The Bin Packing problem is the problem of finding the smallest number of containers needed to hold a set of items. More formally: given a set of \( n \) items, each with a size, \( s_i \), and set of bins, \( B \), where each bin has a fixed capacity, \( c \), find the minimum number of bins needed to pack all items.

One strategy for solving the bin packing problem is to do a first fit heuristic. Here you pack the items one at a time into the first bin that the item will fit into. If it doesn’t fit into any of the current bins, you start a new bin. Formally state this as an algorithm and show that it is always within 2 times the optimal solution. (Be sure to draw examples with different sized items.)

The Bin Packing problem is what is called an optimization problem. Here we want to optimize (in this case minimize) the number of bins used. It is not in NP because it is not a decision problem. It is, however, NP-hard and each NP-hard optimization problem has a corresponding decision problem which is NP-complete. Formally rephrase the Bin Packing problem as a decision problem, and specify it as a set.

Solution: We first outline the algorithm described above formally, then analyze its solution relative to the optimal solution. Finally, we rephrase Bin Packing as a decision problem and specify it as a set.

We construct a machine \( M \) to execute the algorithm. \( M \) takes as input \( \langle I, c \rangle \), where

\[
I = \{(t_1, s_1), (t_2, s_2), (t_3, s_3), \ldots, (t_n, s_n)\}
\]

and \( c \) is an integer. Intuitively, \( I \) is the pairing of each item with its size and \( c \) is the capacity of the bins. We also suppose that our machine has an infinite supply of variables \( b_k \) and that, initially, the value of these variables is 0. The values of the \( b_k \)’s will represent how ‘full’ the bins are.

\( M \) on \( \langle I, c \rangle \):
1. For \( i = 0 \) to \( |I| \)
   (a) For \( k = 0 \) to \( \infty \)
      i. If \( b_k + s_i \leq c \), set \( b_k \) to \( (b_k + s_i) \)
         break
   2. Find the largest \( k \) s.t. \( b_{k+1} \) is empty
   3. Return \( k \)

We can see the algorithm runs in \( O(n^2) \) time. For each item, we have to check at most \( n \) bins. The worst case is where every bin holds an individual item.

In order to show that the algorithm is within 2 times the optimal solution, we first prove that when the algorithm halts, all (non-empty) bins are more than half full, except possibly one. Suppose that two bins, \( b_i \) and \( b_k \), were less than half full (and suppose \( i < k \)). Then, by definition of \( M \), the contents of \( b_k \) would have been placed in \( b_i \). Therefore, such a final state is not possible.

Now suppose the optimal algorithm uses \( b \) bins, and our algorithm uses \( 2b + 1 \) bins (or more). In this case, our algorithm results in \( 2b \) bins that are more than half full, and the total contents of
these bins sums to greater than $\frac{1}{2}c \times 2b = cb$. More formally $\Sigma s_i > cb$.

However, given that all items fit into $b$ bins of size $c$ in the optimal solution, the sum of the sizes of all items is less than or equal to $cb$. More formally, $\Sigma s_i \leq cb$, which contradicts our conclusion above.

Therefore, the algorithm above uses no more than $2b$ bins.

We now rephrase Bin Packing as a decision problem: is $b$ the smallest number of size $c$ bins needed to pack the items in $I$? A solution to this problem would decide the following set:

$$\{ \langle I, c, b \rangle \mid b \text{ is the smallest number of size } c \text{ bins needed to pack the items in } I \}$$