Problem 1. A Max-Heap with \( n \) elements is a full binary tree with \( n \) nodes, and is represented as an array \( A[1 \ldots n] \). Suppose that instead of a full binary tree, we use a full \( k \)-ary tree. That is, every node can have at most \( k \) children, instead of just two.

(a) How would you represent such a full \( k \)-ary tree using an array \( A[1 \ldots n] \)? In particular, where is every node of the tree stored in the array? For a node stored at location \( A[i] \), where is its parent stored? Where are its children stored?

(b) Where are the leaves of the tree stored in the array?

(c) How would you modify the procedures Max-Heapify, and Build-Max-Heap, so that they can use your new representation? What is the new running time of these procedures?

(d) Based on your above findings, is there a benefit in using a full \( k \)-ary tree, for some \( k > 2 \), instead of a full binary tree?

Problem 2. The running time of the Heapsort algorithm is \( O(n \cdot \log n) \).

(a) What is the best possible running time for Heapsort? Justify your answer by giving an array \( A[1 \ldots n] \), and proving that Heapsort on input \( A \) achieves your claimed running time. Why is this running time the best possible?

(b) Give an array \( A[1 \ldots n] \), and prove that running Heapsort on input \( A \) takes time \( \Omega(n \cdot \log n) \).