Problem Description

The bin packing problem is taking a list of item sizes and identical bins with capacity C, put the items into the minimum number of bins required.

Practical Applications

Practical applications include:

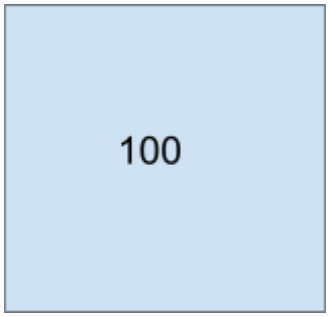
shipping
resource and memory allocation
Semiconductor design

- fitting logic circuits onto physical components

Input Details

Input:

List of item sizes and the capacity of each bin



Bin of size 100

items and their size

Output:
The minimum number of bins required to store all the items

Example of polynomial time modification

Restrict Item Sizes to a Constant Number of Distinct Values

When the number of distinct item sizes is constant:

- You can count how many of each size exist.
- Since each bin can only fit a bounded number of combinations of these sizes, and the number of combinations depends only on the constant k, not on n, **you can enumerate all feasible bin configurations in constant time**.
- Then the problem reduces to solving an integer linear program with **constant dimension**, which is polynomial-time solvable.

Reduction

How the reduction works: (pictures)

Min Graph coloring problem is

- if you are coloring each vertex a color
- the minimum number of colors required to color the graph
- so that no two adjacent vertices share the same color

The reduction is

- Given a graph G = (V, E) and integer k
- We can build a bin-packing instance
- Such that G is k-colorable and the constructed items can be packed into k bins of capacity C

put one **digit position** (a binary place) for every **edge** of the graph. Give each vertex an integer whose binary representation has a 1 in exactly the bit-positions corresponding to edges incident on that vertex. Two adjacent vertices share an edge bit; placing both into the same bin sets that bit to 2 (in binary arithmetic), which will then exceed the capacity digit (which is 1 for every edge-position). Therefore, vertices packed together correspond exactly to an independent set (no edges between them), i.e., a color class. So k bins = k color classes.

Formal Construction of the reduction:

Input:

graph G = (V, E),
V = n
E = m
Integer k

Label the edges e_1…e_m.

For each vertex $v \in V$ define an integer weight (necessary for item sizes and to encode the vertex adjacency)

$$w_v = \sum_{i=1}^{m} 2^{i-1}$$

Set the bin capacity C to

$$C = \sum_{i=1}^{m} 2^{i-1} = 2^m - 1$$

This completes in time O(m + n) compute, for each vertex, the bit-sum of its incident edges): polynomial time.

Polytime proof:

Construction time:

Labels edges and for each each vertex add up at most 2^v O(m +n)

The integers w_v and C are each at most 2^m in value, so encoded in m bits. The input size of the output is polynomial in the input size of G (the graph already has m edges), so the reduction is polynomial-time

Correctness proof:

If G is k-colorable, then the items pack into k bins

A k-coloring partitions the vertex set V into k color classes S1, …, Sk, each of which is an independent set.

For any independent set Sj, consider the integer sum of the weights:
Sum_{v in Sj} w_v.
For each edge-bit i (corresponding to edge e_i), at most one vertex in Sj is incident to e_i (because Sj contains no edges). Therefore, the binary digit in bit position i of the sum is at most 1. Thus the total sum is less than or equal to the bin capacity C.

Therefore, each Sj fits into one bin of capacity C. Since the k color classes S1, …, Sk each fit into a bin, all items pack into k bins.

If the items pack into k bins, then G is k-colorable

Suppose we have a packing of the vertices into k bins. View the vertices placed in each bin as a candidate color class.

Consider any bin with vertex set S. If two vertices u and v in S were adjacent (connected by edge e_t), then both weights w_u and w_v contain a 1 in bit position t (the bit corresponding to edge e_t). Adding them would create a value of at least 2 in that bit position, which exceeds the capacity C (because C has only a 1 in every bit position). This contradiction shows that no bin may contain both endpoints of an edge.

Thus each bin contains only mutually non-adjacent vertices—that is, each bin is an independent set. The packing therefore defines a partition of V into at most k independent sets, which is exactly a k-coloring of G.

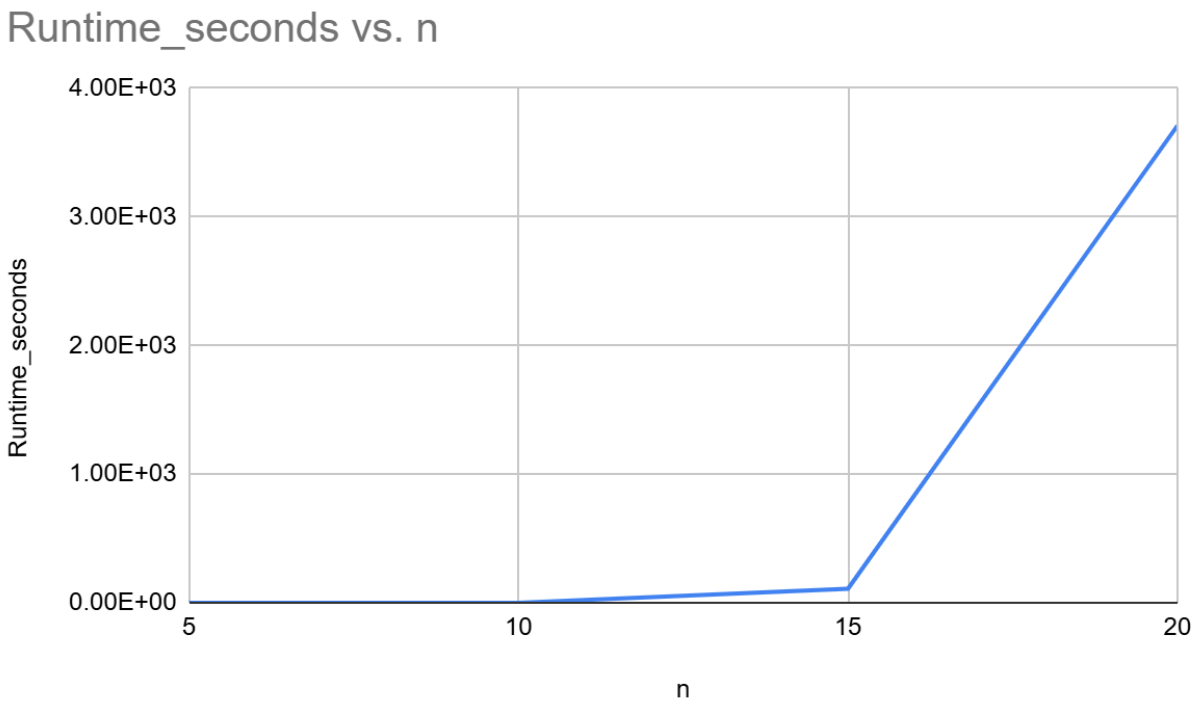
Exact Solution Approach

The program implements a **brute-force backtracking search** to find the *exact* minimum number of bins needed to pack all items without exceeding bin capacity. Because Bin Packing is **NP-hard**, the algorithm explores all feasible arrangements of items into bins, except where recursion naturally prunes infeasible branches.

The algorithm has factorial worst-case complexity, approximately O(n!). This occurs because each item can branch into increasingly many bins (up to nnn), leading to (n+1)! total recursive calls in the worst case. However, pruning and small practical problem sizes often make it faster in real use. Space complexity is

Input Order Analysis

Changing the input order so that **larger items are processed first** significantly improves the speed of the backtracking solver. Large items quickly constrain bin capacity, causing early pruning of infeasible branches. This prevents the solver from exploring large numbers of partial packings that would inevitably fail once a large item is finally considered. Additionally, good solutions are found earlier, enabling more global pruning. As a result, the effective branching factor of the recursion tree drops sharply, leading to much faster runtimes even though the worst-case complexity remains exponential.


Runtime_seconds vs. n
4.00E+03
3.00E+03
2.00E+03
1.00E+03
0.00E+00
Runtime_seconds
5
10
15
20
n