Problem 1. (30 points)

Give a tight bound (using the big-theta notation) on the time complexity of following method as a function of $n$. For simplicity, you can assume $n$ to be a power of two.

Algorithm WeirdLoop ($n : \text{integer}$)

$i \leftarrow n$

while $i \geq 1$
    for $j \leftarrow 1$ to $i$
        $k \leftarrow 1$
        while $k \leq n$
            $k \leftarrow 2k$
        $i \leftarrow i/2$

Answer: First note that the innermost loop (while $k$) takes always $O(\log n)$, since it does not depend on $i$ and $j$. When $i = n$, the for $j$ loop is executed $n$ times, where each iteration costs $\log n$, for a total of $n \log n$. When $i = n/2$, the for $j$ loop is executed $n/2$ times, where each iteration costs $\log n$-time, for a total of $(n/2) \log n$. . . . When $i = 1$, the for $j$ loop is executed once, for a total of $\log n$ time. In summary, the total complexity is $(n + n/2 + n/4 . . . + 2 + 1) \log n \leq 2n \log n \in \Theta(n \log n)$.

Problem 2. (40 points)

You are given an array $A$ of size $n$ which contains integers in the range $[0, n - 1]$. Give a $O(n)$-time algorithm to print all integers that appear more than once in $A$, using only $O(1)$ additional memory space. Note that since you can use only constant additional space, the use of external data structures or hash tables is not allowed.

Answer: This is common interview question, and I have seen several solutions to this problem. An elegant solution by Caf on stackoverflow.com (assumes arrays starting at index 0, like in C/C++) is the following

Algorithm PrintDuplicates ($A : \text{array}$)

for $i \leftarrow 0$ to $n - 1$
    while $A[A[i]] \neq A[i]$
        swap ($A[i], A[A[i]]$)
    for $i \leftarrow 0$ to $n - 1$
        if $A[i] \neq i$ then
            print $A[i]$

The first loop rearranges the array so that if element $x$ is present at least once, then one of those occurrences will be stored at position $A[x]$. The first loop runs in linear time because a swap occurs only if there is an $i$ such that $A[i] \neq i$, and each swap sets at least
one element such that $A[i] = i$, which wasn’t true before. This means that the total number of swaps (and thus the total number of executions of the while loop body) is linear.

The second loop prints the values of $x$ for which $A[x] \neq x$ since the first loop guarantees that if $x$ exists at least once in the array, one of those instances will be at $A[x]$, this means that the second for loop prints the repetitive integers (multiple times). The algorithm can be easily fixed to print each repetitive integer only once.

**Problem 3. (30 points)**

Given the following recurrence relation

$$T(n) = \begin{cases} 
1 & n = 1 \\
T\left(\frac{n}{4}\right) + \sqrt{n} & n > 1 
\end{cases}$$

1. Solve it exactly (i.e., without using any asymptotic notation) by iterative substitutions

2. Prove by induction that your solution is correct

**Answer:** We have

$$T(n) = T(n/4) + \sqrt{n}$$

$$= T(n/4^2) + \sqrt{n}(1/2 + 1)$$

$$= T(n/4^3) + \sqrt{n}(1/4 + 1/2 + 1)$$

$$\cdots$$

$$= T(n/4^i) + \sqrt{n}\left(1/2^{i-1} + 1/2^{i-2} + \ldots + 1/2^1 + 1/2^0\right)$$

$$= T(n/4^i) + 2\sqrt{n}\left(1 - 1/2^i\right)$$

now we set $n/4^i = 1$ which is $i = \log_4 n$ and we get

$$T(n) = T(1) + 2\sqrt{n}\left(1 - 1/\log_4 n\right)$$

$$= 1 + 2\sqrt{n}\left(1 - 1/\sqrt{n}\right)$$

$$= 2\sqrt{n} - 1$$

We now prove by induction that $T(n) = 2\sqrt{n} - 1$ is the correct solution of the recurrence relation.

Base case ($n = 1$). $T(1) = 2\sqrt{1} - 1 = 1$, which matches the base case value for $T$.

Induction step. Assume the statement is true for $n/2$, that is

$$T(n/4) = 2\sqrt{n/4} - 1 = \sqrt{n} - 1$$

We have

$$T(n) = T(n/4) + \sqrt{n}$$

$$= (\sqrt{n} - 1) + \sqrt{n}$$

$$= \sqrt{n} - 1$$