b2. The most convincing way to prove b is by an "adversary argument" -- an adversary tells you about values in the array, but doesn't decide what the values are until they are inspected or compared. (You may have done this in "20 questions" or "I spy:" keeping several possibilities that are consistent with your previous answers. See also http://jeffe.cs.illinois.edu/teaching/algorithms/notes/29-adversary.pdf)
Create an adversary that assigns $A[i] = i$ whenever some algorithm inspects $A[i]$. Suppose there exist $j \neq k$ where neither $A[j]$ nor $A[k]$ have been inspected. Show the algorithm cannot know the first peak.

~~Given that we know A[i] = i, the peak should occur at the last index. However, there~~

Depending on if $A[j] > A[k]$ or $A[j] < A[k]$, you can determine the peak. If $A[j] < A[k]$ and $j < k$, the first peak is at the end. If $A[j] > A[k]$, the first peak will be near j. We do not know, however, the position nor value of j or k, so we cannot determine where the first peak happens

c. Derive a method to identify a peak (not necessarily the first) in asymptotically faster than linear time (in little-$o(n)$ time). Hints that lead to two different methods: inspect heights in pairs to see which direction height is increasing, or note that from inspecting heights at four widely-spaced locations you can find an interval of three that must contain a peak. Hint2: Names for the two methods in the first hint are Binary search and Fibonacci search.

Look at four heights in the array at some equal interval. Compare the heights, If a height from either of the middle two indices is the greatest of the four, take the sampled indices on the left and right and repeat the sampling method. If the greatest height sampled is at the end of the array, take the section of indices between the end and the closest sample point and repeat the sample process again. The base case / always check for case is looking if the indices sampled are right next to eachother, if they are then there is a peak. This runs in lg(n) time.