Tutorial 3

Analysis and Design of Algorithms

Divide and Conquer

Divide & Conquer

It is an **algorithmic paradigm** in which the problem is solved by following **3 main steps**:

Divide

Conquer

Combine

Divide & Conquer II

We depend on **recursion** to solve divide & conquer problems

Example: Write pseudocode for the factorial function

Direct solution for base case

Divide and Conquer for larger case

```
public static int factorial(int n) {
   if (n == 0) return 1;
   else return n * factorial(n-1);
   // post-condition: returns n!
}
```

Recurrence Relations

It is an **equation** in which a function is defined **in terms of itself**.

Example:
Fibonacci(n) = Fibonacci(n−1) + Fibonacci(n−2)

Recursive algorithms can be analyzed using recurrence relations because of that **self-reference property**

Divide & Conquer Recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ aT(b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

where a is the number of subproblems, b is the size of each subproblem in terms of n, $D(n)$ is the divide time, $C(n)$ is the combine time.

Exercise 3-1 From CLRS (©MIT Press 2001)

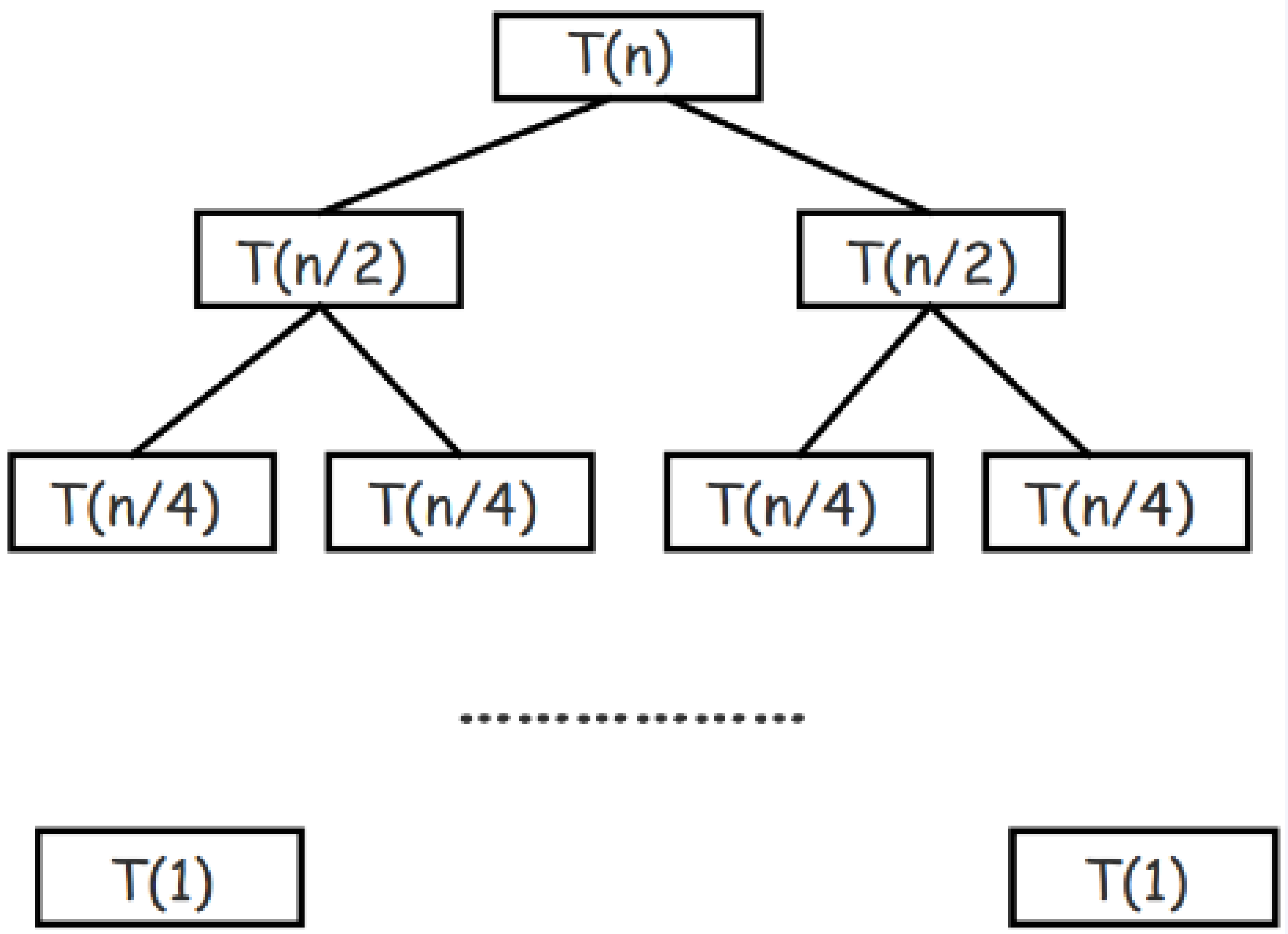
Insertion sort can be expressed as a recursive procedure as follows:
In order to sort $A[1..n]$, we recursively sort $A[1..n-1]$ then insert $A[n]$ into the sorted array $A[1..n-1]$.
Write a recurrence for the running time of this recursive version of insertion sort.

Solving Recurrences: Tree Method

Solving recurrences helps us obtain an upper bound on the running time of the algorithm
We will do this using a **Recursion Tree**

- **Nodes of the tree** represent **recursive calls**
- **Root of the tree** is **the initial call**
- **Leaves** correspond to **the exit condition**

$$T(n) = 2\,T\left(\frac{n}{2}\right) + n^2$$



Tree Method: Steps

1. Draw the recursion tree.
2. Figure out the height of the tree h.
3. Cost of the leaves $=$ number of leaves $\times$ c.
4. Figure out a formula representing the cost of each level (possibly in terms of the level number).
5. Cost of the rest of the tree $= \Sigma_{i=0}^{h-1}$ cost of each level.
6. Total running time $=$ cost of leaves $+$ cost of the rest of the tree.

Exercise 3-2 From CLRS (©MIT Press 2001)

Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 3T(\lfloor n/2 \rfloor) + n$.

Exercise 3-4 From CLRS (©MIT Press 2001)

Use the divide-and-conquer approach to write an algorithm that finds the largest item in a list of n items. Analyze your algorithm and get it's worst-case time complexity.

Exercise 3-4 From CLRS (©MIT Press 2001)

Use the divide-and-conquer approach to write an algorithm that finds the largest item in a list of n items. Analyze your algorithm and get it's worst-case time complexity.

```
static int largest( int low, int high ) {
    if ( low == high ) {
        // Subarray size is 1, solution is trivial
        return list[low];
    }
    else {
        int mid    = (low + high) / 2;
        int lLeft  = largest( low,   mid  );
        int lRight = largest( mid+1, high );

        if ( lLeft > lRight )
            return lLeft;
        else
            return lRight;
    }
}
```

Exercise 3-5

Write a divide-and-conquer algorithm for the **Towers of Hanoi** problem. The Towers of Hanoi problem consists of three pegs and n disks of different sizes. The objective is to move the disks that are stacked on one of the three pegs (in decreasing order of their size) to a new peg using the third one as a temporary peg. The problem should be solved according to the following rules:

i when a disk is moved, it must be placed on one of the three pegs;

ii only one disk may be moved at a time, and it must be the top disk on one of the pegs; and

iii a larger disk may never be placed on top of a smaller disk.

What is the worst-case time complexity of your algorithm?

```
public class Hanoi {
    enum Tower { A, B, C };

    static void moveDisk( Tower from, Tower to ) {
        System.out.println(
            "Move disk from " + from + " to " + to
        );
    }

    public static void towers( int n, Tower x, Tower y, Tower z ) {
        if ( n > 0 ) {
            towers( n-1, x, z, y );
            moveDisk( x, y );
            towers( n-1, z, y, x );
        }
    }
}
```

All done!