STACKS, QUEUES, LINKED LISTS & RECURSION, AMORTIZED ANALYSIS

(1) Describe a recursive algorithm to compute the integer part of the base-two logarithm of n using only addition and integer division.

(2) Suppose you are given an n-element array A containing distinct integers that are listed in increasing order. Given a number k, describe a recursive algorithm the find two integers in A that sum to k, if such a pair exists. What is the running time of your algorithm?

(3) Describe a linear time algorithm to reverse a singly-linked list L so that the ordering of the nodes becomes opposite of what it was before.

(4) Describe in pseudo-code how to implement the stack ADT using two queues. What is the running time of push(), and pop() methods in this case? (Note: there are multiple solutions to this. Think of a solution where push() is constant time, and another solution where pop() is constant time).

(5) A (singly linked) circular list is a collection C of n positions such that each has a `next` variable and following `next` links starting from any position can visit all positions in C. Describe how to perform `insertBefore`(p, e) and `insertAfter`(p, e) for position p and element e in such a scheme. What are the running times of these operations? Can you do both operations in constant time?

(6) Consider the problem of storing a very large counter. Say, we use an array A where i^{th} bit of the count is stored $A[i]$. Suppose the counter had to count a total of N events. What is the total running time of counting up to N using a such a counter? (Hint: think amortized analysis).

(7) [Challenge Problem] A singly linked list L has a cycle if the last position of the list, instead of having a null `next` pointer, points to some previous position in the list. Develop a method for checking if L has a cycle.