Exercise 1

Min/max priority queue. Design a data type that supports the following operations: insert, delete the maximum, and delete the minimum (all in logarithmic time); and and the maximum and and the minimum (both in constant time). Hint: Use two heaps.

Exercise 2

Fast insert. Develop a compare-based implementation of the MinPQ API such that insert uses log log N compares and delete the minimum uses 2 log N compares. Hint : Use binary search on parent pointers to and the ancestor in swim().

Exercise 3

Merging K Sorted Arrays Using Min-Priority Queue. Given a set of K sorted arrays, implement a program that merges them into a single sorted array using a min-priority queue. Design a data type that implements the min-priority queue functionality by storing elements in an unordered array. Your task is to utilize this class to extract the smallest elements from the input arrays efficiently. Start by inserting the first element from each array into the priority queue, and repeatedly extract the minimum element, adding it to the merged result. If the extracted element has a next element in its corresponding array, insert that next element into the priority queue. Finally, print the merged sorted array.

Exercise 4

Top K Frequent Elements. Implement a program that finds the top K frequent elements from a given array of integers. You are provided with a function that takes an integer array and an integer K as inputs. The program should design a data type for max-priority queue to efficiently track and retrieve the K elements that appear most frequently in the array. To achieve this, first count the frequency of each element, then insert them into the priority queue based on their frequency. Finally, extract the top K elements from the priority queue and return them as a vector. For example, given the input array [1, 1, 1, 2, 2, 3] and K = 2, your output should be [1, 2], since 1 appears most frequently.