number␣of␣discs:␣");
       Scanner scanner = new Scanner(System.in);
       int discs = scanner.nextInt();
       towersOfHanoi.solve(discs, "A", "C", "B");
   }
}
```

10.2.4.5 How long it will take to move 64 disks?

Lets see how many moves it takes to solve this problem, as a function of n, the number of disks to be moved.

n	Number of disk-moves required
1	1
2	3
3	7
4	15
5	31
...	...
i	$2^i - 1$
64	$2^{64} - 1$ (a big number)

10.2.5 Palindrome

Listing 12: Palindrome

```java
public static boolean palindrome(String s){
        if(s.length() == 1 || s.length() == 0){
            return true;
        }
        if(s.charAt(0) != s.charAt(s.length()-1)) return false;
        return palindrome(s.substring(1,s.length()-1));
    }
```

10.2.6 Fibonacci

Listing 13: Fibonacci

```java
    //recursive implementation of Fibonacci. Extremely slow
    public static int fib1(int n){
        System.out.println("working␣hard␣" + count++ + "␣times");
        if(n == 1) return 1;
        if(n == 2) return 1;
        return fib1(n-1)+ fib1(n-2);
    }
     //iterative implementation of Fibonacci. Linear time
    public static int fib2(int n){
        int a = 0;
        int b = 1;
        int t = 1;
        for(int i =1; i < n; i++){
            t = a + b;
            a = b;
            b = t;
        }
        return t;

    }



    //BigInteger example
    public static BigInteger fib3(int n){
        BigInteger a = new BigInteger("0");
        BigInteger b = new BigInteger("1");
        BigInteger t = new BigInteger("1");
        for(int i =1; i < n; i++){
            t = a.add(b);
            a = b;
            b = t;
        }
```

```
            return t;
        }
        //Test
        public static void main(String[] args) {
            int n = 10000;
            BigInteger f = fib3(n);
            System.out.println(f);
        }
```

Listing 14: Recursive Fibonacci with Memoization

```
public Map<Integer, Integer> fibo;
public int fib(int n){
  int f1=0,f2=0;
  if( (n == 1)|| (n == 2)) return 1;
  else{
        if(fibo.containsKey(n-1)) f1 = fibo.get(n-1);
        else{
           f1 = fib(n-1);
           fibo.put(n-1,f1);
        }
        if(fibo.containsKey(n-2)) f2 = fibo.get(n-2);
       else{
           f2 = fib(n-2);
           fibo.put(n-2,f2);
        }
  return f2+f1;
   }
}
```

10.2.7 Recursive Tree

http://introcs.cs.princeton.edu/java/23recursion/Tree.java.html

10.2.8 Maze

http://algs4.cs.princeton.edu/41undirected/Maze.java.html