Disclaimer: There might be typo(s) or even error(s) in the following. If you spot one, please email me (guria@cs.nyu.edu) immediately. Some of the problems might have multiple solutions and only a subset of them will be presented here.

This assignment is about recursion and solving by divide and conquer. For all the following problems, give recursive solution (even if better iterative solution exists), prove correctness by induction, solve runtime using recurrence.

Q1. CLRS (Page 85): Problems 4-2 : Finding the missing integer

An array $A[1...n]$ contains all the integers from 0 to $n$ except one. It would be easy to determine the missing integer in $O(n)$ time by using an auxiliary array $B[0...n]$ to record which numbers appear in $A$. In this problem, however, we cannot access an entire integer in $A$ with a single operation. The elements of $A$ are represented in binary, and the only operation we can use to access them is "fetch the $j$th bit of $A[i]"$, which takes constant time.

Show that if we use only this operation, we can still determine the missing integer in $O(n)$ time.

SOLUTION:

Our convention is to call the least significant bit (LSB) as 0-th bit. Number of digits required to represent $n$ is at least $\lceil \lg(n + 1) \rceil$. Most significant bit (MSB) of $n$ is the $(\lceil \lg(n + 1) \rceil - 1)$-th bit of it.

Among the integers between 0 to $n$, exactly $2^{(\lceil \lg(n+1) \rceil - 1)}$ numbers are $< 2^{(\lceil \lg(n+1) \rceil - 1)}$. If we look at the $(\lceil \lg(n + 1) \rceil - 1)$-th bit of any non-negative number $\leq n$ we can determine whether the number is $< 2^{(\lceil \lg(n+1) \rceil - 1)}$. We now look at the $(\lceil \lg(n + 1) \rceil - 1)$-th bit of all the numbers in $A[1...n]$. We populate two lists of references (e.g. pointer, index, handle etc) during the first $n$ lookup. Reference to the numbers having 0 in their $(\lceil \lg(n + 1) \rceil - 1)$-th bit are kept in one and the rest in the other. The size of the first list is the count of numbers $< 2^{(\lceil \lg(n+1) \rceil - 1)}$. The missing integer is $< 2^{(\lceil \lg(n+1) \rceil - 1)}$ iff this count is $2^{(\lceil \lg(n+1) \rceil - 1)} - 1$. We look for the missing integer in the first list in that case. Otherwise, we look in the second list. In this process, we figure out the $(\lceil \lg(n + 1) \rceil - 1)$-th bit of the missing integer in $n$ bit lookup.

We want to use recursion to search the appropriate list. The fact that now we search in a list of references to numbers in stead of a list of numbers can be handled as follows. The initial problem comes with the list all the references. The fact that the search range could be shifted in the subproblem is handled by replacing all the $(\lceil \lg(n + 1) \rceil - 1)$-th bits as 0. However, as we never read the $(\lceil \lg(n + 1) \rceil - 1)$-th bits again we don’t have to do this modification explicitly.
find_missing_integer ( list_of_references, n )
  if ( n is 1 )
    print 0-th bit of missing number is (1 - the number in the list)
    return
  left_list . initialize ()
  right_list . initialize ()
  msb_position <- ceiling( lg(n+1) ) - 1
  forall number in list_of_references
    if ( number [ msb_position ] is 0 )
      left_list . insert ( number )
    else
      right_list . insert ( number )
  if ( left_list . size () < power_of_two ( msb_position ) )
    print msb_position-th bit of missing number is 0
    find_missing_integer ( left_list, power_of_two ( msb_position ) - 1 )
  else
    print msb_position-th bit of missing number is 1
    find_missing_integer ( right_list, n - power_of_two ( msb_position ) )

Proof of correctness
Correctness follows from induction on \( n \).

**Base case**: \( n = 1 \). We are given one element from \{0, 1\}. Our algorithm correctly handles this case.

**Induction step**: Suppose our algorithm works for \( n = 1 \) to \( m \). We have to show that it works for \( n = m + 1 \). We have show that length of the sub-problem we solve is \( \leq m \).

\[
2^{(\lceil \lg(n+1) \rceil) - 1} \leq n \leq 2^{\lceil \lg(n+1) \rceil} - 1
\]

Length of the left_list is \( 2^{(\lceil \lg(n+1) \rceil) - 1} - 1 < n \). Length of the right_list is \( n - 2^{(\lceil \lg(n+1) \rceil) - 1} \leq 2^{\lceil \lg(n+1) \rceil} - 1 - 2^{(\lceil \lg(n+1) \rceil) - 1} = 2^{\lceil \lg(n+1) \rceil} - 1 - 1 < n \). We have shown previously that we correctly decide the Most significant bit of the missing number and the list in which to look for its remaining part.

**Runtime using recurrence**
Let \( T(n) \) be some upperbound on the runtime for \( n \). We perform at most \( cn \) work to divide the problem and combine the solution for some \( c > 0 \). We solve only one subproblem whose size is at most \( 2^{(\lceil \lg(n+1) \rceil) - 1} - 1 \).

Assume \( T(n) \) to be a non-decreasing function.

\[
T(n) \leq T(2^{(\lceil \lg(n+1) \rceil) - 1}) + cn
\]

Let \( k = \lfloor \lg(n+1) \rfloor \).

\[
T(n) \leq T(2^k - 1)
\]
\[
\leq T(2^{k-1} - 1) + c(2^k - 1)
\]
\[
\leq T(2^{k-2} - 1) + c((2^{k-1} - 1) + (2^k - 1))
\]
\[
\leq T(2^1 - 1) + c \left( \sum_{i=2}^{k} (2^i - 1) \right)
\]
\[
= T(1) + c(2^{k+1} - 2 - (k - 1))
\]
\[
= T(1) + c(2^{k+1} - 2)
\]
\[
= T(1) + 2c(2^k - 1)
\]
\[
\leq T(1) + 2c(n)
\]
So $T(n)$ is $O(n)$.

Q2. Tiling with L-shaped tiles:
We are given a square chess-like board of size $2^k$ by $2^k$. One of $2^{2k}$ unit-squares in the board is broken. We are required to cover the remaining part by L-shaped tiles of size 3. An L-shaped tile is obtained by removing any of the 4 unit-squares from a 2 by 2 square tile.

Try to reduce this problem to 4 sub-problems. The division into subproblem is similar to the division of a square matrix in 4 parts (as we saw in Strassen’s algorithm).

Q3. Maximum Subsequence sum:
We have an array (or sequence) of integers (not necessarily positive). We are required to find a contiguous sub-array (or subsequence) for which the sum of elements in it are maximized.

Try to reduce this problem to 2 sub-problems (like in merge-sort). The subsequence you are looking for are either fully contained in the sub-problems or is made of some suffix of one part and some prefix of the other part.

Q4. CLRS (Page 87): Problems 4-6: VLSI chip testing
If you did the parts (b) and (c) correctly in the last assignment, you don’t have to do this. Otherwise, you can resubmit your solution.