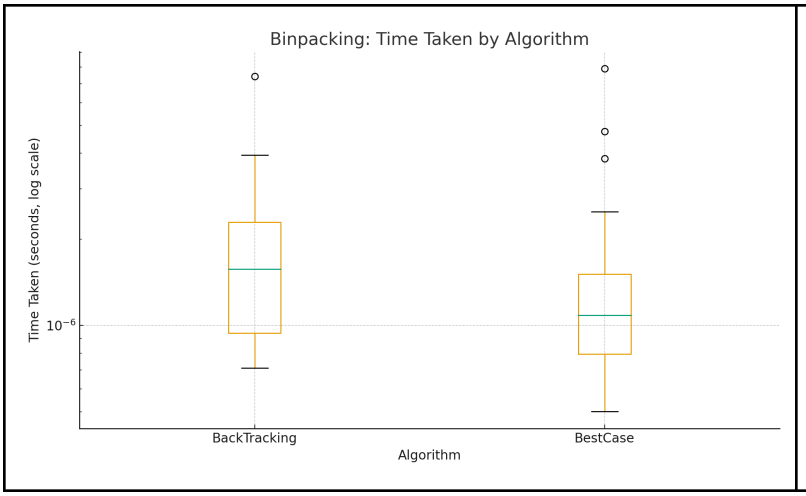
Project 1 Readme Team Transition Titans

Version 1 11/16/25

<table>
<tr><td>1</td><td>Team Name: Transition Titans</td></tr>
<tr><td>2</td><td>Team members names and netids: Matt Daly - mdaly22, Andrew Myers, amyers9</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects:</td></tr>
<tr><td>4</td><td>Overall success of the project: Pretty solid success, we were able to hook up two separate algorithms for the bin packing problem that generate novel and correct solutions, and recognize when problems are unsolvable. They are not meant to show all solutions, as they just present the first one found.</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 12 hours</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/MDaly27/Daly-Myers-Project1-TOC</td></tr>
<tr><td>7</td><td>List of included files (if you have many files of a certain type, such as test files of different sizes, list just the folder): (Add more rows as necessary). Add more rows as necessary.
<table>
<tr><td>File/folder Name</td><td>File Contents and Use</td></tr>
<tr><td colspan="2">Code Files</td></tr>
<tr><td>src/bin_packing.py</td><td>Algorithms code for best case and backtracking solutions</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>Binpacking_tests.txt, binpacking.txt</td><td>Test input, input</td></tr>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>results/ (best_case_binpacking_sat_solver_results.csv, btracking_binpacking_sat_solver_results.csv)</td><td>Function outputs</td></tr>
<tr><td colspan="2">Plots (as needed)</td></tr>
</table>
</td></tr>
</table>

	
8	Programming languages used, and associated libraries: Python, standard library
9	Key data structures (for each sub-project): lists, stacks (function call stack)
10	General operation of code (for each subproject): Backtracking: We have a recursive function with an idx, current_sum, and current_list parameters, and run through the list of clauses. If adding a clause to current_sum gets us to target, return our set of clauses. Otherwise, continue by either skipping the current coin and moving to idx+1 or accepting the coin and going to idx+1 with current_sum+current. Best case: We have a list of target + 1 elements that are either None or a list of clauses. If the target + 1 element of the list is not None, return it. Otherwise, loop down through the list from the current element, and update i+c for any i in the list that is not None. Do this for every clause.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. We made test cases showing various possible and impossible configurations, trying to define edge cases when possible.
12	How you managed the code development We worked in iterations, first learning about how the github code was structured, and then diving into the actual algorithms. Keeping versions and communicating within the team were critical to our success.
13	Detailed discussion of results: Interestingly, I found that the best case performed worse than the backtracking solution for the given input cases. This is likely due to the smallness of the data and I believe

	would level out with larger tests, however it could be due to how the algorithms get implemented by the python interpreter.
14	How team was organized Matt - Lead developer Andrew - Co lead developer
15	What you might do differently if you did the project again I would have spent more time trying to understand the reasoning behind the algorithms, and try to solve them without resources.
16	Any additional material: I used Geeks for Geeks and w3school to help understand the bin packing problem. ChatGPT was occasionally used for inspiration and debugging.