1. This problem is to compare different dictionary implementations in Python, including Python’s own built-in implementation.

(a) Develop a dictionary class in Python using an underlying binary search tree implementation. Use the same interface as the dictionary implementations we saw in class based on hash tables. You only need to support the methods necessary to add, delete, and search (getitem, setitem, delitem, contains, and associated helper functions).

(b) Compare your dictionary implementation to the two hash table implementations from class and to Python’s own built-in implementation across a range of use cases.

2. Consider the following scheme for sorting a list of $n$ numbers. For simplicity, let’s assume $n = 2^k$ for some $k$ (the numbers may have more than $k$ bits).

- We first insert the items into a hash table with chaining, with the hash function being the first $k$ bits of the number (this is not a good hash function in general, but serves a purpose here).
- Next, we sort the individual lists of items in each slot in the hash table with insertion sort.
- Finally, we just concatenate all of these sorted lists together.

(a) Why does this algorithm sort correctly?
(b) What is the worst case running time of this algorithm and with what kinds of inputs is it realized?
(c) Assuming the numbers are completely random, what running time would you expect in practice? Explain.

3. A $d$-heap is a heap in which non-leaf nodes in the tree can have up to $d$ children instead of just 2.

(a) Explain how to store a $d$-heap in an array.
(b) What is the minimum height of a $d$ heap of $n$ elements in terms of $n$ and $d$?
(c) Analyze the running times of the basic heap operations in terms of $n$ and $d$.

4. Argue that the average depth of the tree in binary search is $\Theta(\lg n)$. Hint: write a recursion for the average depth a tree of size $n$ and solve it.