1. (40 pts.) **Collapse and shrink.**
   (A) (5 pts.) You are given an undirected graph with \( n \) vertices and \( m \) edges, with positive weights on the edges (for simplicity, you can assume all the weights are distinct). Consider the procedure that given a weight \( x \), outputs the graph \( G_{\leq x} \) that results from removing all the edges in \( G \) that have weight larger than (or equal to) \( x \). Describe (shortly – no need for pseudo code) an algorithm for computing \( G_{\leq x} \). How fast is your algorithm?
   The graph \( G_{\leq x} \) might not be connected – how would you compute its connected components?

   (B) (5 pts.) Consider the procedure that receives as input an undirected weighted graph \( G \), and a partition \( \mathcal{V} \) of the vertices of \( G \) into \( k \) disjoint sets \( V_1, \ldots, V_k \). The **meta graph** \( G(\mathcal{V}) \) of \( G \) induces by \( \mathcal{V} \) is a graph having \( k \) vertices, \( v_1, \ldots, v_k \), where \( v_iv_j \) has an edge if and only if, there is an edge between some vertex of \( V_i \) and some vertex of \( V_j \). The weight of such an edge \( v_iv_j \) is the minimum weight of any edge between vertices in \( V_i \) and vertices in \( V_j \).
   Describe an algorithm, as fast as possible, for computing the meta-graph \( G(\mathcal{V}) \). You are not allowed to use hashing for this question, but you can use that **RadixSort** works in linear time (see wikipedia if you do not know **RadixSort**). How fast is your algorithm?

   (C) (10 pts.) Consider the randomized algorithm that starts with a graph \( G \) with \( m \) edges and \( n \) vertices. Initially it sets \( G_0 = G \). In the \( i \)th iteration, it checks if \( G_{i-1} \) is a single edge. If so, it stops and outputs the weight of this edge. Otherwise, it randomly choose an edge \( e_i \in E(G_{i-1}) \). It then computes the graph \( H_i = (G_{i-1})_{\leq w(e_i)} \), as described above.
   • If the graph \( H_i \) is connected then it sets \( G_i = H_i \) and continues to the next iteration.
   • Otherwise, \( H_i \) is not connected, then it computes the connected components of \( H_i \), and their partition \( \mathcal{V}_i \) of the vertices of \( G_{i-1} \) (the vertices of each connected component are a set in this partition). Next, it sets \( G_i \) to be the meta-graph \( G_{i-1}(\mathcal{V}_i) \).
   Let \( m_i \) be the number of edges of the graph \( G_i \). Prove that if you know the value of \( m_{i-1} \), then \( \mathbb{E}[m_i] \leq (7/8)m_{i-1} \) (a better constant is probably true). Conclude that \( \mathbb{E}[m_i] \leq (7/8)^i m \).

   (D) (15 pts.) What is the expected running time of the algorithm describe above? **Prove** your answer.
   (The better your bound is, the better it is.)

   (E) (5 pts.) What does the above algorithm computes, as far as the original graph \( G \) is concerned?

2. (30 pts.) **Majority tree**
   Consider a uniform rooted tree of height \( h \) (every leaf is at distance \( h \) from the root). The root, as well as any internal node, has 3 children. Each leaf has a boolean value associated with it. Each internal node returns the value returned by the majority of its children. The evaluation problem consists of determining the value of the root; at each step, an algorithm can choose one leaf whose value it wishes to read.

   (A) (15 pts.) Show that for any deterministic algorithm, there is an instance (a set of boolean values for the leaves) that forces it to read all \( n = 3^h \) leaves. (hint: Consider an adversary argument, where you provide the algorithm with the minimal amount of information as it request bits from you. In particular, one can devise such an adversary algorithm.)

   (B) (10 pts.) Consider the recursive randomized algorithm that evaluates two subtrees of the root chosen at random. If the values returned disagree, it proceeds to evaluate the third sub-tree. If they agree, it returns the value they agree on.
   Write an explicit exact formula for the expected number of leaves being read, for a tree of height \( h = 1 \), and height \( h = 2 \).
(C) (5 pts.) Using (B), prove that the expected number of leaves read by the algorithm on any instance is at most $n^{0.9}$.

3. (30 pts.) **Attack of the edge killers.**

Let $T_h$ denote the full balanced binary tree with $2^h$ leaves. Due to eddies in the space-time continuum, every edge gets deleted with probability half (independently) – let $T'_h = \text{leftover}(T_h)$ denote the remaining tree (rooted at the original root). The remaining tree $T'_h$ is **usable**, if there is a path from the root of the tree to one of the original leaves of the tree.

(A) (10 pts.) Let $\rho_1$ be the probability that leftover($T_1$) is a usable. What is the value of $\rho_1$? Prove your answer.

(B) (10 pts.) Let $\rho_h$ be the probability that a tree $T'_h$ is usable. Give a recursive formula for the value of $\rho_h$ as a function of $\rho_{h-1}$.

(C) (10 pts.) Prove, by induction, that $\rho_h \geq 1/(h + 1)$. 