Problem 1b. Enter the Time and compute the Ratio of Times to two decimal places (x.xx)

Graph Size	Time for Computing Spanning Tree	Ratio of Time: Size 2N/Size N
1,000	0.06	No ratio for first graph size
2,000	0.13	0.13/0.06 = 2.17
4,000	0.27	0.27/0.13 = 2.08
8,000	0.57	0.57/0.27 = 2.11
16,000	1.41	1.41/0.57 = 2.47
32,000	3.60	3.60/1.41 = 2.55
64,000	8.20	8.2/3.6 = 2.28
128,000	18.24	18.24/8.2 = 2.22

Approximate the complexity class for the spanning_tree function based on the data above. Briefly explain your reasoning.

Answer: The complexity class for 'spanning_tree' is O(N Log N). This is because the ratios of time are, on average, 2.26. This is slightly higher than O(N)'s ratio, thus it is O(N Log N).

Problem 2b. Answer each of the following question based on the profiles produced when running spanning_tree : use the ncalls information for parts 2 and 3; use the tottime information for parts 1 and 4.

1) What function/method takes the most tottime to execute?

Answer: **{built-in method builtins.sorted}**

2) What non-built in function/method is called the most times?

Answer: **equivalence.py:28(_compress_to_root)**

3) What method defined in graph.py is called the most times?

Answer: **graph.py:23(__getitem__)**

4) What percent of the entire execution time is spent in the 5 functions with the most tottime?

Answer: **76%**