The questions on this problem set are due to Jeff Erickson: `http://www.cs.illinois.edu/~jeffe/teaching/algorithms`.

Question 1. Suppose we are maintaining a data structure under a series of n operations. Let $f(k)$ denote the actual running time of the kth operation. For each of the following functions f, determine the resulting amortized cost of a single operation.

1. $f(k)$ is the largest integer i such that 2^i divides k.
2. $f(k) = k$ if k is a power of 2, and $f(k) = 1$ otherwise.
3. $f(k) = k$ is k is a Fibonacci number, and $f(k) = 1$ otherwise.
4. Let T be a *complete* binary search tree, storing the integer keys 1 through n. $f(k)$ is the number of ancestors of node k.

Question 2. An *extendable array* is a data structure that stores a sequence of items and supports the following operations:

- ADDTOFRONT(x) adds x to the *beginning* of the sequence.
- ADDTOEND(x) adds x to the *end* of the sequence.
- LOOKUP(k) returns the kth item in the sequence, or NULL if the current length of the sequence is less than k.

Describe and analyze a *simple* data structure that implements an extendable array. Your ADDTOFRONT and ADDTOBACK algorithms should take $O(1)$ *amortized* time, and your LOOKUP algorithm should take $O(1)$ *worst-case* time. The data structure should use $O(n)$ space, where n is the *current* length of the sequence.

Question 3. Describe how to implement a queue using two stacks and $O(1)$ additional memory, so that the amortized time for any enqueue or dequeue operation is $O(1)$. The *only* access you have to the stacks is through the standard methods PUSH and POP.

Question 4. Suppose we can insert or delete an element into a hash table in $O(1)$ time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:

- After an insertion, if the table is more than $3/4$ full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
- After a deletion, if the table is less than $1/4$ full, we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

Show that for any sequence of insertions and deletions, the amortized time per operation is still $O(1)$.

Hint: Do not use potential functions.