Question 1. (5 points) Heap mechanics
Unimportant...

Question 2. (10 points) A heaping challenge
Think of the heap as a binary tree and do an preorder traversal of the tree. Only go down a subtree if the current node has a key less than $x$ and the total number of nodes that have been visited that are less than $x$ is still smaller than $k$.
Hence you will visit a maximum of $2k$ nodes, if there are $k$ nodes less than $x$. Else you will visit an even smaller number of nodes.

Question 3. (10 points) Longest divisible subsequence
For each index, $i$, you can find the previous index, $j$, such that $a_j | a_i$ and choose the one with the largest divisible subsequence ending at that point, $j$. This means that for each index $i$ you have to do an $O(n)$ search giving a total of $O(n^2)$.

Question 4. (10 points) Potholes on the road
For each point $(i, j)$ you can find the path with the minimum weight to it as the minimum of the path from $(i - 1, j)$ or $(i, j - 1)$ plus the weight of the current cell. Hence for each cell you have to do constant amount of work giving a total of $O(n^2)$.

Question 5. (25 points) Summing up numbers
First notice that we can just replace $a_i$ by $b_i := a_i^3$, and now we need to answer the queries of the form $\sum_{i=l}^{r} b_i$.

1. (10 points) For each index $l$ you can calculate $f(l) = \sum_{i=1}^{l} b_i$, giving us that $CubeSum(l, r) = f(r) - f(l) + b_l$. We can build this array of $f$ values in linear time by keeping a running sum and incrementing on a new index.

2. (5 points) You can try out each possible pair of indices and choose the maximum in $O(n^2)$.

3. (10 points) For each index $r$, we will calculate the maximum value of CubeSum over all arrays ending at $r$, call it $v_r$.
Notice that $v_r = \max(v_{r-1} + b_r, b_r)$, as if $v_{r-1} > 0$, we can either extend the maximum array ending at $r-1$ and set $l_r = l_{r-1}$, or if $v_{r-1} < 0$, then we might as well discard it and set $l_r = r$.
Hence we can calculate the $v_r$ array in $O(n)$, and its easy to see that we can also keep track of the starting point, depending on where the maximum value is coming from.

Question 6. (15 points) Looking for stability
Look at the consecutive differences between numbers, $d_1 := a_2 - a_1, d_2 := a_3 - a_2, \ldots$. They will be of the form

$$1, -1, 0, -1, 0, -1, 1, 1, \ldots$$
For each index $r$ we will try to find the length of the maximum stable sequence ending at $r$. First we solve the problem for no 0’s in the above sequence.

As there are no 0’s, a stable sequence needs to be of the form $1, -1, 1, -1, 1, -1, \ldots$, i.e. the consecutive differences need to alternate between $-1$ and 1, else if there are two consecutive 1’s or $-1$’s then the difference between the minimum and maximum goes up to 2, and destroys the stability.

Hence for each point $r$, if $d_{r-1} = d_{r-2}$, the maximum sequence ending at $r$ has to start at $r - 1$, i.e. $l_r = r - 1$, else if $d_{r-1} = -1 \times d_{r-2}$ then $l_r = l_{r-1}$.

Now we can calculate this array in $O(n)$ and also see how to generalize it for having 0’s in the array.

**Question 7.** (20 points - Extra credit) Recursing on XOR

1. (10 points) Let us store the values in $DP[i][l]$, which stores $RXOR([a_i, \ldots, a_{i+l-1}])$.

   We can see that $DP[i][l] = DP[i][l-1] \oplus DP[i+1][l-1]$, using pascal’s triangle.

   Using this we can calculate it in $O(n^2)$.

2. (10 points) Notice that

   $RXOR_{max}([a_i, \ldots, a_r]) = \max(RXOR([a_i, \ldots, a_r]), RXOR_{max}([a_i, \ldots, a_{r-1}]), RXOR_{max}([a_{i+1}, \ldots, a_r]))$

   and use this to calculate it in $O(n^2)$. 