1. (30 marks) For each of the algorithms below, give a tight asymptotic worst-case, \textit{i.e.}, Big-Oh, time complexity function \(O(f(n))\). Briefly explain the reasoning behind each derivation.

(a) (5 marks)

\begin{verbatim}
sum = 1
for i = 1 to n do
    sum = sum / i
    k = i * i
    for j = 1 to n do
        sum = sum * (k - 13)
    for j = 1 to n do
        sum = sum * (k - 13)
    sum = sum * 14 * 1000000
for i = 1 to n * n do
    sum = sum + i
sum = sum - 42
\end{verbatim}

\textbf{Answer:} The two for-loops embedded inside the first for-loop execute \(n\) times apiece, and as this first for-loop executes \(n\) times, the first for-loop runs in \(O(n^2)\) time. The second for-loop executes \(n^2\) times and hence runs in \(O(n^2)\) time as well. Hence, the algorithm as a whole runs in \(O(n^2 + n^2) = O(n^2)\) time.

(b) (5 marks)

\begin{verbatim}
sum = 13
cond = false
for i = 1 to n do
    for j = 1 to 7 do
        sum = sum / (i + j)
    if COND(sum)
        cond = true
    else
        for j = 1 to n do
            sum = sum - j
    if cond
        for j = 1 to n * log(n) do
            sum = sum + (i/j)
\end{verbatim}

Note that method \texttt{COND()} runs in 5 timesteps and method \texttt{log(n)} returns the logarithm (base 2) of \(n\), \textit{i.e.}, \(\log_2 n\).
Answer: The for-loop immediately inside the first for-loop always executes, and executes 7 times; however, in the worst case (when COND(sum) == True, the for-loop in the else-clause executes n times, which means that (as the first for-loop executes n times), the first for-loop in the worst case runs in $O(n^2)$ time. If cond == True after the first for-loop finishes, the for-loop inside the associated if executes $n \log_2 n$ times, meaning that in the worst case this if-statement runs in $O(n \log_2 n)$ time. Hence, the algorithms as a whole runs in $O(n^2 + n \log_2 n) = O(2n^2) = O(n^2)$ time.

(c) (5 marks)

```plaintext
sum = 157
cond = false
for i = 1 to 7 do
    for j = 1 to n do
        sum = sum * (i/j)
        if (sum > 23)
            sum = sum + 23
            k = sum
            sum = sum + k
        else
            k = sum - 23
            for k = 1 to log(n) * n do
                sum = sum - (k/j)
        if (cond)
            sum = sum - 256
```

Answer: The deepest embedding of loop control structures is three for-loops which execute (going from innermost to outermost) $n \log_2 n$, $n$, and 7 times, respectively. Hence, the algorithm runs in $O(n \log_2 n \times n \times 7) = O(n^2 \log_2 n)$ time.

(d) (5 marks)

```plaintext
sum = 42
for i = 1 to n * n do
    j = 1
    finished = true
    while ((j <= n) and (not finished)) do
        for k = 1 to log(n) * n do
            sum = sum / (k * i) + j
        if COND(sum)
            finished = true
```

Answer: Note that the while-loop never executes, as variable finished is always true when the while-loop condition is first evaluated. Hence, the deepest embedding of
loop control structures is one for-loop which execute $n^2$ times and the algorithm runs in $O(n^2)$ time.

(e) (5 marks)

\[
\begin{align*}
\text{sum} &= 42 \\
\text{for } i = 1 \text{ to } n \times n \text{ do} \\
& \quad j = 1 \\
& \quad \text{finished} = \text{false} \\
& \quad \text{while } ((i \leq n) \text{ and } (\text{not finished})) \text{ do} \\
& \quad \quad \text{for } k = 1 \text{ to } \log(n) \times n \text{ do} \\
& \quad \quad \quad \text{finished} = \text{true} \\
& \quad \quad \text{if COND(}\text{sum}\text{)} \\
& \quad \quad \quad \text{sum} = \text{sum} / (k \times i) + j
\end{align*}
\]

Note that method COND() runs in $(n + 13)$ timesteps.

**Answer:** Note that the while-loop executes exactly once for each iteration of the outermost for-loop as variable finished is set to true during that first iteration of the while-loop and hence causes the while-loop condition to evaluate to false on the second iteration. As that single execution of the while-loop takes $O(n \log_2 n + (n + 13)) = O(n \log_2 n)$ time and the outermost for-loop executes $n^2$ times, the algorithm runs in $O(n^2 \times n \log_2 n) = O(n^3 \log_2 n)$ time.

(f) (5 marks)

\[
\begin{align*}
\text{sum} &= 42 \\
\text{for } i = 1 \text{ to } n \times \log(n) \text{ do} \\
& \quad j = 1 \\
& \quad \text{finished} = \text{false} \\
& \quad \text{for } k = 1 \text{ to } n \text{ do} \\
& \quad \quad \text{if COND(}\text{sum}\text{)} \\
& \quad \quad \quad \text{sum} = \text{sum} / (k \times i) + j \\
& \quad \quad \text{while } ((j \leq n) \text{ and } (\text{not finished})) \text{ do} \\
& \quad \quad \quad \text{finished} = \text{true}
\end{align*}
\]

Note that method COND() runs in $(n + 13)$ timesteps.

**Answer:** Note that the while-loop executes exactly once for each iteration of the innermost for-loop as the variable finished is set to true during the first loop iteration and hence causes the while-loop condition to evaluate to false on the second iteration. As method COND() is evaluated each time the innermost for-loop executes and the outermost and innermost for-loops execute $n \log_2 n$ and $n$ times, respectively, the algorithm runs in $O(n \log_2 n \times n \times (n + 13)) = O(n^3 \log_2 n)$ time.
2. (30 marks) Prove or disprove the following:

(a) (10 marks) \( f(n) = (n - 2)(n - 6) \) is not \( \Theta(n^2) \).

(b) (10 marks) \( f(n) = n^d + 10n^2 \), where \( d \) is some integer constant greater than or equal to 2, is \( O(n^d) \).

(c) (10 marks) \( f(n) = 10^{127}2^n \) is \( \Omega(3^n) \).

All proofs should give appropriate bounds on values of \( c \) and \( n_0 \).

**Answer:** In each case below, we will work from the definitions for \( O(g(n)) \) and \( \Omega(g(n)) \), either to show that the inequalities in these definitions hold or that we can derive a contradiction.

- Proof that \( f(n) = (n - 2)(n - 6) \) is \( \Theta(n^2) \): As \( (n - 6)(n - 2) = n^2 - 8n + 12 \), this can be rewritten as \( n^2 - 8n + 12 \leq c_1n^2 \) (the big-Oh part) and \( n^2 - 8n + 12 \geq c_2n^2 \) (the big-Omega part). The first inequality holds for \( c_1 = 1 \) and \( n_0,1 = 8 \) and the second inequality holds for \( c_2 = \frac{1}{8} \) and \( n_0,2 = 8 \).

- Proof that \( f(n) = n^d + 10n^2 \), where \( d \) is some integer constant greater than or equal to 2, is \( O(n^d) \): This can be rewritten as \( n^d + 10n^2 \leq cn^d \). This inequality holds for \( c = 11 \) and \( n_0 = 1 \) when \( d \geq 2 \).

- Proof that \( f(n) = 10^{127}2^n \) is not \( \Omega(3^n) \): This can be rewritten as follows:

\[
\begin{align*}
10^{127}2^n &\geq c3^n \\
\log_2(10^{127}2^n) &\geq \log_2(c3^n) \\
\log_2 10^{127} + \log_2 2^n &\geq \log_2 c + \log_2 3^n \\
127\log_2 10 + n &\geq \log_2 c + n \log_2 3 \\
127\log_2 10 + (n - n \log_2 3) &\geq \log_2 c
\end{align*}
\]

As \( \log_2 3 > 1 \), the quantity \( n - n \log_2 3 \) is negative for positive values of \( n \); moreover, this quantity goes to negative infinity as \( n \) goes to infinity. Therefore, this inequality is false for any \( c \) for sufficiently large values of \( n \).

3. (24 marks) Solve the following recurrences using the iteration method:

(a) (12 marks)

\[
T(n) = \begin{cases} 
27 & n \leq 4 \\
T(n - 4) + cn^2 & n > 4
\end{cases}
\]

**Answer:** \( T(n) = \frac{c}{12}(n^3 - 6n^2 + 8n) + 27 \) (see Figure 1).
(b) (12 marks)

\[ T(n) = \begin{cases} 
3 & n \leq 1 \\
8T(n/2) + cn & n > 1 
\end{cases} \]

**Answer:** \( T(n) = \frac{c}{3}(4n^3 - n) + 3n^3 \) (see Figure 2).

4. (16 marks) For each of the recursive algorithms below, derive a recurrence describing the exact running time of that algorithm.

(a) (8 marks)

\begin{verbatim}
FUNKY-REC1(n, x)
  if (n <= 2)
    print x
  else
    for i = 1 to n * log(n) do
      if i is odd then
        x = x + i
      else
        x = x - 1
      FUNKY-REC1(n - 2, x)
    x = x * x
    FUNKY-REC1(n - 3, x - 1)
    for i = 1 to n do
      x = x / i
    FUNKY-REC1(n - 2, x)
\end{verbatim}

**Answer:**

\[ T(n) = \begin{cases} 
O(1) & n \leq 2 \\
2T(n - 2) + T(n - 3) + O(n \log_2 n) & \text{otherwise} 
\end{cases} \]
\[ T(n) = cn^2 + T(n - 4) \]
\[ = cn^2 + c(n - 4)^2 + T(n - 8) \]
\[ = cn^2 + c(n - 4)^2 + c(n - 8)^2 + T(n - 12) \]
\[ \vdots \]
\[ = \sum_{i=0}^{n/4} c(n - 4i)^2 + T(\leq 1) \]
\[ = c \sum_{i=0}^{n/4} (n - 4i)^2 + T(\leq 1) \]
\[ = c \sum_{i=0}^{n/4} n^2 - 8ni + 16i^2 + T(\leq 1) \]
\[ = c \sum_{i=0}^{n/4} n^2 - c \sum_{i=0}^{n/4} 8ni + c \sum_{i=0}^{n/4} 16i^2 + T(\leq 1) \]
\[ = cn^2 \sum_{i=0}^{n/4} 1 - 8cn \sum_{i=0}^{n/4} i + 16c \sum_{i=0}^{n/4} i^2 + T(\leq 1) \]
\[ = \frac{c}{4} n^3 - 8cn \frac{n(\frac{n}{2} + 1)}{2} + 16c \frac{n(\frac{n}{4} + 1)(\frac{n}{2} + 1)}{6} + T(\leq 1) \]
\[ = \frac{c}{4} n^3 - 8cn \frac{n^2 + 4n}{32} + 16c \frac{n^3 + 6n^2 + 8n}{6 \times 32} + T(\leq 1) \]
\[ = \frac{c}{4} n^3 - \frac{c}{4} (n^3 + 4n^2) + \frac{c}{12} (n^3 + 6n^2 + 8n) + T(\leq 1) \]
\[ = -n^2 + \frac{c}{12} (n^3 + 6n^2 + 8n) + T(\leq 1) \]
\[ = \frac{c}{12} (n^3 - 6n^2 + 8n) + 27 \]

Figure 1: Answer to Question 3(a).
\[ T(n) = cn + 8T(n/2) \]
\[ = cn + 8(c(n/2) + 8T(n/4)) \]
\[ = cn + 8c(n/2) + 8^2T(n/4) \]
\[ = cn + 8c(n/2) + 8^2(c(n/4) + 8T(n/8)) \]
\[ = cn + 8c(n/2) + 8^2c(n/4) + 8^3T(n/8) \]
\[ = cn + 8c(n/2) + 8^2c(n/4) + 8^3(c(n/8) + 8T(n/16)) \]
\[ = cn + 8c(n/2) + 8^2c(n/4) + 8^3c(n/8) + 8^4T(n/16) \]
\[ \ldots \]
\[ = \sum_{n=0}^{\log_2 n} 8^{i}c(n/2^{i}) + 8^{\log_2 n}T(1) \]
\[ = \sum_{n=0}^{\log_2 n} (8^{i}/2^{i})n + 2^{3\log_2 n}T(1) \]
\[ = cn \sum_{n=0}^{\log_2 n} (8/2)^i + 2^{3\log_2 n^3}T(1) \]
\[ = cn \sum_{n=0}^{\log_2 n} 4^{i} + n^3T(1) \]
\[ = cn \frac{4^{\log_2 n+1} - 1}{4 - 1} + 3n^3 \]
\[ = cn \frac{2^{2\log_2 n+2} - 1}{3} + 3n^3 \]
\[ = cn \frac{2^{\log_2 n^2+2} - 1}{3} + 3n^3 \]
\[ = cn \frac{4n^2 - 1}{3} + 3n^3 \]
\[ = \frac{c}{3}(4n^3 - n) + 3n^3 \]

Figure 2: Answer to Question 3(b).
(b) (8 marks)

FUNKY-REC2(n, x, k)
  if (n <= 3)
    print (x - 13 * n)
  else if (n <= 10)
    for i = 1 to n * k do
      print x
  else if (n is odd)
    FUNKY-REC2(n / 3, x - 1, k)
    for i = 1 to n do
      x = x / i
      FUNKY-REC2(n / 3, x - 5, k)
  else if (n <= 6)
    for i = 1 to k * log(k) do
      print x / n
  else
    for i = 1 to n * log(n) do
      x = x * i
    FUNKY-REC2(n - 3, x * x, k)

Answer:

\[
T(n, k) = \begin{cases} 
  O(1) & \text{if } n \leq 3 \\
  O(k) & \text{if } 4 \leq n \leq 10 \\
  2T(n/3, k) + O(n) & \text{if } n \geq 11 \text{ and } n \text{ is odd} \\
  T(n - 3, k) + O(n \log_2 n) & \text{otherwise}
\end{cases}
\]