1) Select the **tightest big-O** expression for the following pseudo codes (30 points)

1.1) 
```java
int sum = 0;
for(int i = 0; i < n; i++)
    for(int j = 0; j < n * n; j++)
        sum ++;
```

A. $O(n)$  
B. $O(n^2)$  
C. $O(n^3)$ → **Correct**: Two nested loops with the inner one loops until $n^2$ → $O(n^2)$, while, the outer loop is linear $O(n)$  
D. $O(n^4)$

1.2) 
```java
int sum = 0;
for(int i = 0; i < n; i++)
    for(int j = 0; j < i; j++)
        sum ++;
```

A. $O(n)$  
B. $O(n \log n)$  
C. $O(n^2)$ **Correct**: Two nested loops with both loop until $n$. Both are linear.  
D. $O(n^2 \log n)$

1.3) 
```java
int prod = 3;
for(int i = 1; i <= n; i = i * 2)
    for(int j = 0; j < n; j++)
        prod = prod * prod;
```

A. $O(n)$  
B. $O(n \log n)$ → **Correct**: Two nested loops, with the outer loop logarithmic → $\log n$  
C. $O(n^2)$ and the inner loop is linear $O(n)$  
D. $O(n^2 \log n)$
1.4)
int sum(n) {
    if(n == 1) {
        return 1;
    }
    return n + sum(n - 1);
}

A. $O(n)$ → Correct: recursive call with one fork up to $n$ calls.
B. $O(n \log n)$
C. $O(n^2)$
D. $O(n^2 \log n)$

1.5)
count1=0; count2=0; count3=0;
for(i=0; i<n; i++)
    for(j=1; j<n; j*=2)  {
        count1++;
        count2++;
        count3++;
    }
for(i=0; i<n; i++) {
    count1++;
    count2++;
    count3++;
}

A. $O(n)$
B. $O(n \log n)$ → Correct: the first two nested loops are $O(n \log n)$ as the inner loop
C. $O(n^2)$ is logarithmic with step of $j=j*2$. Then the last loop is $O(n)$. Hence,
D. $O(n^2 \log n)$ $O(n \log n + n) = O(n \log n)$.

2) Answer the following questions on the below expressions (15 points)

2.1) Choose tightest Big-O for:
$n^3 + n! + 3^n + 2147483647 \times \log^3 n$
A. $O(n^3)$
B. $O(n!)$ → Correct: $n!$ is the most dominant complexity in this expression
C. $O(3^n)$
D. $O(2147483647 \times \log^3 n)$
2.2) Choose tightest Big-Ω for:
\[ \sum_{i=0}^{2n} 5i + i^2 \]

A. \( \Omega(n^3) \) → Correct: the expression will have \( O(n) + O(n^2) \) \( n \) times. This yields a complexity of \( O(n^3) \)
B. \( \Omega(n^2) \)
C. \( \Omega(3^n) \)
D. \( \Omega(2147483647 \times log^3 n) \)

3) What is the worst-case amount of work to find a particular card in a deck of cards of size \( n \)? Note that cards are unique and are not sorted (10 pts).

\( O(n) \). In the worst case scenario, you will need to check all the cards

4) Order the following functions by growth rate. Indicate which functions grow at the same rate (15 points)

\( N, \ N^2, \ log N, \ log(N^2), \ \log^2 N, \ N \log N, \ 2, \ 2^N, \ 37, \ N^2 \log N, \ 5\log N, \ N^3, \ 10N \log N^2 [\text{We put the same growth rate in square brackets}] \)

\( [2, \ 37], \ [\log N, 5\log N, \ \log(N^2)], \ \log^2 N, N , [N \log N, 10N \log N^2], N \log^2 N, \ N^2, \ N^2 \log N, \ N^3, \ 2^N \)

5) In each of the following problems, you are given two algorithms that do the same job. You are required to indicate which of the two algorithms is “better” and why (30 points)

a. The following two algorithms count the number of values that are above the average of a given set of values.

**Algorithms:** AboveAvg1 & AboveAvg2

**Input** → \( L \): List of integer values, \( N \): size of the List

**Output** → \( C \): Number of values above the average of all values

<table>
<thead>
<tr>
<th>AboveAvg1 ( (L, \ N) )</th>
<th>AboveAvg2 ( (L, \ N) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Set Avg = 0, count = 0, C = 0</td>
<td></td>
</tr>
<tr>
<td>2. For I = 0 to N - 1</td>
<td></td>
</tr>
<tr>
<td>1- Set count = count + L[I]</td>
<td></td>
</tr>
<tr>
<td>3. Set Avg = count/N</td>
<td></td>
</tr>
<tr>
<td>4. For I = 0 to N - 1</td>
<td></td>
</tr>
<tr>
<td>1- IF (L[I] &gt; Avg )</td>
<td></td>
</tr>
<tr>
<td>1- Set C = C + 1</td>
<td></td>
</tr>
<tr>
<td>5. Return C</td>
<td></td>
</tr>
<tr>
<td>1. Set Avg = 0, C = 0</td>
<td></td>
</tr>
<tr>
<td>2. For J = 0 to N - 1</td>
<td></td>
</tr>
<tr>
<td>1- Set count = 0</td>
<td></td>
</tr>
<tr>
<td>2- For I 0 to N - 1</td>
<td></td>
</tr>
<tr>
<td>1- set count = count + L[I]</td>
<td></td>
</tr>
<tr>
<td>3- Set Avg = count/N</td>
<td></td>
</tr>
<tr>
<td>4- IF (L[J] &gt; Avg )</td>
<td></td>
</tr>
<tr>
<td>1- Set C = C + 1</td>
<td></td>
</tr>
<tr>
<td>3. Return C</td>
<td></td>
</tr>
</tbody>
</table>
Solution to a:
AboveAvg1 is better.

AboveAvg1: has two separate loops, each executes N times = 2N = O(N)

AboveAvg2: Two nested loops = O(N^2)
In contrast to AboveAvg1, this algorithm calculates the average every time it wants to check if a value above avg or not which is unnecessary as avg will not change and can be calculated once

b. The following two algorithms take a list L and output another list M such that M contains the even numbers of L first then L’s odd numbers.

**Algorithms:** EvensFirst1 & EvensFirst2

**Input** → L: List of integer values, n: size of List L

**Output** → M: List of integers such that even numbers are first then odd numbers

<table>
<thead>
<tr>
<th>EvensFirst1 (L, n, M)</th>
<th>EvensFirst2 (L, n, M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. J = 0</td>
<td>1. J = 0</td>
</tr>
<tr>
<td>2. For I = 0 to n - 1</td>
<td>2. K = Size - 1</td>
</tr>
<tr>
<td>1- IF ( L[I] % 2 == 0 ) // Even</td>
<td>3. For I = 0 to n - 1</td>
</tr>
<tr>
<td>a. M[J] = L[I]</td>
<td>1- IF ( L[I] % 2 == 0 ) // Even</td>
</tr>
<tr>
<td>3. For I = 0 to n - 1</td>
<td>b. J = J+1</td>
</tr>
<tr>
<td>2- IF ( L[I] % 2 != 0 ) // Odd</td>
<td>2- ELSE // Odd</td>
</tr>
<tr>
<td>b. J = J+1</td>
<td>b. K = K - 1</td>
</tr>
<tr>
<td>4. Return M</td>
<td>4. Return M</td>
</tr>
</tbody>
</table>

Solution to b:
Both EvensFirst1 and EvensFirst2 have the same complexity O(n)

However, Note that EvensFirst2 is better in execution time because it iterates on the list once (n) as for EvensFirst1, it iterates on it twice (2*n)

Submit Instructions:
The homework must be turned in by the due date and time at GradeScope.