CSE 102
Final Exam Review
Solutions to Selected Problems

1. Suppose $T(n)$ satisfies the recurrence $T(n) = 3T(n/4) + F(n)$, where $F(n)$ itself satisfies the recurrence $F(n) = 5F(n/9) + n^{3/4}$. Find a tight asymptotic bound for $T(n)$. Be sure to fully justify each use of the Master Theorem. (Hint: $\log_9(5) < 3/4 < \log_4(3)$.)

   Solution:
   First observe

   $$5^2 = 25 < 27 = 3^3 \;\Rightarrow\; 5 < 3^{3/2} = 9^{3/4} \;\Rightarrow\; \log_9(5) < 3/4$$

   and

   $$2^3 = 8 < 9 = 3^2 \;\Rightarrow\; 4^{3/4} = 2^{3/2} < 3 \;\Rightarrow\; 3/4 < \log_4(3).$$

   To determine $F(n)$, let $\epsilon = \frac{3}{4} - \log_9(5)$. Then $\epsilon > 0$ and $\frac{3}{4} = \log_9(5) + \epsilon$, giving $n^{3/4} = \Omega(n^{3/4}) = \Omega(n^{\log_9(5)+\epsilon})$. Also, for any c in the range $\frac{5}{9^{3/4}} \le c < 1$ we have

   $$5\left(\frac{n}{9}\right)^{3/4} = \frac{5}{9^{3/4}} \cdot n^{3/4} \le cn^{3/4}$$

   for all $n \ge 1$, establishing the regularity condition. Case 3 yields $F(n) = \Theta(n^{3/4})$.

   We may now write the recurrence for $T(n)$ as $T(n) = 3T(n/4) + n^{3/4}$. Let $\epsilon = \log_4(3) - \frac{3}{4}$. It follows that $\epsilon > 0$ and $\frac{3}{4} = \log_4(3) - \epsilon$, whence $n^{3/4} = O(n^{3/4}) = O(n^{\log_4(3)-\epsilon})$. Case 1 of the Master Theorem gives us $T(n) = \Theta(n^{\log_4(3)})$. ∎

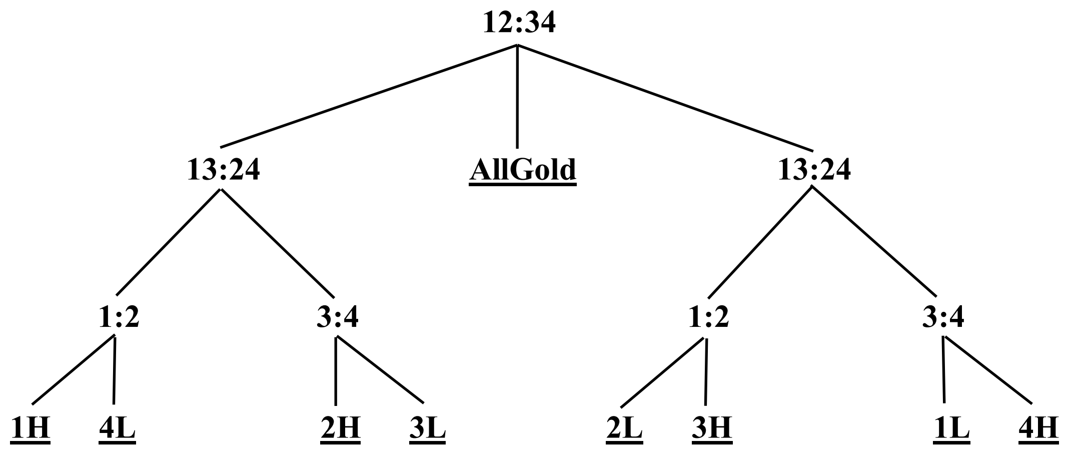
2. Suppose we are given 4 gold bars (labeled 1, 2, 3, 4), one of which *may* be counterfeit: gold-plated tin (lighter than gold) or gold-plated lead (heavier than gold). Again the problem is to determine which bar, if any, is counterfeit and what it is made of. The only tool at your disposal is a balance scale, each use of which produces one of three outcomes: tilt left, balance, or tilt right.

   a. Use a decision tree argument to prove that at least 2 weighings must be performed (in worst case) by any algorithm that solves this problem. Carefully enumerate the set of possible verdicts.

      Proof:
      Any algorithm for this problem can be represented by a ternary decision tree. Since there are 9 possible verdicts: {1L, 2L, 3L, 4L, 1H, 2H, 3H, 4H, AllGold}, at least $\lceil \log_3(9) \rceil = 2$ weighings are necessary. ∎

   b. Determine an algorithm that solves this problem using 3 weighings (in worst case). Express your algorithm as a decision tree.

      Solution:
      As usual a left branch indicates that the scale tilted left, right branch means tilt right, and middle branch means the scale balanced. There are of course many correct solutions, one of which is pictured below.



Note that although this is a ternary tree, only one node (the root) actually has 3 children. The question naturally arises as to whether this can be done with just 2 weighings, as the decision tree lower bound in (a) would suggest. Part (c) below shows this is not possible. ■

c. Find an adversary argument that proves 3 weighings are necessary (in worst case), and therefore the algorithm you found in (b) is best possible. (Hint: study the adversary argument for the min-max problem discussed in class to gain some insight into this problem. Further hint: put some marks on the 4 bars and design an adversary strategy that, on each weighing, removes the fewest possible marks, then show that if the balance scale is only used 2 times, not enough marks will be removed.)

Proof:
We assume that any algorithm for this problem always places an equal number of bars on the left and right sides of the scale, since the greater number of bars will always outweigh the lesser. When such an algorithm is run against the following adversary, the daemon first marks each bar with both a $+$ sign (meaning the bar is possibly heavy), and a $-$ sign (meaning it may be light). As the algorithm progresses, the daemon will remove certain marks. Each bar is therefore always in one of 4 states.

$\pm$:	a candidate both for heavy and light
$+$:	a candidate for heavy, but not light
$-$:	a candidate for light, but not heavy
N (no mark):	neither heavy nor light, known to be pure gold

Each answer that the daemon gives to a weighing (*left*, *balance* or *right*), implies that some marks should be erased. If for instance, a probe is answered by stating that the scale tilted left, then all $-$ signs on the left and all $+$ signs on the right must be deleted. Tilting left also implies that all bars not on the scale are genuine, and hence their marks should be erased as well. If the daemon says the scale tilted right, then all $+$ signs on the left, all $-$ signs on the right, and any marks on bars not on the scale should be removed. If the daemon says that the scale balances, then all bars on the scale are genuine, so their marks must be erased.

The daemon's strategy is to always answer in such a way that the minimum number of marks are removed. Specifically, let $L, R \subseteq \{1, 2, 3, 4\}$ denote the sets of bars placed on the left and right sides of the scale, respectively. Define

$$L_+ = \# \text{ of} + \text{marks on the left side of the scale}$$
$$L_- = \# \text{ of} - \text{marks on the left side of the scale}$$

$$\begin{aligned}
R_+ &= \text{ \# of} + \text{marks on the right side of the scale}\\
R_- &= \text{ \# of} - \text{marks on the right side of the scale}\\
a &= L_+ + R_-\\
b &= L_- + R_+\\
c &= \text{\# of marks on bars not placed on the scale}
\end{aligned}$$

Thus the daemon's strategy is to find $\min(a+b, a+c, b+c)$, remove that number of marks and answer accordingly. Observe that both $a+b \le a+c$ and $a+b \le b+c$ are true if and only if $b \le c$ and $a \le c$, which is equivalent to $c = \max(a,b,c)$. Similarly, $a+c \le a+b$ and $a+c \le b+c$ are true iff $b = \max(a,b,c)$, and also $b+c \le a+b$ and $b+c \le a+c$ hold iff $a = \max(a,b,c)$. When the algorithm places bars on the scale, the daemon executes the following algorithm.

```
Weigh(L, R)     pre: $|L| == |R|$
1.  compute the quantities $a$, $b$ and $c$ defined above
2.  $m = \max(a, b, c)$
3.  if $a == m$
4.          remove $b + c$ marks: $-$ on left, $+$ on right, all marks not on the scale
5.          return "left"
6.  else if $b == m$
7.          remove $a + c$ marks: $+$ on left, $-$ on right, all marks not on the scale
8.          return "right"
9.  else  // $c == m$
10.         remove $a + b$ marks: all marks on the scale
11.         return "balance"
```

Now suppose the algorithm halts and returns a verdict after using the scale at most 2 times. Since there are only 4 bars, and since $|L| = |R|$, only two kinds of weighings can take place: 2 bars on each side, or 1 bar on each side. Since the scale is used two times, we have 4 cases to consider, each with a number of subcases.

	1st weighing	2nd weighing
(1)	2 bars on each side	2 bars on each side
(2)	2 bars on each side	1 bar on each side
(3)	1 bar on each side	2 bars on each side
(4)	1 bar on each side	1 bar on each side

Case (1): From the initial set of states $\{\pm, \pm, \pm, \pm\}$, the first weighing removes 2 +'s and 2 −'s, leaving the states $\{+, +, -, -\}$. The second weighing breaks into two subcases.

(1.1) Place $+\,+$ on one side and $-\,-$ on the other, symbolized by $+\,+\mid-\,-$. Note it does not matter whether the algorithm places $+\,+$ on the left and $-\,-$ on the right, or the other way around, since the daemon's strategy removes the same number of marks (namely 0) in both cases. The daemon simply tilts in the direction that confirms the existing marks, leaving the bars in states $\{+, +, -, -\}$.

(1.2) Place $+\,-\mid+\,-$. In this case we have $a = b = 2$ and $c = 0$, so the daemon removes 2 marks leaving $\{+, -, N, N\}$.

Case (2): Again the initial states $\{\pm, \pm, \pm, \pm\}$ become $\{+, +, -, -\}$ after the first weighing, giving 3 subcases for the second.

(2.1) Place $+\mid+$. In this case $a = 1$, $b = 1$ and $c = 2$, so the daemon returns "balance" and removes 2 marks, leaving $\{-, -, N, N\}$.

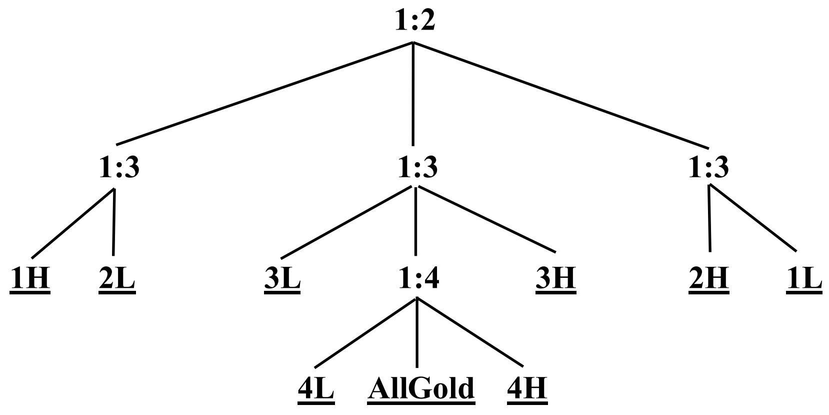
(2.2) Place $- \mid -$. Again the daemon removes 2 marks and leaves $\{+,+,N,N\}$.
(2.3) Place $+ \mid -$. Recall the notation $+| -$ does not specify which side has the $+$ and which has the $-$, since the same number of marks are removed. In this case the daemon removes 2 marks, leaving the states $\{+,-,N,N\}$.

Case (3): After the first weighing with 1 bar on each side, the initial states $\{\pm,\pm,\pm,\pm\}$ become $\{\pm,\pm,N,N\}$, leaving 2 subcases:
(3.1) Place $\pm\pm \mid NN$. In this case $a=b=2$ and $c=0$, so the daemon removes 2 marks leaving either $\{+,+,N,N\}$ or $\{-,-,N,N\}$.
(3.2) Place $\pm N \mid \pm N$. Again 2 marks are removed leaving $\{+,-,N,N\}$.

Case (4): The first weighing again transforms initial states $\{\pm,\pm,\pm,\pm\}$ into $\{\pm,\pm,N,N\}$, leaving 3 subcases:
(4.1) Place $\pm \mid \pm$. The daemon removes 2 marks leaving $\{+,-,N,N\}$.
(4.2) Place $\pm \mid N$. The daemon returns "balance", and erases $\pm$ on one bar leaving $\{\pm,N,N,N\}$.
(4.3) Place $N \mid N$. The daemon again returns "balance, and removes 0 marks leaving the bars in states $\{\pm,\pm,N,N\}$.

Notice that in all cases at least two marks remain. In most cases, these marks are on different bars, but in subcase (4.2) they are on the same bar. In any case other than (4.2), if the algorithm says a particular bar is heavy or light, the daemon can produce another valid candidate as the counterfeit bar. In case (4.2), if the algorithm identifies the bar marked $\pm$ as say heavy, the daemon can claim that bar to be light. A similar contradiction arises if the algorithm identifies the $\pm$ bar as light. In all cases then, the daemon has a valid contradiction to the algorithm's verdict, and therefore, no correct algorithm can perform fewer than 3 weighings. ■

Remark The case by case analysis above shows that it *is* possible to identify the counterfeit bar in just 2 weighings by designing an algorithm that utilizes case (4.2). To decide if the counterfeit is heavy or light still takes one more weighing though. The algorithm represented by the decision tree below utilizes this strategy.



Which algorithm one chooses may depend on prior beliefs as to which bars are good or bad. For instance, if you think it is highly likely that a certain bar is genuine, then calling that bar number 4, the above algorithm will produce a good average case run time. If you believe it likely that there is no counterfeit bar, then the solution given in (b) is preferred. The following exercise should be the last word on the bar weighing problem.

Additional Exercise
Suppose we are given $n \geq 3$ gold bars, where again one bar may be counterfeit, light or heavy. Again we have a balance scale whose use produces one of three outcomes. Since there are $2n+1$ possible verdicts, we immediately have the decision tree lower bound of $\lceil \log_3(2n+1) \rceil$ for the minimum number of weighings to find the counterfeit and its type (light or heavy), or show that there is no counterfeit.

a. Give an adversary argument showing that $\lceil \log_3(2n+3) \rceil$ weighings are necessary in general. (Observe that the two lower bounds are different only when $n = \frac{1}{2}(3^k - 1)$ for some $k \geq 2$.)
b. Design an algorithm that solves this problem using the scale at most $\lceil \log_3(2n+3) \rceil$ times.

3. Recall the coin changing problem again. Given denominations $d = (d_1, d_2, \dots, d_n)$ and an amount N, determine the number of coins in each denomination necessary to disburse N units using the fewest possible coins. Assume that there is an unlimited supply of coins in each denomination. Prove that the greedy strategy works for any amount N with the coin system $d = (1, 5, 10, 25)$.

Proof:
Let $N \geq 0$, let $x = (x_1, x_2, x_3, x_4)$ be the optimal solution, and let $g = (g_1, g_2, g_3, g_4)$ be the solution obtained by the greedy strategy. We will show that $x = g$, whence g is also optimal. Since both solutions disburse the amount N, we have

(1) $$x_1 + 5x_2 + 10x_3 + 25x_4 = N = g_1 + 5g_2 + 10g_3 + 25g_4,$$

which implies $x_1 \equiv g_1 (mod \;\; 5)$. Observe that $0 \leq x_1 < 5$ since otherwise it would be possible to replace 5 coins of type 1 (pennies) by 1 coin of type 2 (nickel), contradicting that x is an optimal solution. Likewise $0 \leq g_1 < 5$, since the greedy strategy guarantees that as many nickels as possible will be disbursed before any pennies are. These inequalities, along with the preceding congruence, imply $x_1 = g_1$.

Cancel x_1 from (1) and divide through by 5 to get $x_2 + 2x_3 + 5x_4 = g_2 + 2g_3 + 5g_4$, which we write as

(2) $$(x_2 - g_2) + 2(x_3 - g_3) + 5(x_4 - g_4) = 0.$$

The fact that x is optimal means $\sum_{i=1}^{4} x_i \leq \sum_{i=1}^{4} g_i$, and since $x_1 = g_1$ we have

(3) $$\sum_{i=2}^{4}(x_i - g_i) \leq 0$$

Note that $0 \leq x_2 < 2$ since any two nickels can be replaced by one dime, and $0 \leq g_2 < 2$ since the greedy strategy disburses the maximum number of dimes before any nickels. Therefore

(4) $$-1 \leq (x_2 - g_2) \leq 1,$$

Our goal is to show $x_2 = g_2$. Assume, to get a contradiction, that $x_2 \neq g_2$, so that $x_2 - g_2 \neq 0$. Inequality (4) now implies $x_2 - g_2 = \pm 1$. Now the very first action of the greedy strategy is to pay as many quarters as possible. Thus $g_4 \geq x_4$, and hence $g_4 - x_4 \geq 0$. If it were the case that $g_4 - x_4 = 0$, then (2) would imply $2(x_3 - g_3) = \pm 1$, which is impossible since the left side is even while the right is odd. Therefore we conclude

(5) $$(g_4 - x_4) \geq 1.$$

Equation (2) implies $(x_3 - g_3) = \frac{1}{2}(g_2 - x_2) + \frac{5}{2}(g_4 - x_4)$. Combining this with inequality (3) gives

$$\begin{aligned} 0 &\geq (x_2 - g_2) + (x_3 - g_3) + (x_4 - g_4) \\ &= (x_2 - g_2) + \left[\tfrac{1}{2}(g_2 - x_2) + \tfrac{5}{2}(g_4 - x_4)\right] + (x_4 - g_4) \\ &= \tfrac{1}{2}(x_2 - g_2) + \tfrac{3}{2}(g_4 - x_4) \\ &\geq \ -\tfrac{1}{2} \ + \ \tfrac{3}{2} \qquad \text{(by inequalities (4) and (5))} \\ &= \ 1, \end{aligned}$$

i.e. $1 \leq 0$, which is absurd. Therefore our assumption was false, and hence $x_2 = g_2$.

Now equation (2) reduces to $2(x_3 - g_3) + 5(x_4 - g_4) = 0$, which we write as

(6) $$x_3 - g_3 = \frac{5}{2}(g_4 - x_4).$$

Inequality (3), together with $x_2 = g_2$, and equation (6) implies

$$\begin{aligned} 0 &\geq (x_3 - g_3) + (x_4 - g_4) \\ &= \tfrac{5}{2}(g_4 - x_4) + (x_4 - g_4) \\ &= \tfrac{3}{2}(g_4 - x_4). \end{aligned}$$

Therefore $g_4 - x_4 \leq 0$, so that $g_4 \leq x_4$. But as noted previously $g_4 \geq x_4$ (no solution disburses more quarters than the greedy solution.) Hence $g_4 = x_4$, and equation (6) now gives $g_3 = x_3$. This completes the proof that $x = g$. We conclude that the greedy strategy is optimal for this denomination set. ■

4. **Scheduling to Minimize Average Completion Time:** (This is problem 16-2a on page 402 of CLRS.) Suppose you are given a set $S = \{a_1, a_2, \ldots, a_n\}$ of tasks, where task a_i requires p_i units of processing time to complete, once it has started. You have one computer on which to run these tasks, and the computer can run only one task at a time. Let c_i be the *completion time* of task a_i, that is, the time at which task a_i completes processing. Your goal is to minimize the average completion time, that is to minimize the quantity $(1/n)\sum_{i=1}^{n} c_i$. For example, suppose there are two tasks, a_1 and a_2, with $p_1 = 3$ and $p_2 = 5$, and consider the schedule in which a_2 runs first, followed by a_1. Then $c_2 = 5$, $c_1 = 8$, and the average completion time is $(5 + 8)/2 = 6.5$.

   Give an algorithm that schedules the tasks so as to minimize the average completion time. Each task must run non-preemptively, that is, once task a_i is started, it must run continuously for p_i units of time. Prove that your algorithm minimizes the average completion time, and state the running time of your algorithm.

   Solution:
   We adopt the following greedy strategy: sort S by increasing processing times p_i, schedule those tasks with shortest processing times first, and those with longer processing times later. To put it another way, determine a permutation of the indices (1, 2,…, n) which sorts the array of processing times $(p_1, \ldots, p_n)$, then apply that same permutation to the array of tasks $(a_1, \ldots, a_n)$ to obtain an optimal schedule. The running time of this algorithm is obviously the running time of the sort used, which can be chosen to be $\Theta(n\,log(n))$ in worst and average cases.

Theorem
The above greedy strategy results in a schedule which minimizes the average completion time of the tasks $a_1, a_2, \dots, a_n$.

Proof:
To simplify notation, let us assume from the start that the tasks $a_1, a_2, \dots, a_n$ are already sorted by increasing processing times, i.e we assume that $p_1 \le p_2 \le \cdots \le p_n$. Let us also assume that processing begins at time 0, so that the completion times are:

$$\begin{aligned} c_1 &= p_1 \\ c_2 &= p_1 + p_2 \\ &\vdots \\ c_n &= p_1 + p_2 + \cdots + p_n \end{aligned}$$

and in general $c_k = \sum_{i=1}^{k} p_i$. Thus the average completion time resulting from the above strategy is

$$A = (1/n)\sum_{k=1}^{n} c_k = (1/n)\sum_{k=1}^{n}\sum_{i=1}^{k} p_i = (1/n)\sum_{k=1}^{n}(n-k+1)p_k$$

More Generally if σ is any permutation of the set $\{1, 2, \dots, n\}$, then the average completion time of the schedule $a_{\sigma(1)}, a_{\sigma(2)}, \dots, a_{\sigma(n)}$ is given by $A(\sigma) = (1/n)\sum_{k=1}^{n}(n-k+1)p_{\sigma(k)}$. Given such a permutation σ, we must show that $A = A(\text{identity}) \le A(\sigma)$. Let i be the least index such that $\sigma(i) \ne i$. Since i is the smallest such index, we must have $i < \sigma(i)$. Let $j = \sigma(i)$ and $l = \sigma^{-1}(i)$, so that $\sigma(l) = i$. Let τ be the permutation obtained from σ by swapping i with j. In other words

$$\tau(k) = \begin{cases} i & \text{for } k = i \\ j & \text{for } k = l \\ \sigma(k) & \text{for all other } k \end{cases}$$

Thus

$$\begin{aligned} A(\sigma) - A(\tau) &= (1/n)\sum_{k=1}^{n}(n-k+1)p_{\sigma(k)} \quad - \quad (1/n)\sum_{k=1}^{n}(n-k+1)p_{\tau(k)} \\ &= (1/n)[(n-i+1)p_j + (n-l+1)p_i - (n-i+1)p_i - (n-l+1)p_j] \\ &= (1/n)[(l-i)p_j + (i-l)p_i] \\ &= (1/n)(l-i)(p_j - p_i) \end{aligned}$$

Now recall $i < \sigma(i) = j$ whence $p_i \le p_j$, and $\sigma(l) = i \ne l$ so that $i < l$. Hence $A(\sigma) - A(\tau) \ge 0$, and therefore $A(\sigma) \ge A(\tau)$. In other words the schedule $a_{\tau(1)}, a_{\tau(2)}, \dots, a_{\tau(n)}$ has a lower average completion time than does the schedule $a_{\sigma(1)}, a_{\sigma(2)}, \dots, a_{\sigma(n)}$. But the permutation τ has one more term in common with the identity permutation than σ does. That is $\tau(k) = k$ for $1 \le k \le i$, while $\sigma(k) = k$ for $1 \le k < i$ and $\sigma(i) \ne i$. If τ equals the identity permutation (i.e. $\tau(k) = k$ for all k), then we are done. If not, then repeat the above procedure with τ in place of σ to obtain a permutation τ_1 with one more term in common with the identity permutation than τ, and giving a schedule with lower average completion time. Continuing in this fashion we can construct a sequence of permutations $\sigma, \tau, \tau_1, \tau_2$, ...ending in the identity permutation giving schedules with descending average completion times: $A(\sigma) \ge A(\tau) \ge A(\tau_1) \ge A(\tau_2) \ge \cdots \ge A(\text{identity}) = A$. Thus $A(\sigma) \ge A$ as required. ■

5. Let $B = b_1 b_2 \dots b_n$ be a bit string of length n. Consider the following problem: determine whether or not B contains 3 consecutive 1's, i.e. whether B contains the substring "111". Consider algorithms that solve this problem whose only allowable operation is to peek at a bit.

   a. Suppose $n = 4$. Obviously 4 peeks are sufficient. Give an adversary argument showing that in general, 4 peeks are also necessary.

   Solution:
   Run any algorithm for this problem against the following adversary simulating a bit string of length 4. Daemon's strategy:

   - Answer 1 to any peek at a bit in the set $\{b_2, b_3\}$.
   - Answer 0 to the first peek at a bit in the set $\{b_1, b_4\}$.
   - Answer 1 to the second peek at a bit in the set $\{b_1, b_4\}$.

   Suppose the algorithm halts and returns a verdict (YES or NO) after doing at most 3 peeks. We may assume, without loss of generality, that the algorithm has performed *exactly* 3 peeks. There are two cases to consider.

   Case 1: Two bits from $\{b_2, b_3\}$ were peeked, and one bit from $\{b_1, b_4\}$ was peeked. In this case the revealed bits are either $011x$ or $x110$, where x indicates an unpeeked bit (and therefore unknown to the algorithm).

   Case 2: One bit from $\{b_2, b_3\}$ was peeked, and two bits from $\{b_1, b_4\}$ were peeked. In this case the revealed bits must be either $01x1$, $11x0$, $0x11$ or $1x10$, where again x indicates an unknown bit.

   In all cases the correct answer depends on the unknown bit. If the algorithm returns YES, the Daemon can claim x is a 0. If the algorithm returns NO, the Daemon can claim that x is 1. In each case, the Daemon's claim does not contradict any answer he has given, but does contradict the algorithm's output. Therefore no algorithm performing at most 3 peeks can be correct. ■

   Alternate Solution:
   Run the same adversary as above against any algorithm for this problem that performs only 3 peeks. We see that there are $4 \cdot 3 \cdot 2 = 24$ different orders in which an algorithm can peek at 3 bits from the set $\{b_1, b_2, b_3, b_4\}$. We exhaustively list all such orders, together with the corresponding revealed bit string containing one unknown bit x.

123	$011x$	312	$011x$
132	$011x$	321	$011x$
124	$01x1$	314	$0x11$
142	$01x1$	341	$1x10$
134	$0x11$	324	$x110$
143	$0x10$	342	$x110$
213	$011x$	412	$11x0$
231	$011x$	421	$11x0$
214	$01x1$	413	$1x10$
241	$11x0$	431	$1x10$
234	$x110$	423	$x110$
243	$x110$	432	$x110$

Observe again that in all cases, the correct answer depends on the unknown bit, so no algorithm that performs only 3 peeks can be correct. ■

b. Suppose $n \geq 5$. Give an adversary argument showing that $4 \cdot \lfloor n/5 \rfloor$ peeks are necessary. (Hint: divide B into $\lfloor n/5 \rfloor$ 4-bit blocks separated by 1-bit gaps between them. Thus bits 1-4 form the first block, and bit 5 is the first gap. Bits 6-9 form the next block and bit 10 is the next gap, etc.. Any leftover bits form a separate block. Now run the adversary from part (a) on each of the 4-bit blocks.)

Solution:
Run any algorithm for this problem against the following adversary simulating a bit string B of length $n \geq 5$.

Daemon's strategy:

- Divide B into $\lfloor n/5 \rfloor$ blocks of length 5, each consisting of 4 unknown bits followed by a 0, $(xxxx0)$, then followed by less than 5 leftover bits set to 0. Thus the Daemon puts

$$B = xxxx0 \;\; xxxx0 \;\; xxxx0 \;\; \cdots\cdots\cdots \;\; xxxx0 \;\; 0 \cdot\cdot 0$$

- Whenever a preset 0 bit is peeked, the Daemon answers 0.
- If any 3 bits from an unknown $xxxx$ section is peeked, the Daemon runs one of $\lfloor n/5 \rfloor$ independent copies of the adversary from part (a), and answers accordingly.
- If a 4th bit from from an unknown $xxxx$ section is peeked, the Daemon answers 0, which establishes that the given block does not contain the substring "111".

Now suppose the algorithm halts and returns a verdict (YES/NO) after performing fewer than $4 \cdot \lfloor n/5 \rfloor$ peeks. Since the Daemon begins with exactly $4 \cdot \lfloor n/5 \rfloor$ unknown x bits, at least 1 such bit has not been peeked. If the algorithm returns YES, the Daemon can claim that all unpeeked bits are 0. This yields a bit string not containing the substring "111". If the algorithm returns NO, the Daemon can claim the unpeeked bit is 1. This yields a bit string which does contain the substring "111". In each case, the Daemon's claim does not contradict any prior answer he has given, but does contradict the algorithm's output. Therefore no algorithm performing fewer than $4 \cdot \lfloor n/5 \rfloor$ peeks can be correct. ■