User Hints:

- The LazyPhoneDialingClass.py takes one argument(argv) the number sequence string (“56789”)
    - Eg: python LazyPhoneDialingClass.py "075"
- The .py file already has the driver (main()) function to test the interface
    - (Or alternatively the file OnlyDef.py has the class definition and the Driver.py has the corresponding test code)
- The required interface is a member function of a class “Keypad” (this allows us to extend to other keypad patterns and formats)
- The class Keypad has a member variable debug, 0 for displaying steps (limited prints, recommended), 1 for debug style prints, and -1 for ignoring all print statements

Implementation outline

- Class Node : is used to create binary tree
- Class Keypad : contains all the functions interfaces
- Two functions are interesting “compute_baseline_path(inputstring)” and “compute_laziest_path (inputstring)””
    - Both will return the result in expected format(question), the latter gives the shortest possible solution(internally calls compute_baseline_path)
- compute_baseline_path
    - just looks the next number in the sequence and determines which finger to use
    - computation complexity : O(n) and space complexity: O(1)
        - n corresponds to the length of the input string
- compute_laziest_path
    - starts building all the possible ways to type in the sequence, hence take 2^n routes
        - eg: for 110 there are 8 possible routes
        - implemented as a binary tree
        - make 2^(n+1) -1 nodes
    - the result from compute_baseline_path is used to stop the growth of branches which exceed the best distance calculated by compute_baseline_path
        - thus, the tree is stopped from becoming “perfect”
        - this reduces few node computation
        - use debug value = 0 to see the difference
    - worst case, computation complexity: O(2^(n+1)) and space complexity: O(2^(n+1))
        - the number of node times recursion

Exceptions

- The code assumes the input string is always correct, i.e. no invalid values (question)
- The compute_baseline_path does not resolve cases with equal distance
    - The code is hard coded to prefer left in this type of situations
- In few sections of the code consciously used extra variable for better readability of the algorithm
    - Potentially they can be simplified