Nicolas Constantinou

Time Complexities:

The 1st Algorithm, chooses a set randomly from the current instance, and for each element of the set (O(c)) tries to do subtraction of the sets with that element (O(m*c)), put the element in the current Hitting Set (O(k)) and recursively try to find the Hitting Set of size k-1 for the remaining sets (worst-case m-1) (T(m-1, k-1)). => O(c^(k+1) * k * m)

The 2nd Algorithm, saves the number of appearance of all the numbers found in the sets (1 – N) in an array (O(m*c)), then sorts the elements of a random set in descending order (O(c^2)). Then, for each element in that set (O(c)), subtracts all the sets that have that element (O(m*c)), creates the new current Hitting Set (O(k)) and recursively try to find the Hitting Set of size k-1 for the remaining sets (worst-case m-1)(T(m-1, k-1)). => O(n^(k) * ((c * m) + n + k))

The 3rd Algorithm, finds the smallest sized set (O(m*c)), then for each element of that set (O(c)), subtract the sets with that element (O(m*c)), create the new current Hitting Set (O(k)) and recursively try to find the Hitting Set of size k-1 for the remaining sets (worst-case m-1)(T(m-1, k-1)). => O((c^2 * m)^k)

The 4th Algorithm gets the smallest sized set (O(m*c). Then for the smallest set, it sorts its elements in order from most found to least found in all the sets of the instance(O(c^2)), then for each element of that set (O(c)), subtract the sets with that element (O(m*c)), create the new current Hitting Set (O(k)) and recursively try to find the Hitting Set of size k-1 for the remaining sets (worst-case m-1)(T(m-1, k-1)). => O(c^(k+1) * (m*c + k))

Predictions:

As N grows, Algorithm 1 will randomly choose a set, the range of number will be bigger so it will be more difficult to choose a correct pick. The 2nd one will be affected regarding speed and efficiency exponentially as the overhead to calculate the most found and the branches of recursion will be impacted directly. The 3rd one's speed and efficiency will not be affected due to its selection of the smallest set. The 4th will be affected but not as much as the 2nd one. The overhead will be bigger but still only the smallest set will be checked so it will still be more efficient than the first 2 algorithms.

As M grows, Algorithm 1 will show linear growth due to the subtraction of each recursion but will still have unstable efficiency due to the random selections. Algorithm 2, will show high impact on speed and efficiency due to bigger. The 3rd one will show almost no impact due to the subtraction and the little overhead that finds the smallest set, but will still be the best performer. The 4th one will be affected speed-wise due to the overhead but less than the 2nd and will be more efficient than the 1st and 2nd.

As C grows, Algorithm 1, 3 and 4 will be exponentially affected due to the increase of branches, the 1st one will be the worst because of the random set selection and the 2 others will have a bit more overhead but less branches as they choose the smallest set (4th will have more overhead than 3rd). The 2nd one, will be affected a little overhead-wise but not much.

As K grows, every algorithm will need to go deeper in more recursion branches. The 1st one, will check more branches due to its random set picking. The 2nd one, will have a lot more overhead and will be slower and less efficient. The 3rd and 4th, will need to go to deeper recursion branches but will go to less than the rest, due to their smallest set picking.

Overall, The fastest will be algorithm 3, the slowest will be algorithm 2, the most consistent, stable and efficient will be the 3rd, the best for small K will be algorithm 1 as it is the simplest without overhead. Also, the best for high frequency numbers will be algorithm 2. Lastly, if a

slight overhead can be afforded, the 4th one will be the most well-rounded for all sizes of all parameters.