Project 1 Readme Team nefario

1	Team Name: team nefario
2	Team members names and netids Brandon Hulse (bhulse)
3	Overall project attempted, with sub-projects: DumbSAT equivalent for Hamiltonian paths
4	Overall success of the project: wildy successful
5	Approximately total time (in hours) to complete: 4-5 hours
6	Link to github repository: https://github.com/bhulse72/TOC-proj01-dr_nefario
7	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.

File/folder Name	File Contents and Use
Code Files	
hamiltonianPath_nefario.py test_automation_nefario.sh	python file takes input files in standard input format, finds yes/no Hamiltonian path and compute time bash script runs each input file through hamiltonianPath_nefario.py and redirects output into output files
Test Files	
checkFolder_nefario: check00-nefario.txt check02-nefario.txt check04-nefario.txt check06-nefario.txt check08-nefario.txt check01-nefario.txt check03-nefario.txt check05-nefario.txt check07-nefario.txt check09-nefario.txt	each file contains a graph in the outlined input format. used to test for hamiltonian path and compute times

Output Files	
outputFolder_Nefario: output00_nefario.txt output02_nefario.txt output04_nefario.txt output06_nefario.txt output08_nefario.txt output01_nefario.txt output03_nefario.txt output05_nefario.txt output07_nefario.txt output09_nefario.txt	output files that contain results from test files: compute times and yes/no Hamiltonian path
Plots (as needed)	
plots_nefario.png	scatter plot with best-fit lines to demonstrate time complexities

8	Programming languages used, and associated libraries: python. sys, collections, time. also a bash script
9	Key data structures (for each sub-project): dictionary, set, list
10	General operation of code (for each subproject): recursive algorithm with backtracking to find a hamiltonian path. It takes the current graph, a path (list of visited vertices), and a visited set (to track visited vertices). If the length of path equals the number of vertices in the graph, it means all vertices have been visited, and a Hamiltonian path is found. It then iterates through the neighbors of the last vertex in the path. If a neighbor hasn't been visited, it marks it as visited and appends it to the path. The function recursively calls itself. If the recursive call returns True, a Hamiltonian path has been found. If not, it backtracks by removing the neighbor from the visited set and popping it from the path, allowing for other permutations to be explored
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. I developed a series of test cases, each of which was progressively slightly bigger (more edges and vertices) to demonstrate the growing time. I included an even amount of test cases that contained Hamiltonian paths and cases that did not. For the test cases with less than 12 nodes, I simply drew them out on paper and manually determined if there was a hamiltonian path. For the larger test cases, I found cases online. Because I knew wether a test case had/didn’t have a hamiltonian path, I was able to use the test cases to see if the algorithm was properly identifying a graph as having/not having a hamiltonian path.
12	How you managed the code development

	I used git to practice iterative development and version control. I started with a brute force approach and then improved it to the recursive approach with backtracking. I then added testing. Because I worked by myself, I basically did whatever I wanted.
13	Detailed discussion of results: This project revealed the high time complexity for finding Hamiltonian paths in undirected graphs. Despite the recursive backtracking approach that greatly improves compute time compared to brute force, adding more nodes dramatically increases compute time, with each node adding more compute time than the last. This is especially visible in cases where there is no Hamiltonian path. This is because if a Hamiltonian path is found, the algorithm stops. These results demonstrate how finding Hamiltonian paths is an NP-hard/complete problem.
14	How team was organized it was just me
15	What you might do differently if you did the project again I would probably try to find access to more graphs I could test on and then try to automate the output to a spreadsheet. This way, I could have a dashboard of sorts that could be updated any time I added a test case.
16	Any additional material: I hope you have a good break/weekend/free time