Your name
CS 261, Spring 2021
Homework 5
Due Monday, May 3

1. Suppose that we wish to use the MinHash technique to maintain a random sample of k elements of a cash-register-model data stream.

(a) Describe how to use an abstract priority queue to update the MinHash sketch of the stream in a constant number of priority queue operations per stream element. (Your answer should not depend on which type of priority queue is used. The sample size k is not a constant for the purposes of this question.)

(b) For a given choice of k, and stream length n, for which combinations of k and n would it be faster (in terms of the O-notation) to use a binary heap for the priority queue, and for which combinations would it be faster to use a flat tree? In order to analyze the flat tree, you may assume that the hashed values in MinHash are integers whose maximum value is polynomial in n.

■

2. Let T be a heap-ordered binary tree in which you know the identity of the tree root and can find the value of each tree node, and the identities of the left and right children of each tree node (if it has them) in constant time per operation. Describe an algorithm that takes as input T and a parameter k, and uses a priority queue to find the k smallest values in T using $O(k)$ priority queue operations. (Your priority queue can be a different structure than T. You should not modify T itself. You do not need to choose which kind of priority queue to use. Your answer should clearly describe what elements you are storing in the priority queue, what their priorities are, and what operations you are performing on the priority queue.)

■

3. The lecture notes describe the worst case of Dijkstra's algorithm when there are m decrease-priority operations and n delete-min operations. Here, m is the number of edges in the input graph and n is the number of vertices. In this worst case, the optimal trade-off for a k-ary heap is to set $k = m/n$, to make equal total times for each kind of operation. This choice leads to an overall running time of $O(m(\log n)/\log(m/n))$. However, some heuristic analyses suggest that in many practical cases Dijkstra's algorithm may only perform $O(n \log(m/n))$ decrease-priority operations. When this turns out to be the number of decrease-priority operations, what is the optimal choice for k and the overall running time? (Don't forget that the algorithm must still do a constant amount of work for each edge in the graph, even if it doesn't do any priority queue operations for that edge.)

■

4. Starting from an empty Fibonacci heap, insert 2^i elements (for some given integer i) and then delete one of them. How many trees will be in the heap after the deletion?

■