CSPP Remedial

Time: 2 hours Score: 25 Points

1. **Count Words (4 Points)**
   Implement a Python function `count_words(text)` that counts the number of occurrences of each word in a given string `text` and returns a dictionary with words as keys and counts as values.

   Test Cases:
   assert count_words("hello world hello") == {'hello': 2, 'world': 1}
   assert count_words("one two three two three three") == {'one': 1, 'two': 2, 'three': 3}
   assert count_words("") == {}

2. **Remove Adjacent Duplicates (5 Points)**
   Write a Python function `remove_adjacent_duplicates(s)` that takes a list `s` of integers and returns a new list where all adjacent duplicates have been reduced to a single element.

   Test Cases:
   assert remove_adjacent_duplicates([1, 2, 2, 3, 3, 5, 3, 4]) == [1, 2, 3, 5, 3, 4]
   assert remove_adjacent_duplicates([1, 1, 2, 3, 1, 5, 5, 5, 5, 6, 6]) == [1, 2, 3, 1, 5, 6]
   assert remove_adjacent_duplicates([1,1,1,1,1,1,1]) == [1]

3. **Transpose Matrix (5 Points)**
   Create a Python function `transpose(matrix)` that transposes a given 2-D list `matrix` (flips a matrix over its diagonal).

   Test Cases:
   assert transpose([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) == [[1, 4, 7], [2, 5, 8], [3, 6, 9]]
   assert transpose([[1, 2], [3, 4], [5, 6]]) == [[1, 3, 5], [2, 4, 6]]
   assert transpose([[1]]) == [[1]]

4. **String Compression (5 Points)**
   Implement a Python function compress_string(s) that performs basic string compression using the counts of repeated characters. For example, the string aabcccccaaa would become a2b1c5a3. If the compressed string would not become smaller than the original string, your function should return the original string. The function has to treat uppercase and lowercase letters as distinct characters.

   Test Cases:
   assert compress_string("aabcccccaaa") == "a2b1c5a3"
   assert compress_string("abcd") == "abcd" # "a1b1c1d1" is longer than "abcd"
   assert compress_string("aaAAaa") == "a2A2a2"
   assert compress_string("") == ""

5. **Maximum Subarray Sum (6 Points)**
   Implement the function `max_subarray_sum(nums)` that finds the maximum sum of a non-empty subarray (contiguous elements) from an array of integers `nums`.

   Test Cases:
   assert max_subarray_sum([-2,1,-3,4,-1,2,1,-5,4]) == 6 # [4,-1,2,1]
   assert max_subarray_sum([1]) == 1
   assert max_subarray_sum([-1,-2,-3]) == -1