A. A Quintic Factor

Most high school students are taught to use the quadratic formula to solve a quadratic equation. It is also possible to solve a cubic equation or a quartic equation using formulas (very ugly formulas). However, no one seems to be able to come up with a formula to solve a quintic (degree 5) polynomial. In fact, a young and bright French mathematician, Evariste Galois, proved that it is impossible to come up with such a formula!

As such, solving a quintic polynomial is seen as an impossible task… or is it?

Given integers a, b, c, d, e, f that satisfies $0 \le a, b, c, d, e, f \le 200$, solve the following equation:

$$ax^5 + bx^4 + cx^3 + dx^2 + ex = f$$

You are guaranteed that $0 \le x \le 1$, and that the equation is solvable.

Instructions:

Complete the following method in ProblemA.java:

```
public static double solveEquation(int a, int b, int c, int d, int e, int f)
input : a, b, c, d, e, f as specified in the question
output: x, the solution to the equation.
```

Note that the solution must correct up to 4 decimal place.

Sample execution:

```
int a = 1, b = 2, c = 3, d = 4, e = 5, f = 4;
double ans = ProblemA.solveEquation(a, b, c, d, e, f);
// ans = 0.4975
```

P.S. Galois is the most badass mathematician ever. When he was 21, he got killed in a duel sparked by his romantic interest.

B. Binary Revenge

I have just shown you how to use Binary Search to look for a single element in a sorted array. However, we often encounter cases when there are duplicate elements in the sorted array. Instead of returning the index of the element, can you return the range of indices of the element in the array?

Instructions:

Complete the following method in ProblemA.java:

```
public static int[] binarySearchRange(int[] array, int x) {
input :     int[] array - Sorted array which may contain duplicates
            int x       - The element to be searched within array
output:     an integer array of 2 elements
            output[0] is the starting index of the range
            output[1] is the ending index of the range
            if the element is not in the array, return {-1, -1}
```

Sample execution:

```
int[] testArray = {1, 2, 3, 4, 4, 4, 5, 6, 7};
int[] solution = ProblemB.binarySearchRange(testArray, 4);
// solution = {3, 5}

int[] testArray = {1, 2, 3, 4, 4, 4, 5, 6, 7};
int[] solution = ProblemB.binarySearchRange(testArray, 8);
// solution = {-1, -1}

int[] testArray = {1, 2, 3, 4, 4, 4, 5, 6, 7};
int[] solution = ProblemB.binarySearchRange(testArray, 5);
// solution = {6, 6}
```