Q1. [15 marks] Answer the following questions with proper justification (no marks will be awarded if the reasoning is incorrect.)

a. What is the running time complexity of the following code? You can use order notation. Justify your answer.

```c
sum = 0;
for (k=1; k<n; k++)
    for (i=1; i<k*k; i++)
        if (i % k == 0)
            for (j=0; j<i; j++)
                sum++;
```

```latex
\begin{align*}
\sum_{k=1}^{n} \left[0 + k + 2k + \ldots + k \cdot k\right] \\
= \sum_{k=1}^{n} k \left[1 + 2 + \ldots + k\right] \\
= \sum_{k=1}^{n} k^2 \left(\frac{k(k+1)}{2}\right) = O(n^4).
\end{align*}
```

b. Is it true that $\log 1 + \log 2 + \ldots + \log n$ is $\Theta(n \log n)$? Explain your answer.

Yes. Two parts:

1. $\log 1 + \ldots + \log n \leq \log n + \ldots + \log n = n \log n \Rightarrow$ LHS is $O(n \log n)$.

2. $\log 1 + \ldots + \log n \geq \log \frac{n}{2} + \ldots + \log n = \log \frac{n}{2} + \ldots + \log \frac{n}{2} = \frac{n}{2} \log \frac{n}{2} = \Omega(n \log n)$.

Thus, LHS is both $O(n \log n)$ and $\Omega(n \log n) \Rightarrow \Theta(n \log n)$. 
Q2. [10 marks] Starting from an empty AVL tree, the following elements are inserted (in this order) in the AVL tree: 11,17,20,9,7,5,15,13. You should use the rotation/balancing procedures described in the class. Show the AVL tree after each insert operation. After inserting the above numbers in the AVL tree, delete the root.

Delete 9

i. Delete successor node of 9:

OR

ii. Delete predecessor node:

Delete the root.
Q3. [15 marks] Given a node A in a binary search tree, write an efficient procedure to find its inorder predecessor. Fill in the details of the procedure below. Do not declare any other function.

Node* Predecessor (Node* A)
{
    if (A->right is not a leaf)  
        p = A->right;
        while (p->left is not a leaf)
            p = p->left;
        return p;
    else
        p = A;
        q = A->parent;
        while (p is the right child of q)  
            p = q;
            q = p->parent;  
        if q becomes null,  
            A was the largest node.
    return q;
}

Q4. [20 marks] Given a stack, and a sequence of distinct numbers in an array A = [a_1, .., a_n] you can perform the following operations starting from the first number in A: either push the next number in the sequence on the stack, or pop from the stack.
Given a permutation B = [b_1, .., b_n] of A = [a_1, ..., a_n], we would like to know if it can be obtained from a_1...a_n by such operations.

For example, let A = [1 2 3 4 5] and B = [2 3 5 4 1]. Then B can be obtained from A by the following operations:

(i) Let A = [1 2 3 4 5], B = [4 3 1 5 2]. Can B be obtained from A? Why?
No. First we need to push 1, 2, 3, 4 and then pop, pop, Now, I cannot be printed before 2.
(ii) Write a linear time procedure which given A and B of length $n$ each, outputs whether B can be obtained from A using the method above. If yes, the procedure should output the sequence of operations. Note that you can access the stack using `push`, `pop`, `is_empty` and `top` functions only. You should not modify the arrays A and B, and should use only a constant amount of extra space (besides using the stack and reading from the two arrays A and B).

\[
\begin{align*}
i &= 0 \quad (i \text{ scans A}) \\
j &= 0 \quad (j \text{ scans B}) \\
\text{while } (j < n) \{ & \\
\quad \text{if } (\text{stack } S \text{ is not empty and } S.\text{top}() == B[j]) \{ & \\
\quad \quad S.\text{pop}(); & \\
\quad \quad j++; & \\
\quad \} & \\
\quad \text{else } \{ & \\
\quad \quad S.\text{push}(A[i]); & \\
\quad \quad i++; \quad \text{if } i \text{ exceeds } n, \text{ "ERROR"} & \\
\quad \} & \\
\} & \text{If quadratic time algorithm, max. 5 marks.} \\
\text{Worse than quadratic time: NO MARKS.} &
\end{align*}
\]