1. **(40 points)** Suppose you are given an array $A$ with $n$ entries, with each entry holding a distinct number. You are told that the sequence of values $A[1], A[2], \ldots, A[n]$ is **unimodal**: For some index $p$ between 1 and $n$, the values in the array entries increase up to position $p$ in $A$ and then decrease the remainder of the way until position $n$. (So if you were to draw a plot with the array position $j$ on the $x$-axis and the value of the entry $A[j]$ on the $y$-axis, the plotted points would rise until $x$-value $p$, where they’d achieve their maximum, and then fall from there on.)

You’d like to find the “peak entry” $p$ without having to read the entire array - in fact, by reading as few entries of $A$ as possible. Show how to find the entry $p$ by reading at most $O(\log n)$ entries of $A$.

Correct Algorithm: 30 pts
Runtime Analysis: 10 pts

2. **(45 points)** In this problem, we will look at a query system that is motivated by applications in wireless networks. If you are not interested in the application details, just skip the next paragraph and head straight to the formal description of the problem.

In particular, consider the scenario where there is a central node such that all the other sensor nodes can communicate directly with the central node. Each sensor node has a bit of information (e.g. “Is the temperature at my location > 70 degrees?”) The central node wants to compute some aggregate function over these bits: e.g. are there at least two sensor nodes with temperature greater than 70 degrees? The central node can “poll” multiple sensor nodes at once to see if their bits are one. Each sensor node replies with a positive back if it is polled and its bit is one. Else it remains silent. Now the central node can easily detect whether at least one of the sensor nodes it polled had it’s bit as one by just checking if some sensor node responded or not. Due to the nature of the wireless medium, it is very hard to count the number of responses (due to collision) but it is easy to check if at least one sensor node responded by just checking for “silence.” Now for computing any function, we want to minimize the number of polls as each poll needs a transmission, which in turn lower the battery life. The problem below talks about this scenario but only for “threshold” functions.

In this problem the input are $n$ bits $x_1, \ldots, x_n$. However, you can access the input using the following kind of queries. A **query** is a subset $S \subseteq \{1, \ldots, n\}$. The **answer** to a query $S$ is 0 if $\sum_{i \in S} x_i = 0$ and is 1 otherwise (i.e. if $\sum_{i \in S} x_i \geq 1$). Note that you have the full freedom to pick the query. So e.g. you can query all the bits one by one and have the full knowledge of all the bits $x_1, \ldots, x_n$. However, this means you will have to make $n$ queries, which is a lot. Your goal will be compute certain function using **as few queries as possible**.
In this problem, we will look at the \( t \)-threshold function \( f_t \) for some integer \( 1 \leq t \leq n \). In particular, \( f_t(x_1, \ldots, x_n) = 0 \) if \( \sum_{i=1}^{n} x_i < t \) and \( f_t(x_1, \ldots, x_n) = 1 \) otherwise. For example, let us consider the the 3-threshold function for \( n = 5 \). Note that \( f_3(1, 0, 0, 1, 0) = 0 \) and \( f_3(1, 0, 0, 1, 1) = 1 \).

Note that for any input \( x_1, \ldots, x_n \) and any \( 0 \leq t \leq n \), one can compute \( f_t(x_1, \ldots, x_n) \) in \( n \) queries. (In this case we will query \( \{i\} \) for every \( 1 \leq i \leq n \) and assign \( x_i = 1 \) if and only if the answer to \( \{i\} \) is 1. After all the \( n \) queries are done, we can decide \( f_t(x_1, \ldots, x_n) \) depending on whether \( \sum_{i=1}^{n} x_i \) (which we can now calculate as we know all the values of \( x_i \)) is < \( t \) or not.)

In this problem, you will design algorithms that decide the threshold functions using less than \( n \) queries.

(a) Here is an algorithm to compute the \( f_1 \) function with one query. Query the set \( \{1, \ldots, n\} \) and declare \( f_1(x_1, \ldots, x_n) \) to be 1 if and only if the answer to the query is 1.

Design a divide and conquer algorithm that can decide the function \( f_2 \) on any input \( x_1, \ldots, x_n \) using only \( O(\log n) \) queries. (You cannot do asymptotically better)

(b) For any \( 1 \leq k \leq n \), design an algorithm that can compute the function \( f_k \) on any input \( x_1, \ldots, x_n \) using only \( O(k \cdot \log (\frac{n}{k})) \) queries.

(Note: If you solve this part correctly, then you do not need to write up the solution for the previous part.)

Correct Algorithm for part a: 20 pts
Correct Algorithm for part b: 25 pts (45 pts if you skip part a)

3. (15 points) In the divide and conquer algorithm for finding the closest two points, we saw that if two points in \( S \) were < \( \delta \) apart then they could be at most 15 positions apart in \( S_y \). Come up with a number \( \alpha < 15 \) and prove that the result above still holds if one replaces 15 with \( \alpha \).

Note: To get any credit your proof must work for some \( \alpha \leq 12 \). To get full credit, your proof must work for some \( \alpha \leq 10 \).

Hint: It might be useful to define concretely what happens when a point is on one of the lines at \( x = x^* \), \( x = x^* - \delta \), and \( x = x^* + \delta \).

If your proof works for \( (\alpha = 12) \): 5 pts
If your proof works for \( (\alpha = 11) \): 10 pts
If your Proof works for \( (\alpha \leq 10) \): 15 pts