2 Problem 2.12

In this problem we have to give an recurrence for the number of lines printed by the algorithm. The recurrence is given as follows

$$L(n) = \begin{cases} 1 + 2L(\frac{n}{2}) & \text{if n > 1} \\ 0 & \text{if n = 1} \end{cases} \tag{5}$$

Theorem 1 *L(n)* $= \Theta(n)$

Proof: Base Case: L(1) = 0 which is $\Theta(1)$
Hypothesis: c_1·k ≤ L(k) ≤ c_2·(k-1), k < n
Induction: L(n) = 1 + 2L($\frac{n}{2}$) ≥ 1 + 2($c_1 \cdot \frac{n}{2}$)=1+$c_1 n$=$k_1 n$ where k_1 is a constant equal to $c_1 + \frac{1}{n}$
Similarly for the other bound, L(n) = 1 + 2L($\frac{n}{2}$) ≤ 1 + 2($c_2 \cdot (\frac{n}{2} - 1)$) = 1+$c_2$n - 2$c_2$ = k_2(n-1) where $k_2 = c_2 - \frac{(c_2-1)}{(n-1)}$
□
Using above result we can say that L(n) is $\Theta(n)$. We can do a more accurate analysis using recursion tree and establish that the line will be printed n-1 times, which is still $\Theta(n)$.

3 Problem 2.14

Given an array of n elements, we need to remove the duplicate elements from the array in O($n \log n$) time. Idea is to maintain the order of elements in the array after the duplicates are removed. The following example explains the idea. Let the array A has following numbers: 2 3 1 3 1 4.
Once the duplicate numbers are removed the output array should be 2 3 1 4.
Note that the order of elements in the final output array is maintained. In other words the final array is not sorted.

```
function remove-duplicate(a[1...n])
Input:  An array of numbers a[1...n]
Output:  Array A with duplicates removed
Construct an array temp[1..n]:
  temp[i] has two fields key and value
for i = 1 to n
  temp[i].value = a[i]
  temp[i].key = i
sort temp based on value
remove duplicates from temp based on value:
  keep the entry with minimum key
sort temp based on key
construct array A from temp:
  A[i] = temp[i].value
return A
```