Comprehensive Exam Review Problems: Divide and Conquer Algorithms

1a. Show how to multiply two linear polynomials $ax+b$ and $cx+d$ using only three multiplications. (HINT: One of the multiplications is $(a+b)(c+d)$.)

1b. Assume that there exists a divide-and-conquer algorithm for multiplying two polynomials of degree n that runs in time $\Theta(n^{\log_2 3})$. Show that two n-bit integers can be multiplied in $O(n^{\log_2 3})$ time, where each step operates on at most a constant number of 1-bit values.

2a. You are given $n > 1$ coins. $n-1$ of the coins weigh the same, while one other coin weighs slightly less than the others. You have a scale and can put a single coin on either side of the scale for each weighing. Give an algorithm to find the light coin in no more than $\lfloor \frac{n}{2} \rfloor$ weighings.

2b. Prove that $\lfloor \frac{n}{2} \rfloor$ is necessary for finding the light coin.

3. The Maximum Subsequence Sum problem takes as input an array of integers $a_0, a_1, \ldots, a_{n-1}$, and returns the largest sum of any subsequence of the form $a_j + a_{j+1} + \cdots + a_{j+k}$, where $0 \le k \le n-1$ and $k+j \le n-1$. Determine a divide and conquer algorithm for this problem and prove that the algorithm runs in $O(n \log n)$ steps.

4. You are developing a divide-and-conquer algorithm which must have asymptotic complexity $O(n \log n)$ to be of practical use. Moreover, you have decided to divide the problem into 3 subproblems of size $n/4$ each, where n is the size of the original problem. Using this strategy is it possible to achieve the desired complexity? If so, what is the most number of steps (as a function of n) that may be used for dividing up the original problem and combining the solutions of the subproblems? Explain.

5. Suppose array $A[1:n]$ contains all integers from 0 to n except one. You want to determine which integer is missing, but you may not access any of the array elements using a single operation. Instead, the elements of A are written in binary, and you only have access to the function $B(i,j)$ which returns the j th bit of element $A[i]$, $1 \le i \le n$ and $j \ge 0$. Using this function, describe in one or more paragraphs an algorithm that finds the missing integer in O(n) steps. Prove that your algorithm runs in linear time.

6. Given n integers, we know that $n-1$ comparisons are needed in the worst case to find the least of the integers. Describe an algorithm that uses comparisons so that the second least of n integers can always be found using no more than $n + \lceil \log n \rceil - 2$ comparisons. Hint: also find the least integer. Demonstrate your algorithm on the permuation $1, 5, 6, 4, 2, 8, 3, 7$.

The Fake-Coin Problem: Decrease by a Constant Factor

- Problem:
  - Among n identical looking coins, one is a fake (and weighs less)
  - We have a balance scale which can compare any two sets of coins
- Algorithm:
  - Divide into two size $\lfloor n/2 \rfloor$ piles (keeping a coin aside if n is odd)
  - If they weigh the same then the extra coin is fake
  - Otherwise proceed recursively with the lighter pile
- Efficiency:
  - $W(n) = W(\lfloor n/2 \rfloor) + 1$ for $n > 1$
  - $W(n) = \lfloor \log_2 n \rfloor = \Theta(\log_2 n)$
- But there is a better $(\log_3 n)$ algorithm

```
def main(n)
    if( n>=2 )
        if( n%2 == 1 )
            tmp_coin = rand(1,n)
            n = n-tmp_coin
        end
        if( n%2 == 0)
            fake_coin = find_fake_coin(n)
        end
        if( fake_coin == "Not_Find" )
            fake_coin = tmp_coin
        end
        return fake_coin
    else
        return "You must have at least 2 coins!"
    end
end

def find_fake_coin(n)
    if( n==1 )
        return the last reminder coin
    end
    a = weight( n/2 )
    b = weight( n/2 )
    if (a==b)
        return "Not_Find"
    else
        return find_fake_coin( min( a,b ) )
    end
end
```