CIS 2168 Data Structures **Fall 2014**

Lecture 8: Oct. 2 & 4

Lecturer: Anwar Mamat

8.1 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

```
1 if(some condition for which the answer is known){
2         return solution; // base case
3 }else{
4         recursive function call //smaller version of the same problem
5 }
```

8.2 Examples

8.2.1 Factorial

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

Listing 2: Factorial

```
1 public int fact(int n){
2         if(n == 0) return 1;
3         return n * fact(n-1);
4     }
```

8.2.2 GCD

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

Listing 3: Factorial

```
1 public int gcd(int a, int b){
2         if(b == 0){return a;}
3         return gcd(b, a%b);
4 }
```

8.2.3 Print a linked list

Listing 4: Print linked list

```
1 public void print(Node h){
2         if(h == null){return;}   //base case
3         System.out.print(h.data+",");
4         print(h.next);
5     }
```

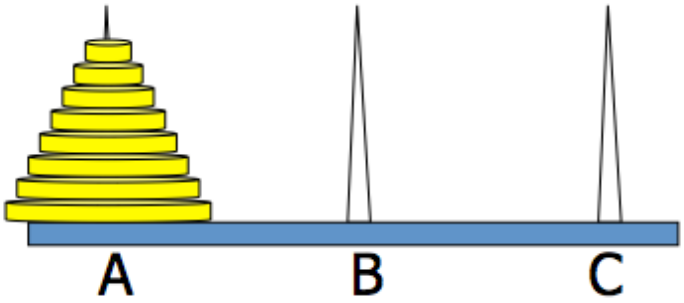
Listing 5: Print linked list in reverse order

```
1 public void print(Node h){
2         if(h == null){return;}   //base case
3         print(h.next);
4         System.out.print(h.data+",");
5 }
```

8.2.4 The Towers of Hanoi

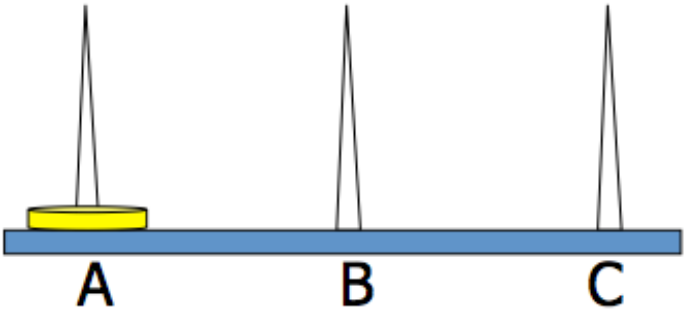
Legend has it that there were three diamond needles set into the floor of the temple of Brahma in Hanoi. Stacked upon the leftmost needle were 64 golden disks, each a different size, stacked in concentric order, as shown in Figure 8.2. The monks were to transfer the disks from the first needle A to the 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 8.1: Towers of Hanoi



8.2.4.1 Base case: one disk only

Figure 8.2: One Disk

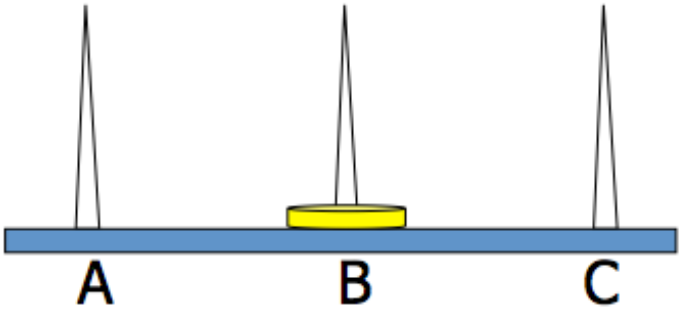


If there is only one disk, it is trivial. We just move the disk from A to B as shown in Figure 8.3.

Listing 6: One Disk

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

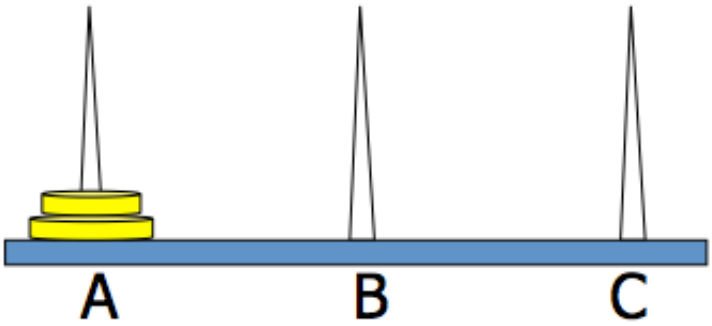
Figure 8.3: One Disk



8.2.4.2 Two Disks

You know how to move one disk. Now we move two disks. We have two disks as shown in Figure 8.4.

Figure 8.4: 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 8.5: Two Disks: Move from A to C

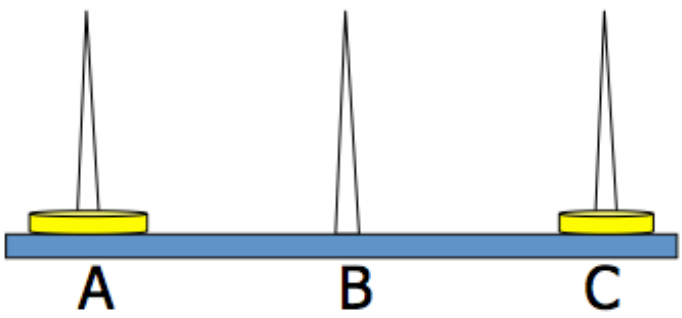


Figure 8.6: Two Disks: Move from A to B

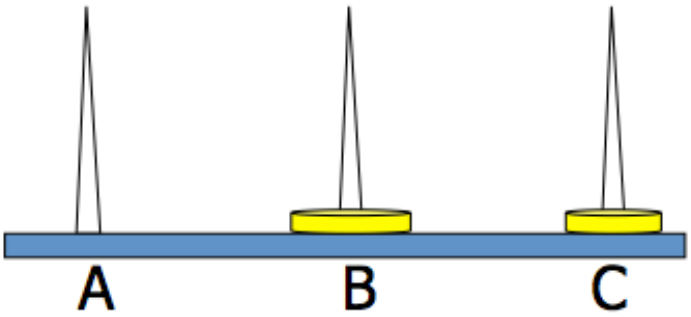
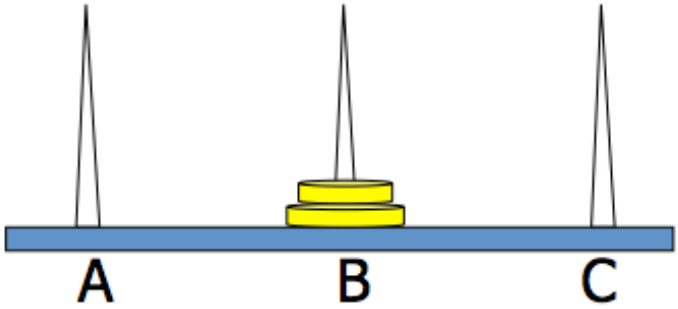


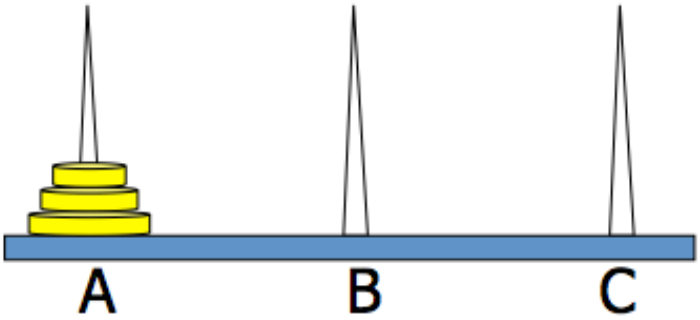
Figure 8.7: Two Disks: Move from C to B



8.2.4.3 Three Disks

You know how to move two disks, right? We just saw it. Now, we move three disks, as shown in Figure 8.11. Here are the steps:

Figure 8.8: 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 8.2.4.2.

Figure 8.9: Three Disks

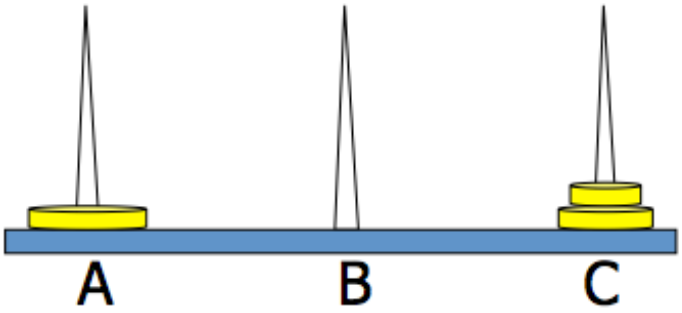


Figure 8.10: Three Disks

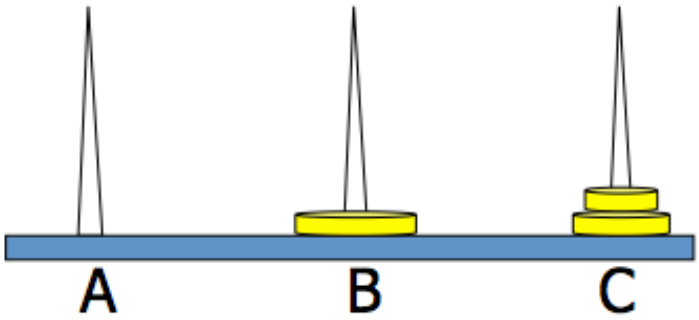
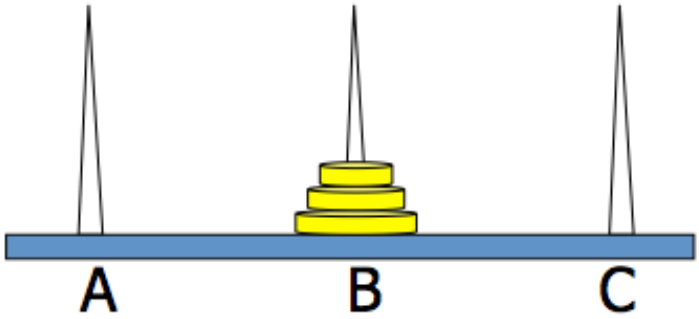


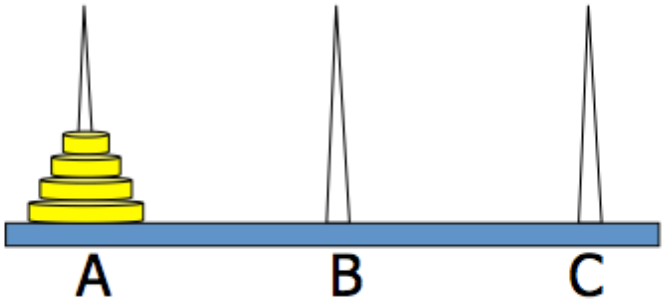
Figure 8.11: Three Disks



8.2.4.4 Four or more Disks

You know how to move three disks. Now, we move four disks, as shown in Figure 8.13.

Figure 8.12: Four Disks



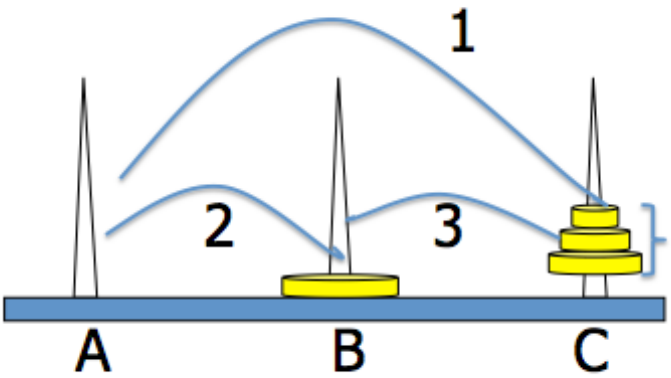
Here are the steps:

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 8.2.4.3.

Figure 8.13: 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

```
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");
    }
}
```

8.2.4.5 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)

8.2.5 Palindrome

Listing 12: Palindrome

```
1  public static boolean palindrome(String s){
2          if(s.length() == 1 || s.length() == 0){
3              return true;
4          }
5          if(s.charAt(0) != s.charAt(s.length()-1)) return false;
6          return palindrome(s.substring(1,s.length()-1));
7      }
```

8.2.6 Fibonacci

Listing 13: Fibonacci

```
1      //recursive implementation of Fibonacci. Extremely slow
2      public static int fib1(int n){
3          System.out.println("working hard " + count++ + " times");
4          if(n == 1) return 1;
5          if(n == 2) return 1;
6          return fib1(n-1)+ fib1(n-2);
7      }
8       //iterative implementation of Fibonacci. Linear time
9      public static int fib2(int n){
10         int a = 0;
11         int b = 1;
12         int t = 1;
13         for(int i =1; i < n; i++){
14             t = a + b;
15             a = b;
16             b = t;
17         }
18         return t;
19
```

```
20      }
21      //BigInteger example
22      public static BigInteger fib3(int n){
23          BigInteger a = new BigInteger("0");
24          BigInteger b = new BigInteger("1");
25          BigInteger t = new BigInteger("1");
26          for(int i =1; i < n; i++){
27              t = a.add(b);
28              a = b;
29              b = t;
30          }
31          return t;
32      }
33      //Test
34      public static void main(String[] args) {
35          int n = 10000;
36          BigInteger f = fib3(n);
37          System.out.println(f);
38      }
```

8.2.7 Recursive Tree

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

8.2.8 Maze

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