**Problem Statement 1**

David, a data center manager, is tasked with efficiently assigning tasks to servers based on their processing times. To achieve this, he decides to use a **Min-Heap** to prioritize the servers with the lowest processing times. David inputs the number of servers and their respective processing times, ensuring that the heap maintains its property where the smallest element is always at the top.

After all servers are added, David needs to display the Min-Heap, showing the order in which tasks should be assigned for optimal efficiency.

**Input format :**

The first line contains an integer n, the number of servers in the heap.

The second line contains n space-separated integers representing the servers' processing times.

**Output format :**

The output prints the Min-Heap array after all elements have been inserted and the heap property has been maintained.

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

1 ≤ n ≤ 15

1 ≤ processing time ≤ 100

**Sample test cases :**

**Input 1 :**

9

5 3 8 2 9 1 7 6 4

**Output 1 :**

1 3 2 4 9 8 7 6 5

**Input 2 :**

5

10 20 30 40 50

**Output 2 :**

10 20 30 40 50

**Problem Statement 2**

Alex is a software engineer working on a project that involves processing user scores from an online gaming platform. He needs to create a **max heap** to efficiently manage these scores for ranking purposes. Additionally, Alex wants to analyze the performance of the top players by calculating the cube of their scores to get a better understanding of their performance metrics.

Your task is to help Alex by implementing a program that builds a **max heap** from the input scores and prints the cubed values of these scores.

**Input format :**

The first line contains an integer N, representing the number of scores Alex will input.

The second line consists of an N space-separated integers representing a user score.

**Output format :**

The first line displays the max heap in a single line, with scores separated by spaces.

The second line displays the cubed values of the scores in the same order as they were input.

**Refer to the sample output for exact format specifications.**

**Code constraints :**

The test cases will fall under the following constraints:

1 ≤ N ≤ 10

1 ≤ scores ≤ 30

**Sample test cases :**

**Input 1 :**

5

3 4 2 1 5

**Output 1 :**

5 4 2 1 3

125 64 8 1 27

**Input 2 :**

8

9 3 12 18 27 6 15 21

**Output 2 :**

27 21 15 18 12 6 9 3

19683 9261 3375 5832 1728 216 729 27

**Problem Statement 3**

Caleb loves to organize his collection of rare coins, and he wants to ensure that he can quickly find any coin in his collection. To do this, he decides to implement a **min-heap** data structure. Caleb inputs the number of coins and their respective values, inserting each value into the min-heap.

After building the heap, he checks if a specific coin is present in his collection. He prints the entire heap and whether the coin is found, helping him maintain his collection efficiently.

**Input format :**

The first line contains an integer n, representing the number of elements.

The second line contains n integers separated by spaces, representing the values of the coins.

The third line contains an integer C, representing the value of the coin to check for presence in the heap.

**Output format :**

The first line outputs the elements of the min-heap in sorted order, separated by spaces.

The second line indicates whether the specified element is present in the heap, formatted as:

* "C is present" if the element is found.
* "C is not present" if the element is not found.

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

1 ≤ n ≤ 15

1 ≤ value ≤ 1000

1 ≤ C ≤ 1000

**Sample test cases :**

**Input 1 :**

4

10 5 15 20

6

**Output 1 :**

5 10 15 20

6 is not present

**Input 2 :**

5

8 3 12 7 25

7

**Output 2 :**

3 7 12 8 25

7 is present

**Problem Statement 4**

John is organizing a programming competition where participants need to input numbers into a system that automatically arranges them in a max heap. Additionally, John wants to keep track of how many of the entered numbers are even and how many are odd. After processing all the inputs, the system should display the numbers in **max heap** order, along with the count of even and odd numbers.

Help John implement this system by writing a program that builds a **max heap** and tracks the number of even and odd numbers.

**Input format :**

The first line contains an integer n, representing the number of elements John will input.

The second line contains n space-separated integers, representing the values John enters.

**Output format :**

The first line displays the max heap in a single line, with elements separated by spaces.

The second line displays the count of even elements in the heap.

The third line displays the count of odd elements in the heap.

**Refer to the sample output for formatting specifications.**

**Code constraints :**

1 ≤ n ≤ 10

1 ≤ values ≤ 1000

**Sample test cases :**

**Input 1 :**

6

75 24 63 56 19 53

**Output 1 :**

Max Heap: 75 56 63 24 19 53

Even elements count: 2

Odd elements count: 4

**Input 2 :**

5

85 60 20 56 70

**Output 2 :**

Max Heap: 85 70 20 56 60

Even elements count: 4

Odd elements count: 1

**Problem Statement 5**

Liam is a data analyst at a logistics company and needs to optimize the delivery routes based on the weight of packages. To do this, he decides to use a min heap to prioritize packages by their weight, ensuring that the lightest packages are processed first. After inserting all package weights into the heap, he wants to display the heap structure and calculate the average weight of the packages.

Write a program that constructs a **min heap** from an array of package weights entered by Liam, displays the resulting min-heap structure, and calculates the Average Weight = (Total Weight of Packages) / (Number of Packages) of the packages.

**Input format :**

The first line contains an integer n, representing the number of packages.

The second line contains n space-separated integers representing the weights of the packages.

**Output format :**

The first line should contain n space-separated integers representing the min heap structure.

The second line should display "Average: " followed by double value representing the average weight of the packages, formatted to two decimal places.

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

The given test cases will fall under the following constraints:

1 ≤ n ≤ 10

1 ≤ weights ≤ 100

**Sample test cases :**

**Input 1 :**

5

5 10 3 1 7

**Output 1 :**

1 3 5 10 7

Average: 5.20

**Input 2 :**

6

15 8 12 6 4 9

**Output 2 :**

4 6 9 15 8 12

Average: 9.00

**Input 3 :**

4

15 67 14 17

**Output 3 :**

14 17 15 67

Average: 28.25