CSE 332: Data Structures and Parallelism

Section 1: `WorkLists`

WorkList ADT

add(**work**)	Notifies the worklist that it must handle **work**
peek()	Returns the next item to work on
next()	Removes and returns the next item to work on
hasWork()	Returns true if there's any work left and false otherwise

0. 𝒪dd Jobs

For each of the following scenarios, choose

(1) an ADT: `Stack` or `Queue` , and

(2) a data structure: `Array` or `LinkedList with front` or `LinkedList with front and back`

Then, explain why your choice works better than the other options.

WorkList Situations

(a) You're designing a tool that checks code to verify that all opening brackets, braces, parentheses, ... have closing counterparts.

(b) Disneyland has hired you to find a way to improve the processing efficiency of their long lines at attractions. There is no way to forecast how long the lines will be.

(c) A sandwich shop wants to serve customers in the order that they arrived, but also wants to look ahead to know what people have ordered and how many times to maximize efficiency in the kitchen.

1. Choosing The Data Structures

Choose data structures and algorithms to solve the following problems:

(a) Call all the phone numbers with a particular area code in someone's phone book.

What is the time complexity of your solution? What is the space complexity?

(b) Text on nine keys (T9)'s objective is to make it easier to type text messages with 9 keys. It allows words to be entered by a single keypress for each letter in which several letters are associated with each key. It combines the groups of letters on each phone key with a fast-access dictionary of words. It looks up in the dictionary all words corresponding to the sequence of keypresses and orders them by frequency of use. So for example, the input '2665' could be the words {book, cook, cool}. Describe how you would implement a T9 dictionary for a mobile phone.