TABLE OF CONTENTS

PART I: Analysis of Normal Code

QUESTION#1

Choose the best matching order of growth of the running time

1.
```
public static int f1(int N) {
    int x = 0;
    for (int i = 0; i < N; i++)
        x++;
    return x;
}
```
(A) N (B) X (C) NX

2.
```
public static int f2(int N, int R) {
    int x = 0;
    for (int i = 0; i < R; i++)
        x += f1(i);
    return x;
}
```
(A) 2^R (B) R^2 (C) RN

3.
```
public static int f4(int N, int R) {
    int x = 0;
    for (int i = 0; i < N; i++)
        for (int j = 1; j <= R; j += j)
            x++;
    return x;
}
```
(A) RN (B) N Log(R) (C) N^R

Question	Answer
1	
2	
3	
4	
5	

QUESTION#2

For each group of functions, sort the functions in increasing order of asymptotic (big-O) complexity:

Group1	Group2	Group3
$f_1(n) = n^{0.999999}\log n$	$f_1(n) = 2^{2^{1000000}}$	$f_1(n) = n^{\sqrt{n}}$
$f_2(n) = 10000000n$	$f_2(n) = 2^{100000n}$	$f_2(n) = 2^n$
$f_3(n) = 1.000001^n$	$f_3(n) = \binom{n}{2}$	$f_3(n) = n^{10}\cdot 2^{n/2}$
$f_4(n) = n^2$	$f_4(n) = n\sqrt{n}$	$f_4(n) = \sum_{i=1}^{n}(i+1)$

1 -> 2 -> 4 -> 3

Group1

Solution: The correct order of these functions is $f_1(n), f_2(n), f_4(n), f_3(n)$. To see why $f_1(n)$ grows asymptotically slower than $f_2(n)$, recall that for any $c > 0$, $\log n$ is $O(n^c)$. Therefore we have:

$$f_1(n) = n^{0.999999} \log n = O(n^{0.999999} \cdot n^{0.000001}) = O(n) = O(f_2(n))$$

The function $f_2(n)$ is linear, while the function $f_4(n)$ is quadratic, so $f_2(n)$ is $O(f_4(n))$. Finally, we know that $f_3(n)$ is exponential, which grows much faster than quadratic, so $f_4(n)$ is $O(f_3(n))$.

Group2

Solution: The correct order of these functions is $f_1(n), f_4(n), f_3(n), f_2(n)$. The variable n never appears in the formula for $f_1(n)$, so despite the multiple exponentials, $f_1(n)$ is constant. Hence, it is asymptotically smaller than $f_4(n)$, which does grow with n. We may rewrite the formula for $f_4(n)$ to be $f_4(n) = n\sqrt{n} = n^{1.5}$. The value of $f_3(n) = \binom{n}{2}$ is given by the formula $n(n-1)/2$, which is $\Theta(n^2)$. Hence, $f_4(n) = n^{1.5} = O(n^2) = O(f_3(n))$. Finally, $f_2(n)$ is exponential, while $f_3(n)$ is quadratic, meaning that $f_3(n)$ is $O(f_2(n))$.

PART II: Analysis of Recursive Code

QUESTION#1

<table>
<tr><td rowspan="6">What's the exact <u>complexity</u> of the following code?
<pre>Fun2(A, S, E)
      IF (S == E) return A[S]
      S1 = (S + E) / 3
      S2 = 2 × (S + E) / 3
      R1 = Fun2(A, S, S1)
      R2 = Fun2(A, S1+1, S2)
      R3 = Fun2(A, S2+1, E)
      Return Max(R1, R2, R3)</pre></td><td>a. $\Theta(N)$</td></tr>
<tr><td>b. $\Theta(N^3)$</td></tr>
<tr><td>c. $\Theta((\log(N))^3)$</td></tr>
<tr><td>d. $\Theta(\log(N))$</td></tr>
<tr><td>e. $\Theta(3 \times \log(N))$</td></tr>
<tr><td>f. $\Theta(1)$</td></tr>
</table>

QUESTION#2

For the following "Main" function:

<table>
<tr><td><pre>Main(A, N)
      I = N
      Sum = 0
      For (J = 0; J < I; J += N/2)
            While (I > 1)
                  Sum = Sum + 1
                  I = I - 1
            EndWhile
            I = N
      EndFor
      Return 5 × Fun(N, 4)
End Main</pre></td><td><pre>Fun(Z, L)
      If (Z == 2)
            Return 0
      For I = 1 to L
            If (random(1,1000) % 2 == 0)
                  X += I × Fun(Z/L, L)
            Else
                  X += (I + 1) × Fun(Z/(2×L), L)
      EndFor
End Fun
//random(1,1000): generates a random integer
between 1 & 1000 in O(1)</pre></td></tr>
</table>

1. What's the complexity of the non-recursive part of code? theta(n)
2. What's the recurrence equation (T(N)) that represents the LOWER BOUND of the entire code? T(N) = 4T(N/8) + theta(1); --> theta(n^0.6667)
3. What's the LOWER BOUND complexity of entire code? theta(n)
4. What's the recurrence equation (T(N)) that represents the UPPER BOUND of the entire code? T(N) = 4T(N/4) + theta(1); --> theta(n)
5. What's the UPPER BOUND complexity of entire code? theta(n)

so, as lower bound complexity == upper bound complexity
then, theta of entire code = n;

PART III: **Asymptotic Notations**

QUESTION#1

Chose the notation that **BEST REPRESENTS** the **WORST** case?

```
Fun2 (N)
     Sorted = true
     I = 1
     While (I < N AND Sorted ==
     true)
          If (A[I] > A[I + 1])
               Sorted = false
          End If
          I = I + 1
     End while
      Print Sorted
```

a.	O(1)
b.	Θ(1)
c.	O(N)
d.	Θ(N)
e.	Ω(1)
f.	Ω(N)
g.	Θ(N-1)
h.	None of the choices

QUESTION#2

Let f(N) = 10N×Log_2(N) and g(N) = N, what's the possible value(s) of N_0 that makes f(N) = Ω(g(N)) for constant C = 20?

a. 1≤ N0≤32	**b.** 0< N0 ≤ 2	**c.** N0 ≥ 2	**d.** N0 ≥ 4	**e.** No values exist. The relation is not valid

QUESTION#3

Given the following orders (***log*** is base 2):

$1)\sqrt{e^{\log(N)} + N^3}, 2)\ 1/\sqrt{\log(N)}, 3)\ N, 4)\ N^3, 5)\ 4^{\log(N)}, 6)\ N\log(N)$

Arrange them in **increasing order of growth rate** (with g(n) following f(n) in your list if and only if f(n)=O(g(n))). Chose the correct order?

a. 2,3,6,5,1,4	**b.** 2,3,1,6,5,4	**c.** 2,6,3,1,5,4	**d.** 2,3,6,1,5,4	**e.** 4,5,1,3,6,2	**f.** None of choice

PART IV: Recurrence Equations

QUESTION#1

Using the RECURSION TREE method, Answer the following questions:

$$\mathbf{T(N) = T(N/5) + T(7N/10) + \Theta(N)\,;T(1) = 1}$$

Question	Answer
1. What's the EXACT total number of levels in this tree?	lg10/7(n)
2. What's the complexity of each level?	(9/10)^i * n
3. What's the complexity of LAST level?	1
4. What's the UPPER BOUND order of this tree?	**O(** n **)**

QUESTION#2

Given T(N) = 64 T(N / 16) + (22/7) N√N, using Master Method: what's the correct value of Ɛ

- ○ 0 < Ɛ ≤ 1.5
- ○ 0 < Ɛ ≤ 0.5
- ○ Ɛ = 1.6
- ○ No possible Ɛ
- ✓ It's case2: no Ɛ is required

QUESTION#3

1. Select the correct asymptotic complexity of an algorithm with runtime T(n; n) where:

$$\begin{aligned} T(x,c) &= \Theta(x) && \text{for } c \le 2, \\ T(c,y) &= \Theta(y) && \text{for } c \le 2, \text{ and} \\ T(x,y) &= \Theta(x+y) + T(x/2, y/2). \end{aligned}$$

1. $\Theta(\log n)$.
2. $\Theta(n)$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.
5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

Solution: The correct answer is $\Theta(n)$. To see why, we rewrite the recurrence relation to avoid Θ notation as follows:

$$T(x, y) = c(x + y) + T(x/2, y/2).$$

We may then begin to replace $T(x/2, y/2)$ with the recursive formula containing it:

$$T(x, y) = c\,(x + y) + c\left(\frac{x+y}{2}\right) + c\left(\frac{x+y}{4}\right) + c\left(\frac{x+y}{8}\right) + \dots$$

This geometric sequence is bounded from above by $2c(x + y)$, and is obviously bounded from below by $c(x + y)$. Therefore, $T(x, y)$ is $\Theta(x + y)$, and so $T(n, n)$ is $\Theta(n)$.

2. Select the correct asymptotic complexity of an algorithm with runtime T(n; n) where:

$$\begin{aligned} T(x, c) &= \Theta(x) && \text{for } c \le 2, \\ T(c, y) &= \Theta(y) && \text{for } c \le 2, \text{ and} \\ T(x, y) &= \Theta(x) + T(x, y/2). \end{aligned}$$

1. $\Theta(\log n)$.
2. $\Theta(n)$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.
5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

Solution: The correct answer is $\Theta(n \log n)$. To see why, we rewrite the recurrence relation to avoid Θ notation as follows:

$$T(x, y) = cx + T(x, y/2).$$

We may then begin to replace $T(x, y/2)$ with the recursive formula containing it:

$$T(x, y) = \underbrace{cx + cx + cx + \ldots + cx}_{\Theta(\log y) \text{ times}}.$$

As a result, $T(x, y)$ is $\Theta(x \log y)$. When we substitute n for x and y, we get that $T(n, n)$ is $\Theta(n \log n)$.

3. Select the correct asymptotic complexity of an algorithm with runtime T(n; n) where:

$$\begin{aligned} T(x, c) &= \Theta(x) && \text{for } c \leq 2, \\ T(x, y) &= \Theta(x) + S(x, y/2), \\ S(c, y) &= \Theta(y) && \text{for } c \leq 2, \text{ and} \\ S(x, y) &= \Theta(y) + T(x/2, y). \end{aligned}$$

1. $\Theta(\log n)$.
2. $\boxed{\Theta(n)}$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.
5. $\Theta(n^2)$.
6. $\Theta(2^n)$.