HW 2

Due on **Wednesday, April 14, 2021 at 10am CST**

CS 498: Topics in Algorithms, Spring 2021 Version: **1.0**

Same as previous homework.

The hardness of the following problems differs considerably between problems. I would suggest you work on them together to get a fair division of labor (or form a union and find yourself?).

1 (100 PTS.) All the distances, small or large.

1.A. (30 PTS.) The input is n real numbers $x_1, x_2, \ldots, x_n$ not in sorted order, and a parameter k. Consider the set of distances

$$D = \left\{ |x_i - x_j| \;\middle|\; i < j, \text{ where } i, j \in [\![n]\!] \right\}.$$

Describe an algorithm that outputs the k smallest numbers in D (assume that all the distances are distinct and $|D| = \binom{n}{2}$.) and runs in $O(nk)$ time. (Note, that the same point might participate in several distances being output.)

[You can assume $k = O(n \log n)$ if it helps.)

1.B. By first sorting the numbers, describe an algorithm for this problem that runs in $O(n \log n + k \log n)$ time.

1.C. (30 PTS.) Assume that $k = O(n^{1/4} / \log^3 n)$. Provide an algorithm for the above problem (i.e., again the input is not sorted) that works in $O(n \log k)$ time.

(There is a linear time algorithm in this case, but it relies on a clever trick.)

1.D. (40 PTS.) Prove a lower bound, in the comparison model, showing that no algorithm can solve this problem in time faster than $\Omega(n \log k)$.

2 (100 PTS.) Improved randomness extraction.

We have shown that we can extract, on average, at least $\lfloor \lg m \rfloor - 1$ independent, unbiased bits from a number chosen uniformly at random from $\{0, \ldots, m-1\}$. It follows that if we have k numbers chosen independently and uniformly at random from $\{0, \ldots, m-1\}$ then we can extract, on average, at least $k\lfloor \lg m \rfloor - k$ independent, unbiased bits from them. Give a better procedure that extracts, on average, at least $k\lfloor \lg m \rfloor - 1$ independent, unbiased bits from these numbers.

3 (100 PTS.) Conditional entropy.

The ***conditional entropy*** $\mathbb{H}(Y|X)$ is defined by

$$\mathbb{H}(Y|X) = \sum_{x,y} \mathbf{Pr}[(X = x) \cap (Y = y)] \lg \frac{1}{\mathbf{Pr}[Y = y | X = x]}.$$

If $Z = (X, Y)$, prove that

$$\mathbb{H}(Z) = \mathbb{H}(X) + \mathbb{H}(Y|X).$$

4 (100 PTS.) Linear time Union-Find,

4.A. (10 pts.) With path compression and union by rank, during the lifetime of a Union-Find data-structure, how many elements would have rank equal to $\lfloor \lg n - 5 \rfloor$, where there are n elements stored in the data-structure?

4.B. (10 pts.) Same question, for rank $\lfloor (\lg n)/2 \rfloor$.

4.C. (20 pts.) Prove that in a set of n elements, a sequence of n consecutive Find operations take $O(n)$ time in total.

4.D. (10 pts.) Write a non-recursive version of Find with path compression.

4.E. (30 pts.) Show that any sequence of m MakeSet, Find, and Union operations, where all the Union operations appear before any of the Find operations, takes only $O(m)$ time if both path compression and union by rank are used.

4.F. (20 pts.) What happens in the same situation if only the path compression is used?

5 (100 pts.) Off-line Minimum

The *off-line minimum problem* asks us to maintain a dynamic set T of elements from the domain $\{1, 2, \ldots, n\}$ under the operations Insert and Extract-Min. We are given a sequence S of n Insert and m Extract-Min calls, where each key in $\{1, 2, \ldots, n\}$ is inserted exactly once. We wish to determine which key is returned by each Extract-Min call. Specifically, we wish to fill in an array *extracted*$[1 \ldots m]$, where for $i = 1, 2, \ldots, m$, *extracted*$[i]$ is the key returned by the ith Extract-Min call. The problem is "off-line" in the sense that we are allowed to process the entire sequence S before determining any of the returned keys.

5.A. (20 pts.) In the following instance of the off-line minimum problem, each Insert is represented by a number and each Extract-Min is represented by the letter E:

$$4, 8, E, 3, E, 9, 2, 6, E, E, E, 1, 7, E, 5.$$

Fill in the correct values in the *extracted* array.

5.B. (40 pts.) To develop an algorithm for this problem, we break the sequence S into homogeneous subsequences. That is, we represent S by

$I_1, E, I_2, E, I_3, \ldots, I_m, E, I_{m+1}$,

where each E represents a single Extract-Min call and each I_j represents a (possibly empty) sequence of Insert calls. For each subsequence I_j, we initially place the keys inserted by these operations into a set K_j, which is empty if I_j is empty. We then do the following.

```
Off-Line-Minimum(m,n)
for i ← 1 to n
    do determine j such that i ∈ K_j
        if j ≠ m + 1
            then extracted[j] ← i
                let l be the smallest value greater than j for which set K_l exists
                K_l ← K_j ∪ K_l, destroying K_j
return extracted
```

Argue that the array *extracted* returned by Off-Line-Minimum is correct.

5.C. (40 PTS.)
Describe how to implement OFF-LINE-MINIMUM efficiently with a disjoint-set data structure. Give a tight bound on the worst-case running time of your implementation.

6 (100 PTS.) Tarjan's Off-Line Least-Common-Ancestors Algorithm

The *least common ancestor* of two nodes u and v in a rooted tree T is the node w that is an ancestor of both u and v and that has the greatest depth in T. In the *off-line least-common-ancestors problem*, we are given a rooted tree T and an arbitrary set $P = \{\{u, v\}\}$ of unordered pairs of nodes in T, and we wish to determine the least common ancestor of each pair in P.

To solve the off-line least-common-ancestors problem, the following procedure performs a tree walk of T with the initial call LCA(*root*[T]). Each node is assumed to be colored WHITE prior to the walk.

```
LCA(u)
1   MakeSet(u)
2   ancestor[Find(u)] ← u
3   for each child v of u in T
4       do LCA(v)
5          Union(u, v)
6          ancestor[Find(u)] ← u
7   color[u] ← BLACK
8   for each node v such that {u, v} ∈ P
9       do if color[v] = BLACK
10            then print "The least common ancestor of" u "and" v "is" ancestor[Find(v)]
```

6.A. (20 PTS.) Argue that line 10 is executed exactly once for each pair $\{u, v\} \in P$.

6.B. (20 PTS.) Argue that at the time of the call LCA(u), the number of sets in the disjoint-set data structure is equal to the depth of u in T.

6.C. (30 PTS.) Prove that LCA correctly prints the least common ancestor of u and v for each pair $\{u, v\} \in P$.

6.D. (30 PTS.) Analyze the running time of LCA, assuming that we use the implementation of the disjoint-set data structure with path compression and union by rank.