German University in Cairo
Department of Computer Science
Dr. Nourhan Ehab

CSEN 703 Analysis and Design of Algorithms, Winter Term 2022
Practice Assignment 3

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.

Solution:

Assuming that $T(n)$ denotes the worst case time complexity, it will take the algorithm $T(n-1)$ time to solve the recursive problem where $n > 1$ and $\Theta(n)$ to insert $A[n]$ into the sorted array

$$T(n) = \begin{cases} T(n-1) + \Theta(n) & \text{if } n > 1; \\ \Theta(1) & \text{if } n = 1. \end{cases}$$

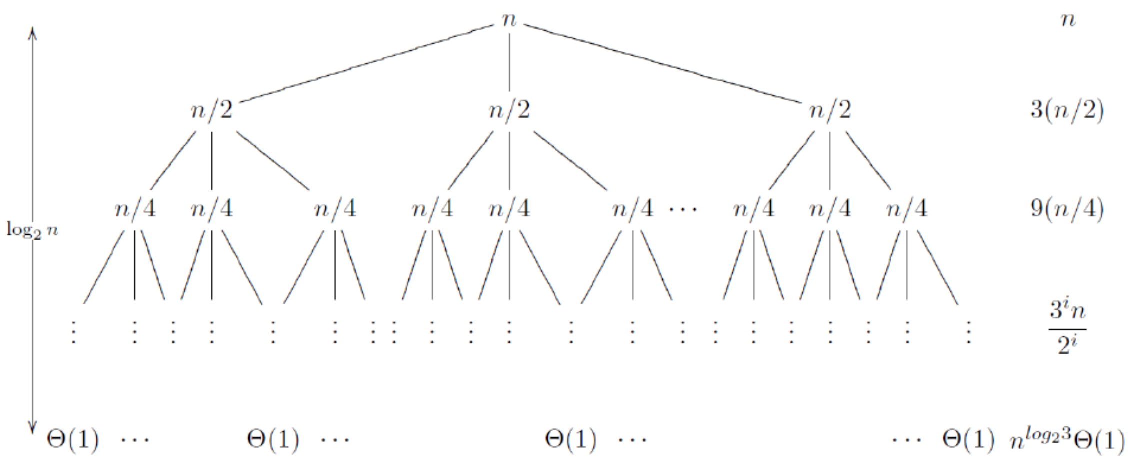
where T(n) = $\Theta(n^2)$

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

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

Solution:

Drawing the tree,



Calculating the cost from the recursion tree:
Cost of the leaves = $n^{\log_2 3}$.

Cost of all the levels except the leaves = $\sum_{i=0}^{\log_2(n)-1} n(\frac{3}{2})^i$.

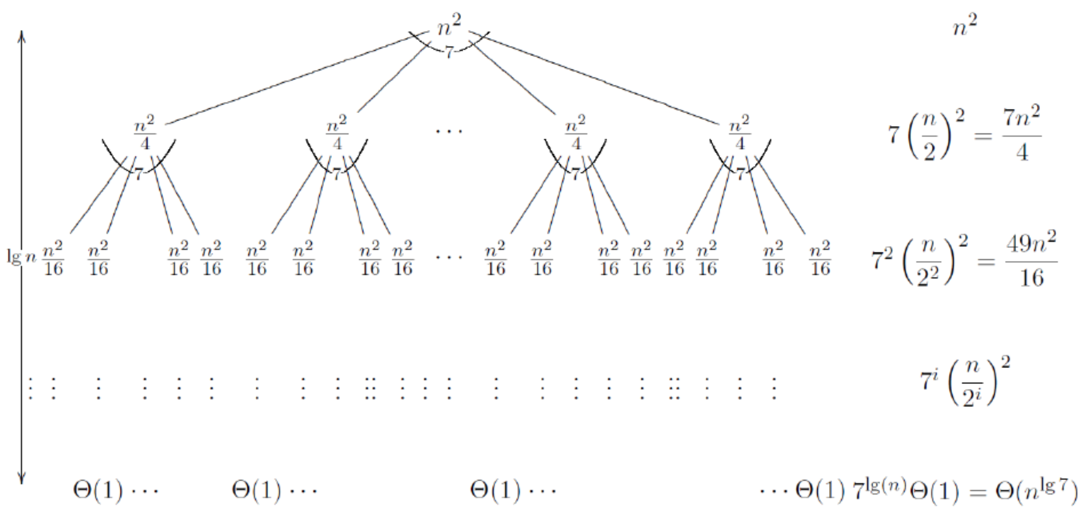
$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_2(n)-1} n(\frac{3}{2})^i + n^{\log_2 3} \\
&= n\left[\frac{1-1.5^{\log_2 n}}{1-1.5}\right] + n^{\log_2 3} \\
&= n\left[\frac{1.5^{\log_2 n}-1}{0.5}\right] + n^{\log_2 3} \\
&= 2n(1.5^{\log_2 n}) - 2n + n^{\log_2 3} \\
&= 2n(n^{\log_2 1.5}) - 2n + n^{\log_2 3} \\
&= 2n^{\log_2 2+\log_2 1.5} - 2n + n^{\log_2 3} \\
&= 2n^{\log_2 2(1.5)} - 2n + n^{\log_2 3} \\
&= 3n^{\log_2 3} - 2n
\end{aligned}
$$

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

Solve the following recurrence using the recursion tree method. $T(n) = 7T(n/2) + n^2$

Solution:

Drawing the recursion tree,



Calculating the cost from the recursion tree.
Cost of the leaves $= n^{\lg 7}$.

Cost of all the levels except the leaves $= \sum_{i=0}^{\log(n)-1} \left(\frac{7}{4}\right)^i n^2$

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log(n)-1} \frac{7^i n^2}{4^i} + n^{\lg 7} \\
&= n^2 \sum_{i=0}^{\log(n)-1} \frac{7^i}{4^i} + n^{\lg 7} \\
&= n^2 \sum_{i=0}^{\log(n)-1} 1.75^i + n^{\lg 7} \\
&= n^2 \tfrac{1-1.75^{\lg n}}{1-1.75} + n^{\lg 7} \\
&= n^2 \tfrac{1-n^{\lg 1.75}}{-0.75} + n^{\lg 7} \\
&= n^2 \tfrac{n^{\lg 1.75}-1}{0.75} + n^{\lg 7} \\
&= \tfrac{1}{0.75}\left(n^{2+\lg 1.75} - n^2\right) + n^{\lg 7} \\
&= \tfrac{1}{0.75}\left(n^{\lg 4+\lg 1.75} - n^2\right) + n^{\lg 7} \\
&= \tfrac{1}{0.75}\left(n^{\lg 4(1.75)} - n^2\right) + n^{\lg 7} \\
&= \tfrac{1}{0.75}\left(n^{\lg 7} - n^2\right) + n^{\lg 7} \\
&= (\tfrac{7}{3})n^{\lg 7} - \tfrac{1}{0.75}n^2
\end{aligned}
$$

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.

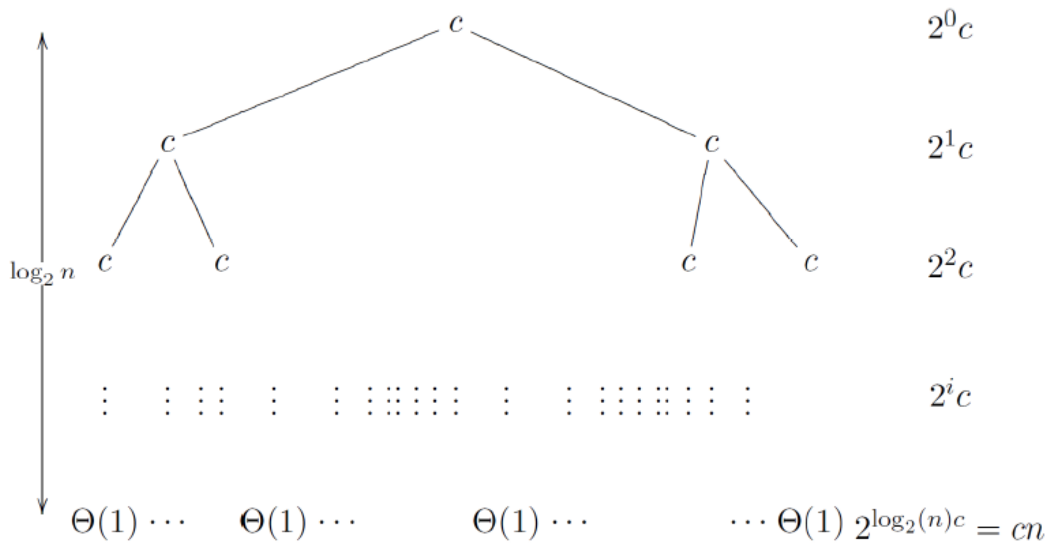
Solution:

```
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;
    }
}
```

We are assuming that `list` is a global array containing the list of elements. The function is initially invoked by calling `largest( 0, list.length-1 )`. The time complexity is given by the following recurrence relation:

$$T(n) = \begin{cases} 2\,T\left(\frac{n}{2}\right) + \Theta(1) & n > 1 \\ \Theta(1) & n = 1 \end{cases}$$

Drawing the recursion tree,



Calculating the cost from the recursion tree:
Cost of the leaves $= cn$.

Cost of all the levels except the leaves $= \displaystyle\sum_{i=0}^{\log_2(n)-1} 2^i c = cn - c$

$$T(n) \quad = \quad 2cn - c = O(n)$$

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 ) {
```

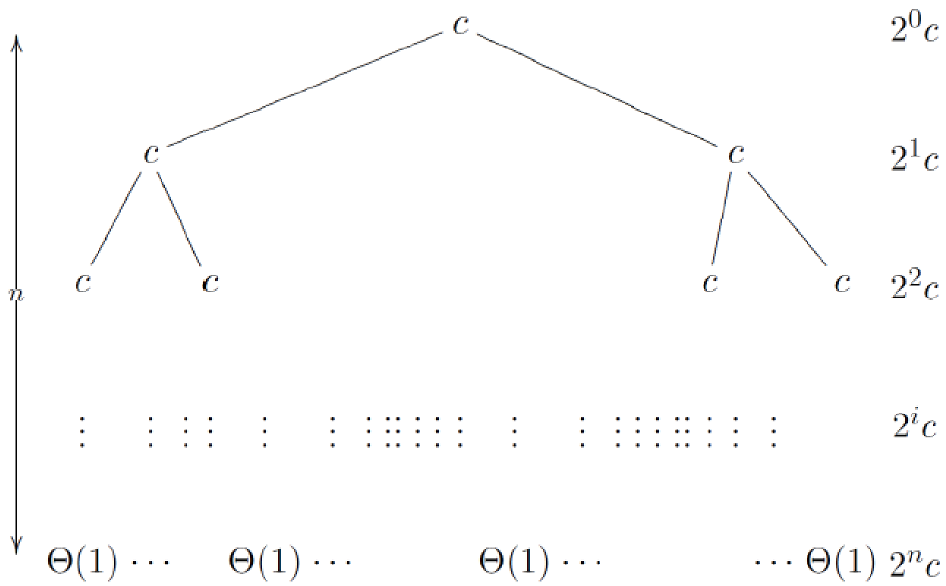
```
            System.out.println(
                "Move disk from " + from + " to " + to
            );
        }

        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 );
            }
        }
    }
```

Solution:

We initially invoke `Hanoi.towers( n, Tower.A, Tower.B, Tower.C )`. Looking at the recursive calls, it becomes clear that the recurrence relation describing the time complexity is given by:

$$T(n) = \begin{cases} 2\,T(n-1) + \Theta(1) & n > 0 \\ \Theta(1) & n = 0 \end{cases}$$

Drawing the recursion tree,



Calculating the cost from the recursion tree:
Cost of the leaves $= c2^n$.

Cost of all the levels except the leaves $= \sum_{i=0}^{n-1} c2^i = c2^n - c$

$$T(n) \;=\; c2^{n+1} - c = O(2^n)$$