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	0.05	No ratio for first graph size
2,000	0.11	2.20
4,000	0.24	2.18
8,000	0.56	2.33
16,000	1.37	2.45
32,000	2.97	2.17
64,000	6.67	2.25
128,000	13.94	2.09

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

Answer: The complexity class is O(N). According to the data in the rightmost column in the chart above, when the graph size doubles, the time python takes to compute spanning_tree almost doubles at the same time. For all kinds of the complexity classes, only O(N) has this kind of behavior. Therefore, the complexity class is O(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: In most cases, `__getitem__` is called the most times. In the other cases, `_compress_to_root` is called the most times.

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

Answer: `__getitem__`

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

Answer: 75.5%