Exercise 1: Amortized Analysis (10 Points)

Consider a binary min-heap data structure that supports the two operations \texttt{insert} and \texttt{delete-min}. The heap is initially empty and we assume that its number of elements never exceeds \( n \).

(a) Use the \textit{accounting} method to show that we can consider the amortized cost of \texttt{insert} to be \( O(\log n) \) and the amortized cost of \texttt{delete-min} to be \( O(1) \). (3 Points)

(b) Show the statement from part (a), this time using the \textit{potential function} method. (5 Points)

(c) We would like to amortize the costs differently such that the amortized cost of \texttt{insert} is \( O(1) \) and the amortized cost of \texttt{delete-min} is \( O(\log n) \). Either define a feasible \textit{potential function} that yields these amortized costs or argue why this is not possible. (2 Points)

Exercise 2: Union Find - Linked List Implementation (8 Points)

In the lecture, we have seen a linked list implementation where each linked list has a pointer to the first and last element. Describe an alternative implementation that uses only one of these pointers. Your scheme should still allow for the union-by-size heuristic and should not increase the asymptotic running time of the operations.

Exercise 3: Union Find - Disjoint-Set Forests (8 Points)

(a) Give a sequence of \( m \) \texttt{make-set}, \texttt{union}, and \texttt{find} operations, \( n \) of which are \texttt{make-set} operations, that takes \( \Omega(m \log n) \) time when we use union by rank only. (3 Points)

(b) Suppose that we wish to add the operation \texttt{print-set}, which is given a node \( x \) and prints all the members of \( x \)'s set, in any order. Show how to add this feature to the disjoint-set forest implementation such that \texttt{print-set} takes time linear in the number of members of \( x \)'s set and the asymptotic running times of the other operations are unchanged. Assume that we can print each member of the set in \( O(1) \) time. (5 Points)