Programming Exercise 2: Ad Hoc Algorithm Design Writing
Individual

Instructor: Neil Patrick Del Gallego

[60 pts] General Instructions: This is a programming exercise for you to test your algorithmic design skills. For convention and consistency, the mini-program are to be done in JAVA. Please use only 1 java class for this exercise.

1. **[15 pts] Cryptanalysis**
Your task is to write a program which performs a simple analysis of a given text.

Input: The first line of input contains zero or more characters. This is the text which must be analyzed.

Output: Each line of output contains a single uppercase letter, followed by a single space, then followed by a positive decimal integer. The integer indicates how many times the corresponding letter appears in the input text. Upper and lower-case letters in the input are to be considered the same. No other characters must be counted. The output must be sorted in descending count order; that is, the most frequent letter is on the first output line, and the last line of output indicates the least frequent letter. If two letters have the same frequency, then the letter which comes first in the alphabet must appear first in the output. If a letter does not appear in the text, then that letter must not appear in the output.

Sample Input:	**Sample Output:**
Count me 1 2 3 4 5! Wow! I love ALGOCOM!	O 5 C 2 E 2 L 2 M 2 W 2 A 1 G 1 I 1 N 1 T 1 U 1 V 1

Coding Instructions
For #1, implement the solution in the function seen below. This function will be called from an external class (created by your instructor) to test your code.

```
public static void analyze (String input) {
   //put your code here and the print statements for the output.
}
```

2. **[15 pts] Train Swapping**
Your task is to write a program that given a certain number of train carriages and their corresponding carriage numbers, you are to output the optimal number of train swaps needed to order the carriages in ascending order (carriage 1, then carriage 2, then carriage 3, etc)

Input: There are two inputs. The first input is the length of the train (**N**). The second input of a test case contains a permutation of the numbers 1 through **L**, indicating the current order of the carriages. The carriages should be ordered such that carriage 1 comes first, then 2, etc. with carriage **L** coming last.

Output: The output must be the sentence: "**The most optimal train swaps: S"**, where **S** is the optimal number of swaps.

Sample Input:	**Sample Output:**
Input length of train: 4 Enter carriage numbers: 4 3 2 1 Input length of train: 2 Enter carriage numbers: 2 1 Input length of train: 4 Enter carriage numbers: 1 2 4 3	The most optimal train swaps: 6. The most optimal train swaps: 1 The most optimal train swaps: 1

Coding Instructions
For #2, implement the solution in the function seen below. This function will be called from an external class (created by your instructor) to test your code.

```
public static void countSwaps (int length, int[] carriageNumbers) {
   //put your code here and the print statements for the output.
}
```

3. [15 pts] Do we have enough harvest?

Winter is coming! The entire kingdom is planning if they have enough food to feed all the villages. Your job as the king's computer wiz is to write a problem that would tally the kingdom's harvest and see if there's enough food for everyone.

Input: There are two inputs. The first line contains two space-separated integers **N** (1 ≤ N ≤ 1000000) and **M** (1≤ M ≤ 5000), where **N** represents the number of villages to feed, and **M** being the cost to feed each village. The second input contains **N** space-separated integers indicating the total harvest for each village.

Output: Your program should output "**NOT ENOUGH FOOD**" if there's not enough food, "**JUST ENOUGH FOR EVERYONE**" if there's exactly that much for everyone, and "**PARTY!**" if there's surplus food.

Sample Input:	**Sample Output:**
5 500 100 0 300 100 100 5 500 1500 0 500 0 500 5 200 1000 100 100 100 100	NOT ENOUGH FOOD JUST ENOUGH FOR EVERYONE PARTY!

Coding Instructions

For #3, implement the solution in the function seen below. This function will be called from an external class (created by your instructor) to test your code.

```
public static void checkHarvest (int numVillages, int costToFeed, int[] harvests) {
   //put your code here and the print statements for the output.
}
```

4. [15 pts] What's the combination?

Create a program that generates the unique combination of numbers from **1** to **N** regardless of their order taken **K** elements. Combinations refer to the combination of **N** things taken **K** at a time without repetition. For a review of combination, check https://en.wikipedia.org/wiki/Combination.

Input: The input will contain two space-separated integers **N** (1 ≤ N ≤ 20) and **K** (1 ≤ K ≤ N).

Output: Your program should output several lines of each possible unique combination of numbers from 1 to **N** of at most **K** elements. Display all combinations in lexicographic order.

Sample Input:	**Sample Output:**
5 3	1 2 3 1 2 4 1 2 5 1 3 4 1 3 5 1 4 5 2 3 4 2 3 5 2 4 5 3 4 5
6 3	1 2 3 1 2 4 1 2 5 1 2 6 1 3 4 1 3 5 1 3 6 1 4 5 1 4 6 1 5 6 2 3 4 2 3 5 2 3 6 2 4 5 2 4 6 2 5 6 3 4 5 3 4 6 3 5 6

	4 5 6
4 3	1 2 3 1 2 4 1 3 4 2 3 4

Coding Instructions
For #4, implement the solution in the function seen below. This function will be called from an external class (created by your instructor) to test your code.

```
public static void printCombination (int N, int K) {
   //put your code here and the print statements for the output.
}
```

Submission Instructions

- Use only 1 Java class for this exercise. You are free to use as many functions, inner classes or structs to organize your code.
- Follow the coding instructions! Not following the coding instructions will merit deductions.
- Test your code thoroughly and try out different test cases, especially the edge cases (0 or N values). Your instructor will run with at least 3 test cases to check your code.
- Optimize your performance. If your code executes for more than 5 seconds, it will automatically fail the test case! Your code will be tested on an i7 processor with 8GB of RAM.
- Submit this exercise on or before the indicated deadline in Canvas.