208 Implement Trie(Prefix Tree)
This is a basic but meaningful question. We wii a class for TrieNode which stores a map and a label for whether it's the end of a word. So the TC for insert and search are both O(n)

```
from collections import defaultdict
class TrieNode:
    def __init__(self):
        self.children = {}
        self.isEndOfWord = False

class Trie:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.root = TrieNode()

    def insert(self, word: str) -> None:
        """
        Inserts a word into the trie.
        """
        cur = self.root
        for w in word:
            if w not in cur.children:
                cur.children[w] = TrieNode()
            cur = cur.children[w]
        cur.isEndOfWord = True

    def search(self, word: str) -> bool:
        """
        Returns if the word is in the trie.
        """
        cur = self.root
        for w in word:
            if w not in cur.children:
                return False
            cur = cur.children[w]
        return cur.isEndOfWord

    def startsWith(self, prefix: str) -> bool:
        """
        Returns if there is any word in the trie that starts with the given prefix.
        """
```

```
        cur = self.root
        for w in prefix:
            if w not in cur.children:
                return False
            cur = cur.children[w]
        return True
```

8. String to Integer (atoi)
This question is not hard, but it has so many edge cases. So it's very hard to AC it in one pass.
TC is O(n).

```
class Solution:
    def myAtoi(self, str: str) -> int:
        word = str.strip()
        digits = ''

        for ind, c in enumerate(word):
            if ind == 0 and c in '-+':
                digits += c
                continue
            elif c in '1234567890':
                digits += c
            else:
                break
        if digits and digits not in '+-':
            num = int(digits)
            if num > 2 ** 31 - 1:
                num = 2 ** 31 - 1
            elif num < -2 ** 31:
                num = -2 ** 31
            return num
        else:
            return 0
```

.

9. Palindrome Number
Nothing to say. Do whatever you like.

```
class Solution:
    def isPalindrome(self, x: int) -> bool:
        stack = []

        if x < 0:
            return False

        while x:
```

```
            x, rest = divmod(x, 10)
            stack.append(rest)
        return stack == stack[::-1]
```

139. Word Break
This is very easy if you come up with dp. Using dfs will cause TME. We should use dp. If dp[i] == True and s[i + j] in wordList, dp[i +j] = True. TC is O(n ** 2)

```
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> bool:
        dp = [False] * (len(s) + 1)
        dp[0] = True
        wordDictSet = set(wordDict)

        for i in range(len(s) + 1):
            for j in range(0, i):
                if dp[j] and s[j:i] in wordDictSet:
                    dp[i] = True
                    break
        return dp[-1]
```

341. Flatten Nested List Iterator
For this question, we will use dfs to flatten nested list. Then it's very easy to implement next and hasnext. TC is O(mn)

```
class NestedIterator(object):

    def __init__(self, nestedList):
        """
        Initialize your data structure here.
        :type nestedList: List[NestedInteger]
        """
        self.store = []
        def dfs(nestedList):
            for e in nestedList:
                if e.isInteger():
                    self.store.append(e.getInteger())
                else:
                    dfs(e.getList())

        dfs(nestedList)
        self.store.reverse()
```

```
def next(self):
    """
    :rtype: int
    """
    return self.store.pop()

def hasNext(self):
    """
    :rtype: bool
    """
    return len(self.store) > 0
```