Problem Set 2

Data Structures and Algorithms, Fall 2021

Due: September 16 23:59:59 (UTC+8), mail to `dsalg21ps@chaodong.me`.

Problem 1

(a) Give a $\Theta(n)$-time non-recursive procedure that reverses a singly linked list of n elements. The procedure should use no more than constant space beyond that needed for the list itself.

(b) Explain how to implement doubly linked lists using only one pointer value $x.np$ per item instead of the usual two (i.e., $next$ and $prev$). Assume that all pointer values can be interpreted as k-bit integers, and define $x.np$ to be $x.next \oplus x.prev$, the k-bit "exclusive-or" (i.e., XOR) of $x.next$ and $x.prev$. (The value $NULL$ is represented by 0.) Be sure to describe what information you need to access the head of the list. Give the pseudocode for the $\texttt{Insert}(x, i)$ (i.e., insert element x at position i) and $\texttt{Delete}(i)$ (i.e., delete the i^{th} element in the list) operations.

Problem 2

Design a MAXSTACK data structure that can store comparable elements and supports stack operations `push(x)`, `pop()`, as well as the `max()` operation, which returns the maximum value currently stored in the data structure. All operations should run in $O(1)$ time in your implementation. (You may assume there exists a STACK data structure which supports `push` and `pop`, and both operations run in $\Theta(1)$ time.) You should give a brief overview of your MAXSTACK data structure, then provide pseudocode for each of the three operations. You should also discuss the space complexity of your implementation.

Problem 3

You are given an infix expression in which each operator is in $\{+, \times, !\}$ and each operand is a single digit positive integer. Write an algorithm to convert it to postfix. (For example, given "$1 + 2 \times 3!$", whose value is 13, your algorithm should output "$123! \times +$".) You should give a brief overview of your algorithm, then provide pseudocode, and finally discuss its time complexity. (To get full credit, your algorithm should have $O(n)$ time complexity.)

Problem 4

Suppose you are choosing between the following three algorithms:

- Algorithm A solves problems of size n by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in $O(n)$ time.
- Algorithm B solves problems of size n by recursively solving two subproblems of size $n - 1$ and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in $O(\cdot)$ notation, try to be as tight as possible), and which would you choose? Explain your answer.

Problem 5

You are given an array of n integers, and some of the elements in the array are duplicates; that is, they appear more than once in the array. Show how to remove all duplicates from the array in time $O(n \log n)$. Specifically, first give an overview of your algorithm, then provide the pseudocode, and conclude with an analysis on running time. Remember also to briefly argue the correctness of your algorithm.

Problem 6

Let A be an array containing n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an *inversion* of A.

(a) List all inversions of the array $\langle 2, 3, 8, 6, 1 \rangle$.

(b) What is the relationship between the running time of insertion sort and the number of inversions in the input array? To get full credit, prove your answer is correct.

(c) Give an $O(n \lg n)$ time algorithm that can count the number of inversions of a size n array. Your algorithm does not need to list all inversions. You do *not* need to prove your algorithm is correct. *(Hint: modify the mergesort algorithm.)*