

# Approximation Approach

It is greedy on the choice of leaving the picking the bin that will leave the least amount of room left.

If it doesn't fit in any box then it adds a new one and add it to that.

The randomness comes in that every time it shuffles the randomly beforehand.

This allows for the anytime solution and stops the greedy solution from getting stuck in bad item order.

Example of greedy not getting the solution immediately:

Bin Capacity: 10

Items: [2,5,4,8]

Optimal is: [[8, 2], [5, 4]]

But we get in this item ordering: [[2, 5], [4], [8]]

# Runtime Analysis

Shuffling the items is  $O(n)$

For each item ( $n$  items):

- Checks all existing bins to find the best-fit

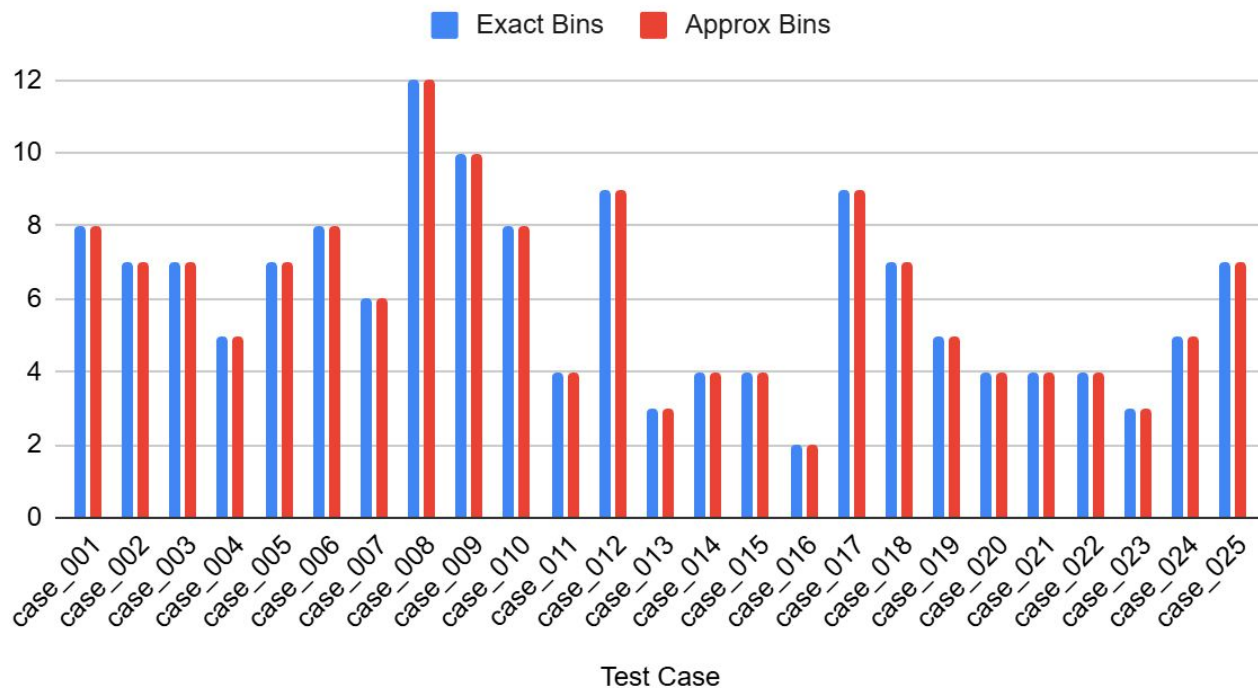
- $b$  = number of bins currently being used

- Worst case  $b = n$  (every item has its own bin)

So it is  $O(b \cdot n)$

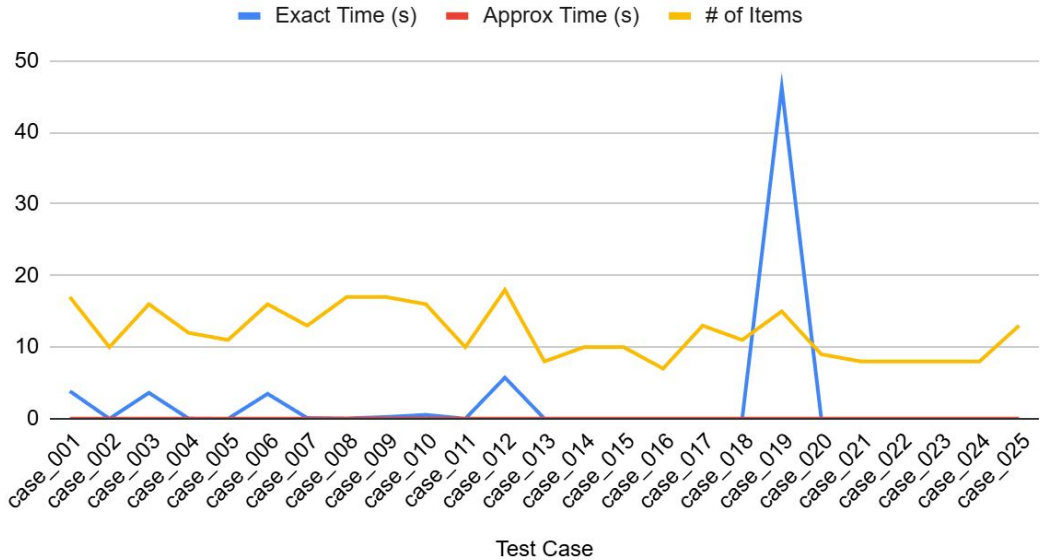
Worst case  $O(n^2)$

## Optimal vs Approximate Solutions

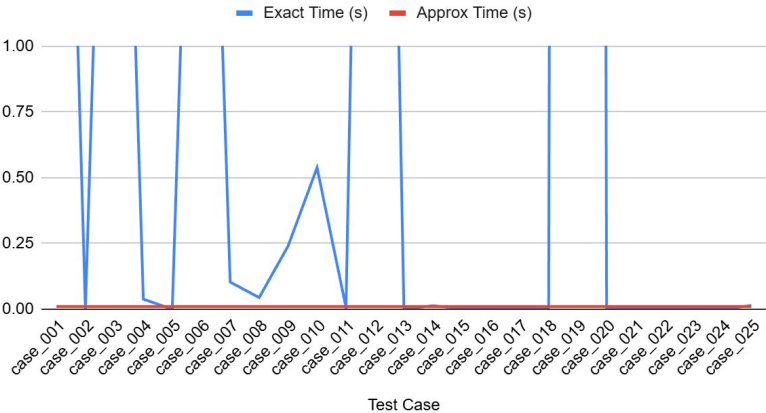


Approximate gets the optimal solution every time with a time limit of 0.01

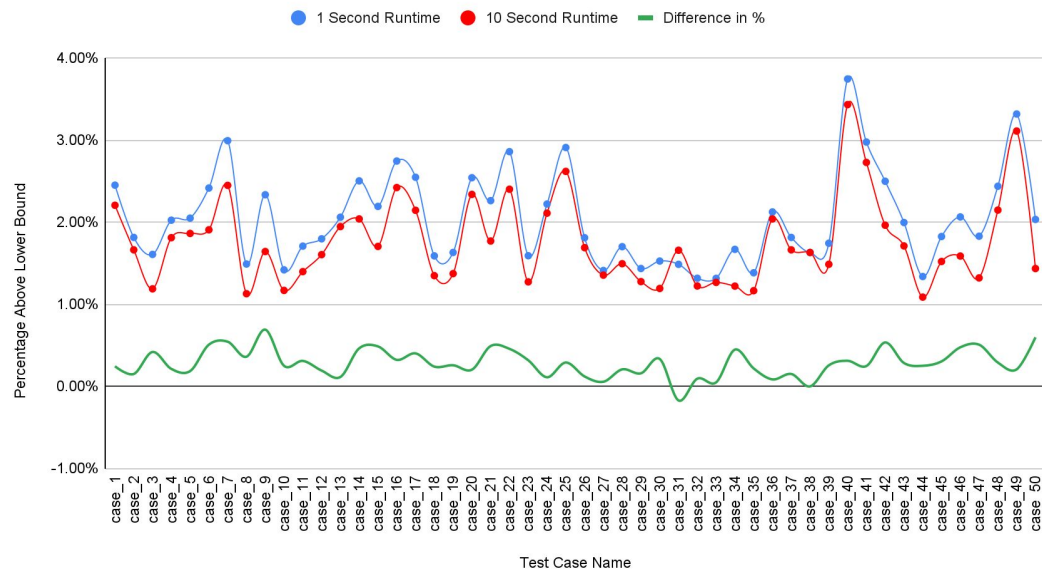
# Optimal vs Approximate Solutions



## Optimal vs Approximate Solutions



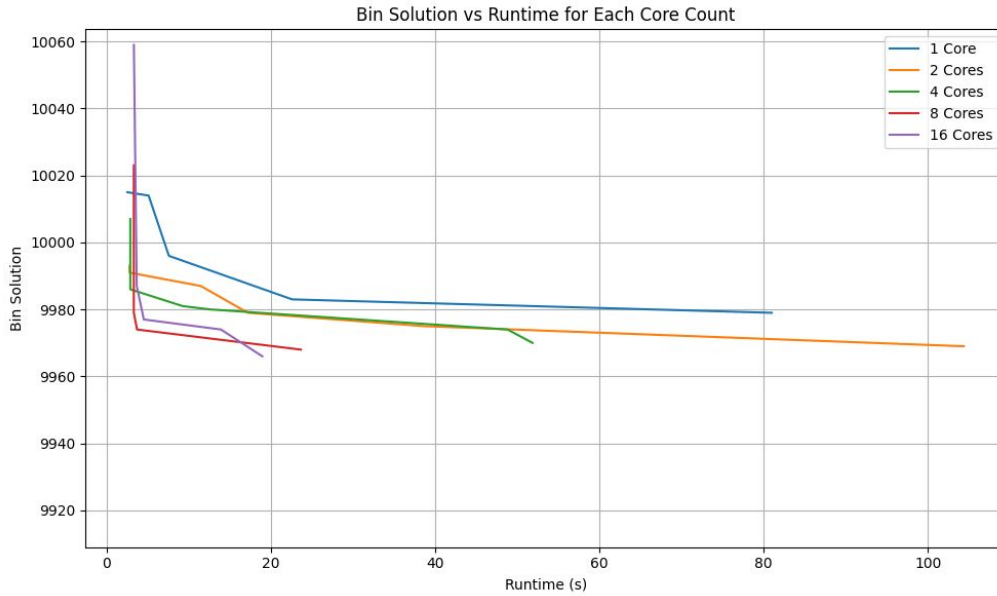
Percentage above Lower Bound per Test Case (Lower Bound = Sum of Items / Bin Capacity)



Worst: case\_31: -0.17%

Best: case\_9: 0.69%

I ran case\_31 again until it got something better than the 1 sec Runtime approximation of 1774 Bins. It got 1772 Bins at 145.508s



The more cores the faster it goes down and the earlier it finds its final solution. This auto-terminated after 120 seconds.

The lower bound of the Y-axis is also the lower bound of the items set for reference (9909).