Lab session 5: Programming Fundamentals

Problem 1: Pythagorean Triples

A *Pythagorean triple* (a, b, c) consists of three positive integers a, b, and c, such that $a < b$ and $a^2 + b^2 = c^2$. The name is derived from the Pythagorean theorem, stating that every right triangle has side lengths satisfying $a^2 + b^2 = c^2$. Clearly, if we scale a right triangle with side lengths a, b, and c by an integer $k > 1$, we find the Pythagorean triples $(k \cdot a, k \cdot b, k \cdot c)$. We call such triples *derived triples*. A Pythagorean triple which is not a derived triple is called a *primitive triple*.

A well-known primitive triple is $(3, 4, 5)$ since $3^2 + 4^2 = 5^2$. Two examples of its derived triples are $(6, 8, 10)$ and $(9, 12, 15)$.
Write a program that reads from the input a positive integer n, and outputs the number of primitive Pythagorean triples (a, b, c) such that $a + b + c = n$. You may assume that $1 \leq n \leq 30000 = 3 \times 10^4$.

Example 1:
input:
```
12
```
output:
```
1
```

Example 2:
input:
```
42
```
output:
```
0
```

Example 3:
input:
```
14280
```
output:
```
3
```

Problem 2: Takuzu Checker

A *Takuzu* is a number placement puzzle. The objective is to fill an 8×8 grid with 1s and 0s, where there is an equal number of 1s and 0s in each row and column (hence four 0s and four 1s) and no more than two of either number adjacent to each other. Moreover, there can be no identical rows, nor can there be identical columns. An example of a Takuzu puzzle and its solutions are given in the following figure.

0				**0**		**0**	
0					**1**		
			1			**0**	
		0		**0**			
							1
0		**0**		**0**	**0**		
						1	
	1						

0	0	1	1	**0**	1	**0**	1
0	1	1	0	0	**1**	1	0
1	0	0	**1**	1	0	**0**	1
1	1	**0**	1	**0**	0	1	0
0	0	1	0	1	1	0	**1**
0	1	**0**	1	**0**	**0**	1	1
1	0	1	0	1	0	**1**	0
1	**1**	0	0	1	1	0	0

Write a program that reads from the input a completely filled 8×8 grid of 0s and 1s. There are 8 input lines, one for each row. A row consists of 8 characters ('`0`' and '`1` '), followed by a newline ('`\n`'). Your program should output `CORRECT` if the grid satisfies all the rules of a Takuzu puzzle, otherwise it should output `INCORRECT`. Of cours e, your output must be a a single line, without spaces, than ends with a newline (`\n`).

Example 1:

input:
```
00110101
01100110
10011001
11010010
00101101
01010011
10101010
11001100
```
output:
```
CORRECT
```

Example 2:

input:
```
00110111
01100110
10011001
11010010
00101101
01010011
10101010
11001100
```
output:
```
INCORRECT
```

Example 3:

input:
```
00110101
01100110
10011001
11010010
00110101
01010011
10101010
11001100
```
output:
```
INCORRECT
```

Problem 3: Swapping data

The first line of the input for this problem contains a positive integer n. The next line contains a series of n integers. The remaining lines contain pairs i and j (where $0 \leq i < n$ and $0 \leq j < n$) representing swap operations to be performed on the series of numbers. For example, if $i = 0$ and $j = 3$, then the elements with indices 0 and 3 should be swapped. Note that indexing starts from index 0. The series of swaps is terminated by the pair `0 0`.

Your program should print `YES` if the series of numbers is ascending after performing the swap operations, otherwise it should print `NO`.

Example 1:

input:
```
5
3 1 2 0 4
0 3
0 0
```
output:
```
YES
```

Example 2:

input:
```
5
3 1 2 0 4
0 4
0 3
4 3
0 0
```
output:
```
YES
```

Example 3:

input:
```
6
3 1 1 2 0 4
0 4
0 3
4 3
0 0
```
output:
```
NO
```

Problem 4: DNA matching

A DNA string consists of the letters `'A'`, `'C'`, `'T'`, and `'G'`. The first line of the input for this problem consists of a DNA string, called the ‘pattern’. The next lin e contains a positive integer n. The following n lines contain a name, followed by a colon (:) and a DNA string called the ‘genome’. If the pattern is a substring of the genome, then your program should output the smallest index (called the ‘shift’) in the genome where the pattern matches. Moreover, the last output line should contain the total number of found matches. Make sure that your program produces output in the exact same format as given in the following examples.

Example 1:

input:
```
ACTG
2
Al Capone:GTCACTGAA
Don Corleone:CGTGTCACTGGACTGA
```
output:
```
Al Capone: shift=3
Don Corleone: shift=6
2 matches found.
```

Example 2:

input:
```
ACTG
2
Al Capone:GTCACGTGAA
Don Corleone:CGTGTCGAGCTGGACTGA
```
output:
```
Don Corleone: shift=13
1 match found.
```

Example 3:

input:
```
CTAG
2
Al Capone:GTCACTGAA
Don Corleone:CGTGTCACTGGACTGA
```
output:
```
0 matches found.
```

Problem 5: Array Reduction

The input for this problem is an array of positive integers. If there exists an element which equals the sum of all other elements in the array, then we remove it and keep repeating this process until no such element exists. The output of your program must be the number of elements in the maximally reduced array. For example, from the array `[16,3,4,1,8]` we remove the value `16`, (`16=3+4+1+8`) resulting in the reduced array `[3,4,1,8]`. Since `8=3+4+1` the array is reduced again to `[3,4,1]`. Since `4=3+1`, this array is reduced to `[3,1]`. This array cannot be reduced any further, so the output is `2`.

The input format of this problem is an integer n, which is the number of elements in the input array followed by a colon (`':'`) and the elements of the array.

Example 1:

input:
```
5: 16 3 4 1 8
```
output:
```
2
```

Example 2:

input:
```
5: 48 18 24 96 6
```
output:
```
2
```

Example 3:

input:
```
8: 22 9 44 2 1 5 88 5
```
output:
```
5
```