139. Word Break (Medium)

Given a **non-empty** string *s* and a dictionary *wordDict* containing a list of **non-empty** words, determine if *s* can be segmented into a space-separated sequence of one or more dictionary words.

Note:

- The same word in the dictionary may be reused multiple times in the segmentation.
- You may assume the dictionary does not contain duplicate words.

Example 1:

```
Input: s = "leetcode", wordDict = ["leet", "code"]
Output: true
Explanation: Return true because "leetcode" can be
segmented as "leet code".
```

Example 2:

```
Input: s = "applepenapple", wordDict = ["apple", "pen"]
Output: true
Explanation: Return true because "applepenapple" can be
segmented as "apple pen apple".
             Note that you are allowed to reuse a
dictionary word.
```

Example 3:

```
Input: s = "catsandog", wordDict = ["cats", "dog", "sand",
"and", "cat"]
Output: false
```

300. Longest Increasing Subsequence (Medium)

Given an unsorted array of integers, find the length of longest increasing subsequence.

Example:

```
Input: [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is
[2,3,7,101], therefore the length is 4.
```

Note:

- There may be more than one LIS combination, it is only necessary for you to return the length.
- Your algorithm should run in O(n^2) complexity.

Follow up: Could you improve it to O(*n* log *n*) time complexity?

72. Edit Distance (Hard)

Given two words *word1* and *word2*, find the minimum number of operations required to convert *word1* to *word2*.

You have the following 3 operations permitted on a word:

1. Insert a character
2. Delete a character
3. Replace a character

Example 1:

```
Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')
```

Example 2:

```
Input: word1 = "intention", word2 = "execution"
Output: 5
Explanation:
intention -> inention (remove 't')
inention -> enention (replace 'i' with 'e')
enention -> exention (replace 'n' with 'x')
exention -> exection (replace 'n' with 'c')
exection -> execution (insert 'u')
```

416. Partition Equal Subset Sum (Medium)

Given a **non-empty** array containing **only positive integers**, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

Note:

1. Each of the array element will not exceed 100.
2. The array size will not exceed 200.

Example 1:

```
Input: [1, 5, 11, 5]

Output: true

Explanation: The array can be partitioned as [1, 5, 5] and
[11].
```

Example 2:

```
Input: [1, 2, 3, 5]

Output: false
```

```
Explanation: The array cannot be partitioned into equal sum
subsets.
```

322. Coin Change (Medium)

You are given coins of different denominations and a total amount of money *amount*. Write a function to compute the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return `-1`.

Example 1:

```
Input: coins = [1, 2, 5], amount = 11
Output: 3
Explanation: 11 = 5 + 5 + 1
```

Example 2:

```
Input: coins = [2], amount = 3
Output: -1
```

Note:
You may assume that you have an infinite number of each kind of coin.