Question 1: A. BKR moves
Topic: Recursion.

With the constraints of this Task, it just becomes a pure brute force question, i.e. try out all the possible moves possible with recursion however the implementation for this though easy can be quite cumbersome.

The thing to keep in mind though is that you can get stuck in an infinite recursion. To take care of that, keep the number of moves left as a parameter of your recursive function and maintain a visited array that stores you have already visited some cell m moves will be the answer for that piece. (i, j) so you never go back to a visited cell already visited with lesser moves remaining. At the end of your m moves will be the answer for that piece. m moves will be the answer for that piece. (i, j) so you never go back to a visited cell already visited with lesser moves remaining. At the end of your m moves will be the answer for that piece.

Knight
Maintain a directional array
$dir = \{(1, 2), (1, -2), (-1, 2), (-1, -2), (2, 1), (2, -1), (-2, 1), (-2, -1)\}$
Now if the knight is at (i, j) then it can move to $(i + dir[k][1], j + dir[k][2]) \forall k \in [1, 8]$ if the new cell is unbroken. So the flow remains simple, Marks (i, j) as visited and then for all possible moves recursively call the function again for that cell if it is not broken and $m > 0$ and the cell is within the board.

Rook/Bishop
Rooks and Bishops are in two ways different from a knight. They cannot jump over obstacles and in one move they can can at most travel by 7 cells i.e. say a rook was at $(1, 1)$, in one move it can go to $(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), or (1, 8)$.
Keeping this in mind define moves for the Rook and bishop
Now if the rook is at (i, j) then it can move to $(i \pm k, j) \forall k \in [1, 7]$ or $(i, j \pm k) \forall k \in [1, 7]$
Similarly if the bishop is at (i, j) then it can move to $(i \pm k, j \pm k) \forall k \in [1, 7]$
So the flow remains simple, Marks (i, j) as visited and then for all possible moves recursively call the function again for that cell if it is not broken and $m > 0$ and the cell is within the board however for these two, incase you encounter a broken cell you stop going forward as Rooks/Bishops won't be able to jump over obstacles.

Expected Time complexity: $O(m(64 + 64 * 4) * 3) = O(m)$

Question 2: B. Birthday Dilemma
Topic : Sorting + Binary Search

Let's assume that our answer has 2 values, one = **maximum** number of friends having the same quantity of toffee and $second$ = the quantity of toffee each one of them is having.

One of the main observations needed to solve this problem, is that the second number in answer always coincides with someone a_j.

Let's see why it is true. Suppose, the second number of the answer is $a_j + d$ for someone j and $a_j + d \neq a_i$ for all i. This means, we increased some numbers, which is less than a_j so that they became equal to a_j, and then all this numbers and some numbers, which is equal to a_j, we increased to $a_j + d$. But if we didn't increase all these numbers to $a_j + d$ and remain equal to a_j., we'd perform less operations and the answer would be better.

Due to this fact we can solve problem in such a manner. Sort array in non-decreasing order. Iterate over a_i and calculate, what is the maximal number of a_i we can obtain. For maximizing the first number of answer, we must increase some lesser numbers to a_i and perform no greater than k operations. It is obvious that firstly we should increase such a_j that $a_i - a_j$ is minimal. So, if we can solve a problem in $O(n^2)$ we would iterate j from i to 0 and increase a_j to a_i, while we could.

But the solution must be faster, and we will use binary search. We will brute the number of numbers, which we must do equal to a_i. Suppose we fix cnt this value. Now we have to check if we can do cnt numbers equal to a_i by not greater than k operations.

For doing this let's calculate $a_i.cnt$ - $\sum_{j=i-cnt+1}^{i} a_j$

If this value is not greater than k , we can do it.

For calculating sums quickly, we can save prefix sums and then $s_{i-cnt+1}, where\ i = s_i - s_{i-cnt}$

Finally we solved this problem in $O(n.log(n))$

Question 1: C. The Death Sentence
Topic: Sorting, Greedy.

This Task is very simple, all we need to do is simulate the process given in an efficient manner.
Store the array as pair of $(val, index)$, sort the array using any $O(nlogn)$ sorting algorithm in a non decreasing order.
Now keep track of two variables, $cnt = 0, sub = 0$. Where cnt the number of moves and sub stores total subtracted till now.
Iterate through the array, left to right, and at each index, if $a[i].val - sub \geq 0$ then
$cnt+ = (a[i].val - sub)/a[i].index + 1$,
$sub+ = ((a[i].val - sub)/a[i].index + 1).(a[i].index)$
At the end your answer is cnt
Finally we solved this problem in $O(n.log(n))$