Prompt | Scratchpad | Solutions | Video Explanation

Difficulty: Easy **Category:** Arrays **Successful Submissions:** 43,768+

Non-Constructible Change

Given an array of positive integers representing the values of coins in your possession, write a function that returns the minimum amount of change (the minimum sum of money) that you **cannot** create. The given coins can have any positive integer value and aren't necessarily unique (i.e., you can have multiple coins of the same value).

For example, if you're given `coins = [1, 2, 5]`, the minimum amount of change that you can't create is `4`. If you're given no coins, the minimum amount of change that you can't create is `1`.

Sample Input

```
coins = [5, 7, 1, 1, 2, 3, 22]
```

Sample Output

```
20
```

Hints

Hint 1

One approach to solve this problem is to attempt to create every single amount of change, starting at 1 and going up until you eventually can't create an amount. While this approach works, there *is* a better one.

Hint 2

Start by sorting the input array. Since you're trying to find the **minimum** amount of change that you can't create, it makes sense to consider the smallest coins first.

Hint 3

To understand the trick to this problem, consider the following example: `coins = [1, 2, 4]`. With this set of coins, we can create `1, 2, 3, 4, 5, 6, 7` cents worth of change. Now, if we were to add a coin of value `9` to this set, we **would not** be able to create `8` cents. However, if we were to add a coin of value `7`, we **would** be able to create `8` cents, and we would also be able to create all values of change from `1` to `15`. Why is this the case?

Hint 4

Create a variable to store the amount of change that you can currently create up to. Sort all of your coins, and loop through them in ascending order. At every iteration, compare the current coin to the amount of change that you can currently create up to. Here are the two scenarios that you'll encounter:

- The coin value is **greater** than the amount of change that you can currently create plus 1.
- The coin value is **smaller than or equal to** the amount of change that you can currently create plus 1.

In the first scenario, you simply return the current amount of change that you can create plus 1, because you can't create that amount of change. In the second scenario, you add the value of the coin to the amount of change that you can currently create up to, and you continue iterating through the coins.
The reason for this is that, if you're in the second scenario, you can create all of the values of change that you can currently create plus the value of the coin that you just considered. If you're given coins `[1, 2]`, then you can make `1, 2, 3` cents. So if you add a coin of value `4`, then you can make `4 + 1` cents, `4 + 2` cents, and `4 + 3` cents. Thus, you can make up to `7` cents.

Optimal Space & Time Complexity

O(nlogn) time | O(1) space - where n is the number of coins

Your Solutions | Run Code

Solution 1 | Solution 2 | Solution 3

```python
def nonConstructibleChange(coins):
    Change  = 0
    coins.sort()

    for coin in coins:
        if coin > Change + 1:
            return Change + 1
        else:
            Change += coin

    return Change + 1

'''
O(nlogn) time | O(1) space - where n is the number of coins
'''

# Kunal Wadhwa

# https://github.com/kunal5042
# https://leetcode.com/kunal5042/
# https://www.hackerrank.com/kunalwadhwa_cs
# https://www.linkedin.com/in/kunal5042/
```

Custom Output | Raw Output | Submit

Yay, your code passed all the test cases!

13 / 13 test cases passed.

- Test Case 1 passed!
- Test Case 2 passed!
- Test Case 3 passed!
- Test Case 4 passed!
- Test Case 5 passed!
- Test Case 6 passed!
- Test Case 7 passed!
- Test Case 8 passed!
- Test Case 9 passed!
- Test Case 10 passed!
- Test Case 11 passed!
- Test Case 12 passed!
- Test Case 13 passed!