1. You are given a list of real numbers of size $n$, and suppose you know that the entries are initially monotonically increasing until some position $p$ in the array, after which they are monotonically decreasing. The item at position $p$ is therefore the maximum of the list. Although the array is not sorted, it is possible to find the maximum in $O(\log n)$ using an approach similar to binary search. Propose such an algorithm and explain why it satisfies this time bound.

Modify your algorithm to the case where you are dealing with an $n$ by $n$ matrix of real values that has the same monotonicity property in both rows and columns. Your algorithm should have a running time that is asymptotically better than $O(n^2)$.

2. Construct a Huffman encoding over the alphabet \{a, b, c, d, e, g, h\} where the letter frequencies are: $f_a = 0.2$, $f_b = 0.1$, $f_c = 0.15$, $f_d = 0.07$, $f_e = 0.12$, $f_g = 0.3$, $f_h = 0.06$.

3. When we discussed big-O earlier in the semester we considered a recursive algorithm for computing $x^n$ when $n$ is a positive integer. It was based on the observation that $x^n = x^{n/2} \times x^{n/2}$ when $n$ is even, and $x^n = x^{n-1} \times x$ when $n$ is odd. What’s a tight big-O bound on the number of multiplication operations required in order to compute $x^n$ using this method? As a side-note, the same method can be used regardless of whether $x$ is a number or a string, in which case $\times$ is the string concatenation operator.