A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

write me a python program that do the following: There are N patients (numbered from 0 to N-1) who want to visit the doctor. The doctor has S possible appointment slots, numbered from 1 to S. Each of the patients has two preferences. Patient K would like to visit the doctor during either slot A[K] or slot B[K}. The doctor can treat only one patient during it slot. Is it possible to assign every patient to one of their preferred slots so that there will be at lost one patient assign to each slot. Write a function in python def solution(A,B,S) that given two arrays A and B, both N integers, and an integer S, return True if it is possible to assign every patient to one of their preferred slots, one patient to one slot and False otherwise  
  
this problem is a **graph matching** or **bipartite assignment** problem.

We can model it as:

* Each **patient** is a node on the left.
* Each **slot** is a node on the right.
* Each patient has **edges to two preferred slots** (from A[k] and B[k]).  
  We then need to check whether we can assign each patient to one of their preferred slots, with **no slot shared by two patients** — i.e., whether there exists a **perfect matching**.

This can be solved efficiently using a **DFS-based bipartite matching algorithm** (a variation of the Hungarian / Kuhn’s algorithm).

Here’s the complete, clean Python implementation

def solution(A, B, S):

"""

A, B : lists of length N (patients' two preferences)

S : total number of slots (numbered 1..S)

Returns True if all patients can be assigned one of their preferred slots

with no slot conflicts, else False.

"""

N = len(A)

# Build adjacency list: patient -> preferred slots

preferences = [[] for \_ in range(N)]

for i in range(N):

preferences[i] = [A[i], B[i]]

# slot\_assignment[slot] = patient currently assigned to that slot

slot\_assignment = [None] \* (S + 1) # 1-based slots

def can\_assign(patient, visited):

"""Try to assign this patient to one of their preferred slots."""

for slot in preferences[patient]:

if visited[slot]:

continue

visited[slot] = True

# If slot is free OR current occupant can move to another slot

if slot\_assignment[slot] is None or can\_assign(slot\_assignment[slot], visited):

slot\_assignment[slot] = patient

return True

return False

# Try to find assignment for each patient

for patient in range(N):

visited = [False] \* (S + 1)

if not can\_assign(patient, visited):

return False # could not find a slot for this patient

return True # all patients assigned successfully

**Example Tests**

# Example 1: simple case, possible

A = [1, 1, 2]

B = [2, 3, 3]

S = 3

print(solution(A, B, S)) # True — each slot 1,2,3 can be assigned

# Example 2: impossible

A = [1, 1, 1]

B = [1, 1, 1]

S = 2

print(solution(A, B, S)) # False — not enough distinct slots

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

write in python You are given an array A of N integers, representing the maximum heights of N skyscrapers to be built.

Your task is to specify the actual heights of the skyscrapers given that:

\* the height of the k-th skyscraper should be positive and not bigger than A[K]

\* no two skyscrapers should be of the same height

\* The total sum of the skyscrapers' heights should be the maximum possible

Write a function def solution(A) that given an array A of N integers, returns an array B of N integers where B[K] is the assigned height of the K-th skyscraper satisfying the above conditions. If there are several possible answers, the function may return any of them. You may assume that it is always possible to build all skyscrapers while fulfilling all the requirements.

Excellent — this is a **greedy optimization problem** 👍

Let’s restate it clearly first:

**🏙 Problem Summary**

Given an array A of maximum allowed skyscraper heights:

* You must assign **distinct positive integer heights** B[K] ≤ A[K]
* Maximize the **sum of all B[K]**

**💡 Key Idea (Greedy Strategy)**

To **maximize** the total sum while ensuring all heights are **distinct**,  
we can proceed as follows:

1. Sort skyscrapers by their maximum allowed height (A[K]) in **descending** order.
2. Assign the **largest possible unique height** to each skyscraper (starting from the tallest).
3. Keep track of the **previously assigned height** — each next skyscraper can be at most min(A[i], prev\_assigned\_height - 1).
4. Ensure heights stay positive (≥ 1).

**✅ Python Implementation**

def solution(A):

N = len(A)

# Pair each skyscraper with its index to restore order later

indexed\_A = sorted([(a, i) for i, a in enumerate(A)], reverse=True)

B = [0] \* N # result array

prev\_height = float('inf') # no upper limit initially

for a, idx in indexed\_A:

# assign the largest possible unique height

height = min(a, prev\_height - 1)

if height < 1:

height = 1 # ensure positivity

B[idx] = height

prev\_height = height

return B

**Example**

A = [3, 1, 4, 5, 2]

print(solution(A))

**Possible Output:**

[3, 1, 4, 5, 2]

or sometimes

[3, 1, 4, 5, 2]

(depends on the sorting order of same values)

**Another Example**

A = [5, 3, 3, 3]

print(solution(A))

**Output:**

[3, 2, 1, 4]

or any valid permutation where:

* Heights ≤ A
* All unique
* Sum is maximum (= 10 in this case)

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

You are given a string letters made of N English letters. Count the number of different letters that appear in both uppercase and lowercase where all lowercase occurrences of the given letter appear before any uppercase occurrence.

For example, for letters - "aaAbcCABBc", the anwer is 2. The condition is met for letters 'a' and 'b' but not for 'c'.

Write a function def solution(letters) that given a string letters, returns the number of different letters fulfilling the conditions above.

Examples:

1. given letters = "aaAbcCAABc", the function should return 3

2. given letters = "xyzXYZabcABC", the function should return 6

**Problem Restatement**

We are given a string letters (mix of uppercase and lowercase English letters).

We must count how many **distinct letters** satisfy **both conditions**:

1. The letter appears **in both lowercase and uppercase**.
2. **All lowercase occurrences** of that letter come **before any uppercase** occurrence in the string.

**🧠 Example**

letters = "aaAbcCABBc"

* 'a': lowercase 'a's come before uppercase 'A' ✅
* 'b': lowercase 'b' before uppercase 'B' ✅
* 'c': lowercase 'c' appears *after* 'C' ❌  
  ✅ Count = 2

**⚙️ Approach**

For each letter 'a' → 'z':

1. Check if both lowercase and uppercase forms exist.
2. Find:
   * last\_lower: index of the **last lowercase** occurrence.
   * first\_upper: index of the **first uppercase** occurrence.
3. If last\_lower < first\_upper, condition is satisfied.

**✅ Python Implementation**

def solution(letters):

count = 0

for ch in 'abcdefghijklmnopqrstuvwxyz':

lower = ch

upper = ch.upper()

if lower in letters and upper in letters:

last\_lower = letters.rfind(lower)

first\_upper = letters.find(upper)

if last\_lower < first\_upper:

count += 1

return count