Homework Assignment #3
Due: June 9, 2020 at 11:30 a.m.

1. We saw in class how to take the union of two binomial heaps. Now, we wish to union k binomial heaps $H_1, H_2, \ldots, H_k$ together into one big binomial heap. Assume each H_i contains n elements. We will consider a few different approaches for doing this and look at the running time of them.

(a) We could do a sequence of pairwise unions:

```
1  H ← H_1
2  for i ← 2..k
3      H ← Union(H, H_i)
4  end for
5  // Now, H is the result we want
```

Show that this takes $O(k \log n + k \log k)$ time.

(b) We could do pairwise unions in a different order by calling UnionSequence$(1, k)$ for the following recursive algorithm:

```
6  UnionSequence(lo, hi)
7      // precondition: 1 ≤ lo ≤ hi ≤ k
8      // returns a binomial heap consisting of H_lo ∪ H_{lo+1} ∪ ··· ∪ H_hi
9      if lo = hi then
10         return H_lo
11     else
12         m ← ⌈(lo+hi)/2⌉
13         return Union(UnionSequence(lo, m − 1), UnionSequence(m, hi))
14     end if
15 end UnionSequence
```

Show that this takes $O(k \log n)$ time.

Hint: One way to do this is to prove a stronger claim that the algorithm's running time is at most $c(k-1)\log n - d\log(kn)$ time, where c and d are positive constants that you should figure out.

(c) Another approach is to first merge the root lists of all heaps into one long sorted list of all the roots of all k heaps. (This merged list is sorted by the degree of the roots.) Then, traverse this sorted list of roots, linking roots where necessary to ensure that there is only one root of each degree.

Let's just focus on the first part of this algorithm: merging the root lists into a single, sorted list of roots. Describe how to do this in $O(k \log n)$ time.

Remark: once you have the merged list of roots, it is possible to traverse it *backwards* to link up roots of trees in $O(k \log n)$ time, so that the whole union takes $O(k \log n)$ time. But you don't have to show this.