1. Given a sequence of distinct house prices projected for each of the next 'n' years, find the minimum possible financial loss Alice can incur when buying the house in one year and selling it in a later year. Alice *must* sell the house at a loss (i.e., the selling price must be lower than the buying price).

   Input:

   - The first line contains an integer 'n', representing the number of years.
   - The second line contains 'n' space-separated integers, representing the projected house prices for each year.

   Output:

   - Output the minimum financial loss Alice can achieve.

   Constraints:

   - $2 \leq n \leq 2 \times 10^5$
   - $1 \leq price[i] \leq 10^{16}$

   - All prices must be distinct

   Test case:

   5

   20 15 8 2 12

   3

2. Given an array of integers of size n and a number 'm', find the largest possible remainder when the sum of any continuous subarray is divided by 'm'.

   Input:
   - First line contains integer n, representing the number of elements in array
   - Second line, space separated values of array elements
   - Third line, the value of m.

   Output:

   Maximum value of remainder

   Test case:

   3

   1 2 3

   2

1

3. Often, when a list is sorted, the elements being sorted are just keys to other values. For example, if you are sorting files by their size, the sizes need to stay connected to their respective files. You cannot just take the size numbers and output them in order, you need to output all the required file information.
   The *counting sort* is used if you just need to sort a list of integers. Rather than using a comparison, you create an integer array whose index range covers the entire range of values in your array to sort. Each time a value occurs in the original array, you increment the counter at that index. At the end, run through your counting array, printing the value of each non-zero valued index that number of times.
   For example, consider an array arr = [1,1,3,2,1]. All of the values are in the range [0…3], so create an array of zeroes, result[0,0,0,0]. The results of each iteration follow:

i	arr[i]	result
0	1	[0, 1, 0, 0]
1	1	[0, 2, 0, 0]
2	3	[0, 2, 0, 1]
3	2	[0, 2, 1, 1]
4	1	[0, 3, 1, 1]

   Now we can print the sorted array: sorted =[1,1,1,2,3].

Input:

- The first line contains an integer n, the length of arr.
- The next line contains space-separated integers arr[i], where 0<=i<=n .

Output:

- Print the sorted list as a single line of space-separated integers.

4. Maximum Value in Knapsack

You are given N items, each with a weight and a value, and a knapsack that can carry a maximum weight of W. Your task is to determine the maximum total value that can be obtained by selecting a subset of these items, ensuring that the total weight does not exceed W.

Each item can either be **included or excluded** (0/1 Knapsack).

Input: N = 3, W = 50

weights = [10, 20, 30]

values = [60, 100, 120]

Output: 220

Explanation: Pick items with weights 20 and 30 (100 + 120 = 220).

Input: N = 2, W = 5

weights = [4, 5]

values = [1, 2]

Output: 2

Explanation: Pick the second item (value 2).

Constraints:

- 1 ≤ N ≤ 1000
- 1 ≤ W ≤ 10^4
- 1 ≤ weights[i] ≤ 10^4
- 1 ≤ values[i] ≤ 10^4

5. Minimum Cost of Encoding

You are given a list of N different characters, each with a corresponding frequency of occurrence. Your task is to determine the **minimum cost** required to encode all the characters using **Huffman Coding**.

The cost of encoding is calculated as the **sum of all merge operations** where two lowest-frequency nodes are combined in a **binary tree**. The cost of merging two nodes is equal to the sum of their frequencies.

Input: N = 5

```
       characters = ['a', 'b', 'c', 'd', 'e']

       frequencies = [10, 20, 30, 40, 50]


Output: 190

Explanation: The minimum cost of merging operations is 190.


Input: N = 4

       characters = ['x', 'y', 'z', 'w']

       frequencies = [5, 7, 12, 20]


Output: 44
```

Constraints:

- 1 ≤ N ≤ 10^5
- 1 ≤ frequencies[i] ≤ 10^6
- The sum of all frequencies will not exceed 10^9.

6. You are given a list of N different characters, each with a corresponding frequency of occurrence. You need to construct a **binary prefix encoding** using Huffman Coding and return **the sum of all encoded string lengths** multiplied by their respective frequencies. Each character's encoded length is determined by its depth in the **Huffman tree** (i.e., the number of bits used in its encoding). Your task is to **return the weighted sum of all encoded string lengths**.

   Input:

N = 4

characters = ['a', 'b', 'c', 'd']

frequencies = [5, 7, 10, 20]

Output: 82

Input:

N = 3

characters = ['x', 'y', 'z']

frequencies = [4, 8, 12]

Output: 44

Constraints:

- `1 ≤ N ≤ 10^5`
- `1 ≤ frequencies[i] ≤ 10^6`
- The sum of all frequencies will not exceed `10^9`.
- The characters are distinct.