Problem 1

You need to write two versions of the Matrix Chain Multiplication Algorithm.

1. Matrix Chain Multiplication using Memoization.
2. Bottom up iterative approach for Matrix Chain Multiplication using Tabulation.

Given the dimension of a sequence of matrices in an array P[], where the dimension of the ith matrix is (P [i-1] * P[i]), the task is to find the most efficient way to multiply these matrices together such that the total number of element multiplications is minimum.

Sample Input:

P = [7, 20, 30, 10, 8];

Sample Output:

(((AB)C)D)

Optimal Cost is : 6860

For bottom-up approach, print both the tables.

Sample Input:

P= [8, 45, 35, 52, 9, 78, 43,4556, 23]

Sample Output:

(((((((AB)C)D)E)F)G)H)

Optimal Cost is : 2468920

For bottom-up approach, print both the tables.

Problem 2:

Elvendale's history is divided into two major epochs: The Dawn Era and The Twilight Era. Each ChronoStone not only has an age value but also belongs to one of these epochs. The Dawn Era's stones have positive age values, while The Twilight Era's stones have negative age values.

The elves always arrange the stones chronologically within each epoch, but the Dawn Era always precedes The Twilight Era. A "ChronoInversion" occurs if:

1. A stone from The Dawn Era is found after a stone from The Twilight Era.
2. Within the same epoch, a younger stone is found before an older stone.

Elandrial, with the help of her magical staff, has ensured that there's at least one stone from each epoch. Now, she needs to know the number of unique ChronoInversions in the arrangement.

Constraints:

- There's always at least one stone from each epoch.
- The output pairs of indices should be unique and sorted in ascending order based on the first index of each pair.

Input Format:

- The first line contains a single integer n (2 ≤ n ≤ 10^5) — the number of ChronoStones.
- The second line contains n space-separated integers $C_1, C_2, \ldots, C_n$ (-10^9 ≤ C_i ≤ 10^9, $C_i \neq 0$) — the age values of the ChronoStones. Positive values are from The Dawn Era, and negative values are from The Twilight Era.

Output Format:

- The first line should contain a single integer — the number of unique ChronoInversions in the arrangement of ChronoStones.
- The subsequent lines should contain the ChronoInversions as pairs of indices, one pair per line, sorted in ascending order based on the first index of each pair.

Constraints:

- There's always at least one stone from each epoch.
- The output pairs of indices should be unique and sorted in ascending order based on the first index of each pair.

Sample Input: `[2000, -1500, 2500, -2300, 3000]`

Sample Output:

```
7
0 1
0 3
1 2
1 3
1 4
2 3
3 4
```

Sample Input: `[2000, 1500, 2500, 2300, 3000]`

Sample Output:

```
2
0 1
2 3
```

Sample Input: `[-2000, -1500, 2500, 2300, 3000]`

Sample Output:

```
7
0 2
0 3
0 4
1 2
1 3
1 4
2 3
```

Problem 3:

In the mystical land of Eldoria, there are ancient stones scattered across vast plains. Each stone emits a unique energy frequency, and legend has it that if two stones with nearly identical frequencies are brought close together, they can open portals to other dimensions.

To prevent unintentional portal activations, the Eldorian Council wants to identify the pair of stones that are closest to each other, so they can be safely relocated.

Given the coordinates of the stones, your task is to find the closest pair and determine the distance between them.

Input Format:

- The first line contains a single integer n (2 ≤ n ≤ 10^5) — the number of stones.
- The next n lines each contain two space-separated integers x_i and y_i (-10^4 ≤ x_i, y_i ≤ 10^4) — the x and y coordinates of the stones. No two stones will have the same coordinates.

Output Format:

- The first line should contain a single floating-point number — the minimum distance between any pair of stones, rounded to 4 decimal places.
- The second and third lines should contain the x and y coordinates of the two stones that are closest to each other.

Sample Input:

```
[(0, 0), (3, 4), (4, 3), (5, 12), (12, 5)]
```

Sample Output:

```
(1.4142135623730951, (4, 3), (3, 4))
```

Sample Input:

```
[(0, 0), (3, 4), (4, 30), (5, 12), (12, 5), (12, 500), (53, 5)]
```

Sample Output:

```
5.0000
0 0
3 4
```

Sample Input:

```
[(10, 50), (3, -4), (4, -30), (5, 12), (12, 5), (12, 500), (53, 5)]
```

Sample Output:

```
9.8995
12 5
5 12
```