Homework I

Question 1

```
Input: integer n
Output: summation sum(n)
function sum(n)
    if n is 1
        return 1
    else
        return n*(n+1)/2 + sum(n-1)
    end if
end function
```

Question 2

```
Input: A[1..N] (array of size N)
Output: largest and second largest numbers
function maximum_two(A, 1, N)
    if N is 2
        if A[1] is greater than A[2]
            return (A[1], A[2])
        else
            return (A[2], A[1])
        end if
    else
        merged <- maximum_two(A, 1, N/2) union maximum_two(A, N/2+1, N)
        if length of merged is 4
            merged <- maximum_two_from_four(merged)
        end if
        return merged
    end if
end function

Input: B (array of size 4)
Output: largest and second largest numbers
function maximum_two_from_four(B)
    if B[1] > B[3]
        largest <- B[1]
        if B[2] > B[3]
            second_largest <- B[2]
        else
            second_largest <- B[3]
```

```
        end if
    else
        largest <- B[3]
        if B[1] > B[4]
            second_largest <- B[1]
        else
            second_largest <- B[4]
        end if
    end if
    return largest, second_largest
end function
```

Question 3

1. Using an algorithm like Bubble sort to sort the elements, and further returning the largest and second largest elements from the sorted array, the worst case complexity of the problem would turn out to be $O(n^2)$, which would be an upper bound for this approach.

2. Using an algorithm like Merge sort to sort the elements, and further returning the largest and second largest elements from the sorted array, the worst case complexity of the problem would turn out to be $O(nlog(n))$, which would be a non-trivial upper bound for this approach.

3. The efficient way to solve the problem would be to find the largest number from an array of N, and further find the largest of all the numbers that the original largest number has been compared to. In the current scheme, finding the largest number in an array of N is atleast $(N-1)$ comparisons. Furthermore, the largest would be compared to log_2N numbers. To find the largest from log_2N numbers, a minimum of $(log_2N - 1)$ comparisons would be required. Thus, the minimum number of comparisons to find the largest and second largest numbers would be $(N-1) + (log_2N - 1)$ comparisons, i.e, a minimum of $(log_2N + N - 2)$ comparisons, which would be the lower bound.

4.

Question 4

1. $100n^2 = O(n^2 - 10n)$
   Proof:

   $100n^2 \leq c(n^2 - 10n)$, for some c > 0

$\implies \frac{100n^2}{n^2-10n} \leq c$

$\implies c \geq \frac{100n^2}{n^2-10n}$

$\implies c \geq \frac{100}{1-\frac{10}{n}}$

This is true for all n > 10
$\therefore$ Hence Proved

2. $n^k = O(2^{n)}$, for any fixed $k \geq 1$
Proof:

$n^k \leq c.2^n$

Both n^k and 2^n are positive, and c > 0.
$\implies k.log(n) \leq log(c.2^n)$

$\implies k.log(n) \leq log(c) + log(2^n)$

$\implies k.log(n) \leq log(c) + nlog(2)$

$\implies k \leq \frac{log(c)}{log(n)} + \frac{n}{log(n)}$

We know that, $log(n) \leq n$. So, $\frac{n}{log(n)} \geq 1$.

$\implies k \leq \frac{log(c)}{log(n)} + 1$

For all $c > 0$ and $n > 1$, k is always positive.
This is true for all n > 1.
$\therefore$ Hence Proved

Question 5

(c) $f(n) = O(g(n))$ implies $lg(f(n)) = O(lg(g(n)))$, where $lg(g(n)) \geq 1$ and $f(n) \geq 1$ for all sufficiently large n.
Proof:

$f(n) = O(g(n))$
$\implies f(n) \leq c.g(n)$

$f(n) \geq 1$ for all n; $f(n)$ and $g(n)$ are asymptotically positive functions

$\implies lg(f(n)) \leq lg(c.g(n))$
$\implies lg(f(n)) \leq lg(c) + lg(g(n))$
Assume $lg(c) + lg(g(n)) = p.lg(g(n))$

$\implies p = \frac{lg(c)}{lg(g(n))} + 1$

$\implies p \leq lg(c) + 1$
It can be seen that $p \geq 1$ as $c > 0$. Thus, for $p \leq lg(c) + 1$, $lg(f(n)) \leq p.lg(g(n))$ such that $p \geq 1$
Therefore, the given statement is true.

(e) $f(n) = O((f(n))^2)$
Proof:

$f(n) \leq c.(f(n))^2$, for some c > 0 (By definition)

$\implies \frac{f(n)}{(f(n))^2} \leq c$

$f(n)$ is an asymptotically positive function.

$\implies \frac{1}{f(n)} \leq c$

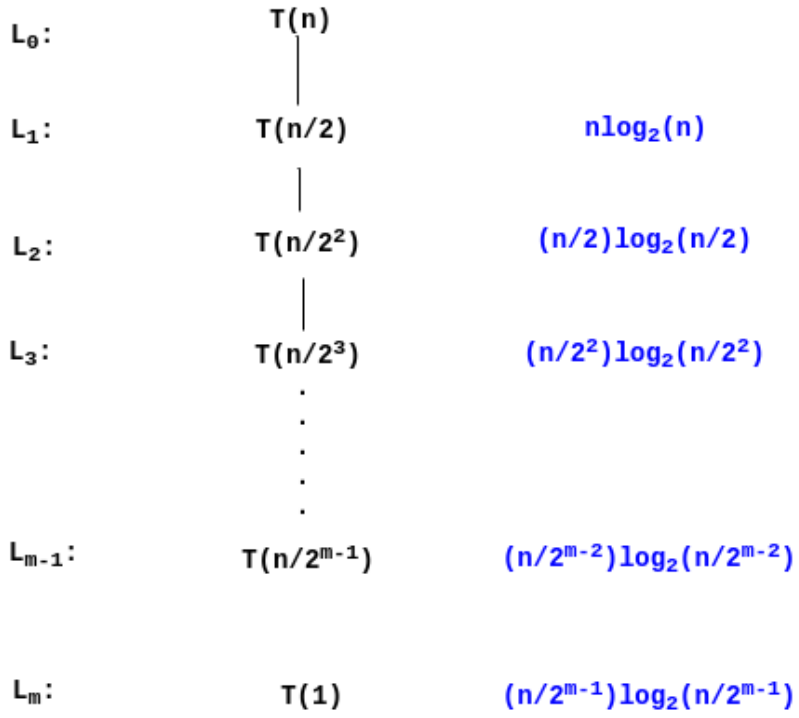
$\implies c \geq \frac{1}{f(n)}$

$\implies c \geq \frac{1}{f(n)} > 0$

This is true for all f(n).
$\therefore$ The given statement is true.

Question 6

$T(n) = T(n/2) + nlog_2 n$
Also, $T(1) = c$ for some constant c > 0



where $\frac{n}{2^m} = 1$, i.e, $m = log_2(n)$. Then, T(n) is the sum.

$T(n) = T(1) + [nlog_2 n + \frac{n}{2}log_2(\frac{n}{2}) + ... + \frac{n}{2^{m-1}}log_2(\frac{n}{2^{m-1}})]$

$\implies T(n) = T(1) + \sum_{k=0}^{m-1} \frac{n}{2^k} log_2(\frac{n}{2^k})$

$\implies T(n) = c + \sum_{k=0}^{m-1} \frac{n}{2^k}(log_2 n - log_2(2^k))$

$\implies T(n) = c + \sum_{k=0}^{m-1} \frac{n}{2^k}(log_2 n - k)$

$\implies T(n) = c - \sum_{k=0}^{m-1} \frac{n}{2^k}(k) + \sum_{k=0}^{m-1} \frac{n}{2^k}(log_2 n)$

$\implies T(n) \leq \sum_{k=0}^{m-1} \frac{n}{2^k}(log_2 n)$

$\implies T(n) \leq nlog_2 n. \sum_{k=0}^{m-1} \frac{1}{2^k}$

$\implies T(n) \leq nlog_2 n.[\frac{1-\frac{1}{2}^m}{1-\frac{1}{2}}]$

$\implies T(n) \leq nlog_2 n.[\frac{1}{1-\frac{1}{2}}]$

$\implies T(n) \leq c.nlog_2n$

$\implies T(n) = O(nlog_2n)$

Therefore, a good asymptotic upper bound on $T(n)$ is $nlog_2n$.

Question 7

$T(n) = T(\frac{n}{2}) + nlog_2n$ with base case T(1) = c

To Prove: $T(n) = O(nlog_2n)$
Proof:

We claim that there are constants a, k such that $T(n) \leq a.n.log_2n$ when $n \geq k$
Step 1: Base Case: When $n = 1$, $T(1) = c$, to make $T(1) \leq a.n.log_2n$, we need $n \geq k = 2$, so, we need to use T(2) as base case.
From recurrence,
$T(2) = T(1) + 2.log_22$
$\implies T(2) = T(1) + 2 \implies T(2) = c + 2$
Thus, when we choose $a = c + 2$, i.e, $a > 2$

Step 2: Assumption: For $\frac{n}{2}$, the claim is true, i.e,
$T(\frac{n}{2}) \leq a.\frac{n}{2}.log_2\frac{n}{2}$, when $n \geq 4$

Step 3: Induction: $T(n) = T(\frac{n}{2}) + nlog_2n$
$\implies T(n) \leq a.\frac{n}{2}.log_2\frac{n}{2} + nlog_2n$
$\implies T(n) \leq a.\frac{n}{2}(log_2n - 1) + nlog_2n$
$\implies T(n) \leq a.\frac{n}{2}(log_2n - a.\frac{n}{2} + nlog_2n$
$\implies T(n) \leq a.\frac{n}{2}log_2n - a.\frac{n}{2} + nlog_2n$
$\implies T(n) \leq (\frac{a}{2} + 1)nlog_2n - a.\frac{n}{2}$
$\implies T(n) \leq (\frac{a}{2} + 1)nlog_2n$
But we already know $a > 2$ from the base case.
Thus, this is true for all $n > 0$ when $c > 2$.
Therefore, $T(n) = O(nlog_2n)$

Hence Proved.