20/20

JL 2/21/21

* Good work!

Project (Week 5)

1. Use and modify the textbook Code Fragment 9.1, 9.4, 9.5 and 9.7 to implement a standard heap-sort algorithm.

See file *Project5\_1\_Mikhail.py*

Enter the number (1000-10000) of random generated float data to be sorted (default: 1000) or 'q' for stop program: 1000

Sort time was taken= 0.030677080154418945

Enter the number (1000-10000) of random generated float data to be sorted (default: 1000) or 'q' for stop program: 1500

Sort time was taken= 0.07460451126098633

Enter the number (1000-10000) of random generated float data to be sorted (default: 1000) or 'q' for stop program: 5678

Sort time was taken= 0.29016613960266113

Enter the number (1000-10000) of random generated float data to be sorted (default: 1000) or 'q' for stop program: 10000

Sort time was taken= 0.5056653022766113

1. Study the textbook section 9.3.2 and implement another in-place heap-sort algorithm. Note the heap here is defined as a ***maximum***-oriented heap, with each position’s key being at least as ***large*** as its children.

See file *Project5\_2\_Mikhail.py*

Enter the number (1000-10000) of random generated float data to be sorted (default: 1000) or 'q' for stop program: 1000

Sort time was taken= 0.04229879379272461

Enter the number (1000-10000) of random generated float data to be sorted (default: 1000) or 'q' for stop program: 1500

Sort time was taken= 0.052597761154174805

Enter the number (1000-10000) of random generated float data to be sorted (default: 1000) or 'q' for stop program: 5678

Sort time was taken= 0.25087618827819824

Enter the number (1000-10000) of random generated float data to be sorted (default: 1000) or 'q' for stop program: 10000

Sort time was taken= 0.5075287818908691

1. Experimentally compare both algorithms’ running time with a list of random float data of size between 1000-10000.

See “Output” folder

**Submit your code together with the run results.**