Question 1 (25 points).  
Consider the following sequence of requests for the Bin Packing problem, for \( N > 0 \):

- \( N \) “small” items, each of size \( \frac{1}{6} - 2\varepsilon \); followed by
- \( N \) “mid-size” items, each of size \( \frac{1}{3} + \varepsilon \); followed by
- \( N \) “large” items, each of size \( \frac{1}{2} + \varepsilon \).

(a) [5 points] Let \( A \) be an online algorithm, and consider its sequence of allocations on the above sequence of requests. If \( A \) has a competitive ratio at most 3/2, how many bins may be occupied after the first \( N \) (small) items have been placed? How many mid-size items can thereafter be placed in these same bins (rather than in new ones)? How many bins may be occupied after \( 2N \) (small and mid-size) items have been placed?

(b) [10 points] Describe a complete schedule to place each of the \( 3N \) items, and show that it achieves a competitive ratio of 3/2 on the sequence.

(c) [10 points] Show that the previous part is optimal: no online algorithm can achieve a competitive ratio \( \rho < 3/2 \) on this input sequence.

(Problem adapted from Weiss.)

Question 2 (20 points).
Consider the Caching problem, with the cache size fixed at \( k \). For parts (a) and (b), we consider the request sequence that consists of access to \( k + 1 \) items, in “round robin” fashion; that is, the sequence of items requested is \( 0, 1, 2, \ldots, k, 0, 1, 2, \ldots, k, 0, \ldots \). (Note that this is a worst-case sequence for LRU.)

(a) [5 points] Suppose that the Randomized Marking Algorithm is used (as described in lecture; also see Kleinberg and Tardos, §13.8, 750–758).

- Describe the sequence of phases. In each phase after the first, how many requested items are fresh, and how many are stale?
- What is the probability that the \( i \)th request of a phase produces a miss?
- What is the expected ratio of misses per phase, compared to the offline minimum?
(b) [10 points] Now consider the Uniformly Random Eviction (URE) policy: at each miss, the item to evict is selected uniformly at random from all of the items in the cache.

- Explain why URE is not a marking algorithm (and thus the analysis given in class, and the text, does not apply).
- Show that after each item has been requested at least once, each further request for an item, in the sequence given, has the same probability of a miss. Estimate this probability (you may assume that \( k \) is large).
- What is the (approximate) expected ratio of misses per phase, compared to the offline minimum? (Hint: use the linearity of expectation.)
- Qualitatively, does URE behave like RMA, like LRU, or neither, on this request sequence?

(c) [5 points] Describe how the analyses in parts (a) and (b) change if the request sequence alternates between requesting in increasing order and requesting in decreasing order. That is, the sequence of requests is 0, 1, 2, ..., \( k \), \( k-1 \), \( k-2 \), ..., 1, 0, 1, 2, ..., \( k \), \( k-1 \), .... (In \( 2k \) consecutive requests, items 0 and \( k \) are requested once each and each other item twice.) You do NOT need exact calculations; do enough to support a qualitative description.

**Question 3** (15 points).

Recall that a NAND tree with \( n = 2^k \) leaves is a formula \( f_k(x_1, x_2, \ldots, x_n) \) that can be described recursively as follows. For any \( k \geq 0 \),

\[
\begin{align*}
    f_0(x_1) &= x_1 \\
    f_{k+1}(x_1, \ldots, x_{2^{k+1}}) &= \text{NAND}(f_k(x_1, \ldots, x_{2^k}), f_k(x_{2^k+1}, \ldots, x_{2^{2k}})),
\end{align*}
\]

where \( \text{NAND}(a, b) = \neg(a \land b) \).

In class, we saw that there is a randomized query algorithm that solves the binary NAND-tree evaluation problem on any input with an expected number of queries that is \( O(n^d) \), where \( d = \log_2\left(\frac{1 + \sqrt{33}}{4}\right) \approx 0.743 \).

Here we show that, for any deterministic algorithm, there exists an input that requires \( n \) queries to solve. We show that by proving that, for any deterministic query algorithm, there exists an **evasive** sequence of length \( n-1 \), which is defined as a sequence of responses to the first \( n-1 \) queries such that, after the \( n-1 \) queries have been made, the answer to the problem critically depends on the remaining bit that has not yet been queried. Proving this implies that the deterministic query complexity of binary NAND-tree evaluation is \( n \).

(a) [5 points] Here we make some observations about \( f_3(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) \). Consider the query algorithm whose sequence of queries is \( x_1, x_2, x_3, \ldots \) (in order), where it stops after enough information is obtained to evaluate \( f_3 \). Note if the results for the first three queries \( x_1, x_2, x_3 \) are all 0, then \( f_3 \) must be 1 and the algorithm stops after making only 3 queries.

- Give an evasive sequence for the first 7 queries \( x_1, x_2, x_3, x_4, x_5, x_6, x_7 \).
- Give an evasive sequence for an algorithm whose first 7 queries are \( x_6, x_4, x_2, x_7, x_5, x_3, x_1 \).

(b) [10 points] Prove that, for all \( k \), the deterministic query complexity of \( f_k(x_1, x_2, \ldots, x_n) \) is \( n \). In other words, for every deterministic algorithm, there exists an input on which it must query all \( x_i \)'s top get the right answer.

Please consider attempting to prove this without help. But if you get stuck then you might benefit by looking at the hint on the next page. (Hint page present only on the on-line version.)
(Hint: prove by induction on $k$ that, for any query algorithm solving $f_k(x_1, x_2, \ldots, x_n)$, there exists an evasive sequence of length $n - 1$.)