Problem 1 (10 points).

(a) (5 points) For each pair of functions \( f \) and \( g \) in the following table, indicate whether \( f = O(g) \), \( f = \Omega(g) \) and \( f = \Theta(g) \) respectively.

<table>
<thead>
<tr>
<th>( f(n) )</th>
<th>( g(n) )</th>
<th>( O )</th>
<th>( \Omega )</th>
<th>( \Theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \log_{10}(n/2) )</td>
<td>( \log_{2}(n^7) )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \lceil \sqrt{10n^2 + 100n} \rceil )</td>
<td>( n )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n^3 - 100n )</td>
<td>( 100n^2 \log n )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( (\log n)^{\log n} )</td>
<td>( n^2 \log^2 n )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( (n - 1)! )</td>
<td>( (4/3)^n )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

([x] means the smallest integer larger than or equal to \( x \), e.g. \( [3] = 3, [2.2] = 3 \)

(b) (5 points) Justify your answer for the question “whether \( \lceil \sqrt{10n^2 + 100n} \rceil = O(n) ? \)”, using both the two equivalent definitions of the \( O \)-notation:

(i) Show that there exists constant \( c, n_0 > 0 \) s.t. when \( n > n_0 \), \( \lceil \sqrt{10n^2 + 100n} \rceil \) and \( cn \) always satisfy some relation.

(ii) Limit test: analyzing the result of \( \lim_{n \to \infty} \frac{\lceil \sqrt{10n^2 + 100n} \rceil}{n} \)

Problem 2 (16 points).

(2a) (4 points). Given an array \( A \) of \( n \) integers, we need to check if there are two integers in the array with summation equal 0. Consider the following simple algorithm:

1: \( \text{for } i \leftarrow 1 \text{ to } n-1 \text{ do} \)
2: \( \quad \text{for } j \leftarrow i+1 \text{ to } n \text{ do} \)
3: \( \quad \quad \text{if } A[i] + A[j] = 0 \text{ then return yes} \)
4: \( \quad \text{return no.} \)
Give a **tight** upper bound (i.e., a $\Theta(\cdot)$ bound) on the running time of the algorithm and justify your answer.

**Problem 2 (12 points).** Now suppose we have the same problem as (2a) except that the array $A$ is sorted in non-decreasing order. Consider the following algorithm:

```
1: $i \leftarrow 1, j \leftarrow n$
2: while $i < j$ do
3: if $A[i] + A[j] = 0$ then return yes
4: if $A[i] + A[j] < 0$ then $i \leftarrow i + 1$ else $j \leftarrow j - 1$
5: return no
```

Briefly argue about the correctness of the algorithm and give a **tight** upper bound on the running time of the algorithm (here you do NOT need to justify the upper bound). To prove the correctness you need to show that: (i) the algorithm always terminates (i.e., it won’t loop forever); (ii) when there do exists some pair $A[i] + A[j] = 0$ (note there can be multiple such pairs), the algorithm will always return yes; (iii) when no such pair exists, the algorithm can only return no.

**Problem 3 (24 points).** For problem (3a), you can either write down the edges or draw the DFS/BFS tree. For problem (3b) and (3c), write your algorithm as pseudo-code, and explain the ideas using a few words.

**Problem 3 (24 points).** For problem (3a), you can either write down the edges or draw the DFS/BFS tree. For problem (3b) and (3c), write your algorithm as pseudo-code, and explain the ideas using a few words.

(3a) **(8 points).** Using DFS and BFS to traverse the graph shown in Figure 1 starting from vertex $a$. List the edges included in the DFS tree and BFS tree. Here we assume the vertices are explored in lexicographic order: for example, when you are checking the neighbors of vertex $a$, you should first look at $b$, then $c$, then $d$.

![Figure 1: Traverse the graph using DFS and BFS](image)
A cycle in an undirected graph $G = (V, E)$ is a sequence of $t \geq 3$ different vertices $v_1, v_2, \ldots, v_t$ such that $(v_i, v_{i+1}) \in E$ for every $i = 1, 2, \ldots, t - 1$ and $(v_1, v_t) \in E$. Given the adjacency-list representation of an undirected graph $G = (V, E)$, design an $O(n + m)$-time algorithm to decide if $G$ contains a cycle or not. (Here $n = |V|$ and $m = |E|$)

(Hint: modify DFS/BFS)

A cycle in a directed graph $G = (V, E)$ is a sequence of $t \geq 2$ different vertices $v_1, v_2, \ldots, v_t$ such that $(v_i, v_{i+1}) \in E$ for every $i = 1, 2, \ldots, t - 1$ and $(v_t, v_1) \in E$. Given the adjacency-list representation of a directed graph $G = (V, E)$, design an $O(n + m)$-time algorithm to decide if $G$ contains a cycle or not. (Here $n = |V|$ and $m = |E|$)

Figure 2: Cycles in undirected and directed graphs. $(1, 2, 5, 3)$ is a cycle in the undirected graph. $(1, 2, 5, 6, 7, 3)$ is a cycle in the directed graph. However, $(1, 2, 5, 8, 3)$ is not a cycle in the directed graph.

Remark On a cycle of a directed graph, the directions of the edges have to be consistent. See Figure 1. So, converting a directed graph to a undirected graph and then using algorithm for (3a) does not give you a correct algorithm for (3b). (Hint: A directed graph with cycle means you cannot order all vertices in one direction.)