C3 Noxious Numbers

Consider the following problem. You have a collection of positive integers (probably with some repeats). There is an array of positions. The task is to place the integers in the array subject to the following constraint:

*For each $i$, any two copies of the number $i$ must have at least $i$ positions between them.*

That is, the 1’s cannot be next to each other; the 2’s must have at least two numbers between them; and so on. For example, the following is one way to place two 1’s and two 2’s in an array of size 5:

\[
\begin{array}{cccc}
1 & 2 & 1 & -
\end{array}
\]

The general question is: when can one place the numbers? One way to prove impossibility is a “density” argument. Specifically, if the array has length $n$, then:

\[
\text{there can be at most } \left\lceil \frac{n}{i+1} \right\rceil \text{ (meaning } \frac{n}{i+1} \text{ rounded up) copies of the number } i.
\]

(Why?)

(Add example.)

In the other direction, to prove that the placement can be done, it seems simplest just to exhibit it. So we can think of various possible approaches. Three natural strategies come to mind. (A) Random (B) Greedy (C) Exhaustive search.

The basic idea behind random strategy is to take the numbers one at a time and place them randomly in any position that they can go in. This does not work so well, but is a decent idea. One can try to improve the process by ordering the numbers before placement; for example, the larger numbers are the hardest to fit (but not if there’s only one of them). A better algorithm would be to use something like what is called simulated annealing. We do not discuss this, but a simplified view is that when we get stuck, we randomly re-position one of the existing numbers.
A greedy algorithm is one that does things one at a time, each step choosing what seems best at that point, without trying to look ahead. (This is for example applied in making change at the checkout.) So for this problem, we might try a greedy “sweep”. Start at the leftmost position and go position by position and place in it the smallest number that can fit (if any). This approach does somewhat better. And there are other ways of implementing a greedy algorithm.

The third possibility is an exhaustive search. In this approach, we consider every possibility in some systematic fashion. We do not implement this.

isLegalAdd.m, randomPlacement.m, greedyPlacement.m