1. LRU cache.
   Utilized OrderedDict from Python to take full advantage of the move_to_end.
   Time complexity: O(1) for both put and get due to OrderedDict
   Spacy: O(capacity) since we need to store the ordered dictionary.
2. Inspired by this: https://stackoverflow.com/questions/61255507/time-and-space-complexity-of-a-file-recursion-algorithm
   Choose not to use os.path.isdir and os.path.isfile since they can be easily inferred in this implementation. The recursion is easy. Say e is the length of the path elements, f is the number of files, traversing through g directories would take O(g), then O(e+f+g) = O(e)
3. Huffman tree: This impl is an adaptation of the idea here: https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/ and we can see using a priorityqueue facilitates the impl.
4. Active directory: Recursion is used again.
   Time Complexity O(# groups * # number of users) Spacy: O(1) as it does not store anything than output
5. Blockchain: It is clearly a linked list with append O(1) search O(n).
6. Union/Intersection: Standard impl.
   Union: Time complexity: O(n)
   Space: O(n)
   Intersection: Time Complexity O(n^2) as we need to find the 'in' relationship.
   Space: O(n)