Name (first last) ........................................................................
Student ID ............................................................................

• This quiz is **closed book, closed notes** and 35 minutes long
• Read the questions carefully
• No electronic equipment allowed (cell phones, tablets, computers, ...)
• Write legibly. What can’t be read will not be graded
• Use pseudocode (or English) to describe your algorithms
• Always remember to analyze the time complexity of your solution
• If you have a question about the meaning of a question, raise your hand

1 \[ \square \] /40
2 \[ \square \] /24
3 \[ \square \] /36
Total \[ \square \] /100
Problem 1. (40 points: 4 points if correct, 2 if unanswered, 0 if wrong)
Mark by true or false each of the following (no need to prove).

\[ \sqrt{n \log_2 n^3} + n \log_2 n \in O(n \log_2 n) \]  \[ \Box \text{True} \quad \Box \text{False} \]

\[ 9^{\log_3 n} \in \Omega(n^2 \log n) \]  \[ \Box \text{True} \quad \Box \text{False} \]

\[ \log_2 2^{n^2} - n \in \Theta(n \log_3 3^n) \]  \[ \Box \text{True} \quad \Box \text{False} \]

An array sorted in increasing order is always a min-heap  \[ \Box \text{True} \quad \Box \text{False} \]

A min-heap is always a sorted array (in increasing order)  \[ \Box \text{True} \quad \Box \text{False} \]

The following questions are on graphs; assume that \( n = |V| \) is the number of vertices, and \( m = |E| \) is the number of edges; DFS is “depth first search”; BFS is “breadth first search”; in DFS/BFS the set of edges visited during the execution of these algorithms are called tree or discovery edges; non-tree edges are the others (also called back edges in DFS, cross edges in BFS)

A directed complete graph with \( n \) nodes has exactly \( n(n - 1)/2 \) edges  \[ \Box \text{True} \quad \Box \text{False} \]

Given the spanning tree \( T \) formed by the discovery (tree) edges of a BFS traversal of a connected undirected graph \( G \) started from node \( s \), for each vertex \( v \), the path on tree \( T \) is the shortest path between \( s \) and \( v \)  \[ \Box \text{True} \quad \Box \text{False} \]

An edge \( e \) whose removal disconnects the graph is called a bridge; if BFS is run on a connected undirected graph \( G \), it is a possible for a bridge in \( G \) to be a cross (non-tree) edge  \[ \Box \text{True} \quad \Box \text{False} \]

For a connected undirected graph \( G \), the absence of back (non-tree) edges with respect to a DFS tree implies that \( G \) is acyclic  \[ \Box \text{True} \quad \Box \text{False} \]

If one runs a DFS on a connected undirected graph, the number of back (non-tree) edges is exactly \( m - n + 1 \)  \[ \Box \text{True} \quad \Box \text{False} \]
Problem 2. (24 points: 6 points each)

For each of the concepts listed below write a precise (possibly formal) definition. Do not explain or comment about the corresponding algorithm, if any.

1. The topological ordering of a directed acyclic graph $G = (V, E)$ is an ordering of its vertices, say $\{v_1, v_2, \ldots \}$, such that for every directed edge $(v_i, v_j) \in E$, we have that $i < j$.

2. A directed cycle in a directed graph $G = (V, E)$ is a set of directed edges
   \[\{(u_1, u_2), (u_2, u_3), \ldots, (u_{l-2}, u_{l-1}), (u_{l-1}, u_l)\}\text{ where } u_l = u_1\]

3. A spanning tree of an undirected graph $G = (V, E)$ is any acyclic subgraph of $G$, i.e., $T = (V, E')$ such that $E' \subseteq E$ and $T$ acyclic.

4. A transitive closure of a directed graph $G = (V, E)$ is a graph $G = (V, E')$ where $(u, v) \in E'$ if there is a directed path from $u$ to $v$ in $G$. 
Problem 3. (36 points)

Suppose you are given an array $A = \{a_1, a_2, \ldots, a_n\}$ of $n$ distinct integers. You are told that the sequence of values $a_1, a_2, \ldots, a_n$ is unimodal, that is for some index $p \in [1, n]$, the values in the array increase up to position $p$ in $A$, and then decrease the remainder of the way until position $n$. Give an algorithm to find the position $p$ in $O(\log n)$ time. You can assume $n$ to be a power of 2.

Answer: The algorithm works like a binary search. Compare the elements $A[n/2], A[n/2-1]$ and $A[n/2+1]$ to decide whether to search on the left, on the right, or whether we are done. More specifically

- if $A[n/2−1] < A[n/2] < A[n/2+1]$, then search recursively in the entries $A[n/2+1 \ldots n]$

The algorithm has the same structure of binary search, its recurrence relation is $T(n) = T(n/2) + O(1)$, which has solution $O(\log n)$. 