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	0.07	No ratio for first graph size
2,000	0.16	2.28
4,000	0.34	2.13
8,000	0.76	2.24
16,000	1.59	2.09
32,000	3.48	2.19
64,000	7.60	2.19
128,000	16.69	2.20

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

Answer: O(N Log_2 N) : ratios are always a bit bigger than 2. If your ratios show ratios smaller than 2 and some ratios bigger than 2, then writing O(N) is OK as well. There isn't much difference between these two complexity classes because the Log_2 N term grows so slowly.

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: **`sorted`** (from the builtins module)

2) What function/method not from the **`builtins`** module is called the most times?

Answer: **`_compress_to_root`** or **`__getitem__`** (depends on the random graph)

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: About 78%.