Required Problems

1. [10 Points]

   (a) [1 Points] 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?

   (b) [1 Points] Same question, for rank $\lfloor (\lg n)/2 \rfloor$.

   (c) [2 Points] Prove that in a set of $n$ elements, a sequence of $n$ consecutive FIND operations take $O(n)$ time in total.

   (d) [1 Points] (Based on CLRS 21.3-2)
   Write a nonrecursive version of FIND with path compression.

   (e) [3 Points] 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.

   (f) [2 Points] What happens in the same situation if only the path compression is used?

2. [10 Points] Off-line Minimum

   (Based on CLRS 21-1)
   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 $i$th 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.
(a) [2 Points]
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:

\[4, 8, E, 3, E, 9, 2, 6, E, E, 1, 7, E, 5.\]

Fill in the correct values in the extracted array.

(b) [4 Points]
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.

\begin{verbatim}
Off-Line-Minimum(m,n)
  1  for i ← 1 to n
  2      do determine j such that i ∈ K_j
  3      if j ≠ m + 1
  4          then extracted[j] ← i
  5          let l be the smallest value greater than j for which set K_l exists
  6          K_l ← K_j ∪ K_l, destroying K_j
  7  return extracted
\end{verbatim}

Argue that the array \(extracted\) returned by Off-Line-Minimum is correct.

(c) [4 Points]
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.

3. [10 Points] Tarjan’s Off-Line Least-Common-Ancestors Algorithm
(Based on CLRS 21-3)
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   \textit{ancestor}[\textit{FIND}(u)] \leftarrow u
3   \textbf{for} each child $v$ of $u$ in $T$
4       \textbf{do} LCA($v$)
5   UNION($u$, $v$)
6   \textit{ancestor}[\textit{FIND}(u)] \leftarrow u
7   \textit{color}[$u$] \leftarrow \textsc{black}
8   \textbf{for} each node $v$ such that $\{u, v\} \in P$
9       \textbf{do if} \textit{color}[$v$] = \textsc{black}
10      \textbf{then} print “The least common ancestor of” $u$ “and” $v$ “is”\textit{ancestor}[\textit{FIND}(v)]

(a) \textbf{[2 Points]} Argue that line 10 is executed exactly once for each pair $\{u, v\} \in P$.

(b) \textbf{[2 Points]} 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 $T$.

(c) \textbf{[3 Points]} Prove that LCA correctly prints the least common ancestor of $u$ and $v$ for each pair $\{u, v\} \in P$.

(d) \textbf{[3 Points]} 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.

4. \textbf{[10 Points]} Ackermann’s Function

The Ackermann’s function $A_i(n)$ is defined as follows:

$$A_i(n) = \begin{cases} 
2 & \text{if } n = 1 \\
2n & \text{if } i = 1 \\
A_{i-1}(A_i(n-1)) & \text{otherwise}
\end{cases}$$

Here we define $A(x) = A_2(x)$. And we define $\alpha(n)$ as a pseudo-inverse function of $A(x)$. That is, $\alpha(n)$ is the least $x$ such that $n \leq A(x)$.

(a) \textbf{[2 Points]} Give a precise description of what are the functions: $A_2(n)$, $A_3(n)$, and $A_4(n)$.

(b) \textbf{[3 Points]} \textbf{Prove} that $\lim_{n \rightarrow \infty} \frac{\alpha(n)}{\log^*(n)} = 0$.

(c) \textbf{[3 Points]} We define

$$\log^{**} n = \min \left\{ i \geq 1 \left| \underbrace{\log^* \ldots \log^*}_i n \leq 2 \right. \right\}$$

(i.e., how many times do you have to take $\log^*$ of a number before you get a number smaller than 2). \textbf{Prove} that $\lim_{n \rightarrow \infty} \frac{\alpha(n)}{\log^{**}(n)} = 0$.

(d) \textbf{[2 Points]} \textbf{Prove} that $\log (\alpha(n)) \leq \alpha(\log^* n)$ for $n$ large enough.