1A. Write pseudocode/algorithm for a recursive program to check if an array is an palindrome or not.

1B. Implement the pseudocode/algorithm to a Java recursive program for checking an array is an palindrome.

2A. Write pseudocode/algorithm for a recursive program to calculate the "Greatest Common Factor" between two integers.

*A much more efficient method is the **Euclidean algorithm**, which uses a division algorithm such as long division in combination with the observation that the gcd of two numbers also divides their difference. To compute gcd(48,18), divide 48 by 18 to get a quotient of 2 and a remainder of 12. Then divide 18 by 12 to get a quotient of 1 and a remainder of 6. Then divide 12 by 6 to get a remainder of 0, which means that 6 is the gcd. Note that we ignored the quotient in each step except to notice when the remainder reached 0, signalling that we had arrived at the answer. Formally the algorithm can be described as:*

gcd(a, 0) = a

If the arguments are both greater than zero then the algorithm can be written in more elementary terms as follows:

gcd(a, a) = a
gcd(a, b) = gcd(a – b, b), if a > b
gcd(a, b) = gcd(a, b – a), if b > a

2B. Implement the pseudocode/algorithm to a Java recursive program for calculating the "Greatest Common Factor" between two integers.

3A. Write pseudocode/algorithm for a recursive program to calculate the approximate zero for any function by Newton's method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

terminate the recursive program and return the calculated value when the difference of the two adjacent calculation of the approximate zero (or x_{n+1} and x_n) are less than the predefined margin of error ε (read as "EPSILON". **Set ε = 1e – 20 for all the following problems**).

3B. Implement the pseudocode/algorithm to a Java recursive program for calculating the approximate zero for any function by Newton's method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

terminate the recursive program and return the calculated value when the difference of the two adjacent calculation of the approximate zero (or x_{n+1} and x_n) are less than the predefined margin of error ε. (Note: In order to make it simple, you can use polynomial function as $f(x)$, such as $f(x) = x^3 - 3x^2 + 2$, then $f'(x) = 3x^2 - 6x = 3x(x-2)$.)

4A. Write pseudocode/algorithm for a recursive program to calculate the approximate zero for any continuous function over an interval [*a*, *b*] by applying the "*Intermediate Value Theorem*" . Terminate the recursive program and return the calculated value of zero $x = c$ when $|f(c)| < \varepsilon$. **(If function $f(x)$ is continuous over the interval [*a*, *b*] , $f(a)$ and $f(b)$ are of different sign at the end point of the interval [*a*, *b*] , then there exists at least one zero $x = c$ and $a < c < b$ such that $f(c) = 0$.)** (*Hint: you can check if* $\left| f\left(\frac{a+b}{2}\right)\right| < \varepsilon$ *, if so, then* $c = \frac{a+b}{2}$ *; otherwise, consider the smaller intervals* $\left[a,\ \frac{a+b}{2}\right]$, *and* $\left[\frac{a+b}{2},\ b\right]$)

4B. Implement the pseudocode/algorithm to a Java recursive program for calculating the approximate zero for continuous function $f(x) = x^3 - 3x^2 + 2$ over the closed interval [–1, 2] by applying the "*Intermediate Value Theorem*".

5A. Write pseudocode/algorithm for a recursive program to calculate the nested radical

$$\sqrt{a+\sqrt{a+\sqrt{a+\sqrt{a+\cdots}}}},$$

terminate the recursive program and return the calculated value when the two adjacent calculation of nested radical values are less than the predefined margin of error ε.

5B. Implement the pseudocode/algorithm to a Java recursive program for calculating the nested radical

$$\sqrt{a+\sqrt{a+\sqrt{a+\sqrt{a+\cdots}}}},$$

terminate the recursive program and return the calculated value when the difference of the two adjacent calculation of nested radical values are less than the predefined margin of error ε.

6A. Write pseudocode/algorithm for a recursive program to calculate the number of disk moving in *Tower of Hanoi* with *n* disks.

6B. Implement the pseudocode/algorithm to a Java recursive program for calculating the number of disk moving in *Tower of Hanoi* with *n* disks.

6C. Mathematically derive the number of disk moving in *Tower of Hanoi* from result of your recursive program for a few inputs, such as $n = 5$, $n = 8$, and $n = 10$. Refer to

https://en.wikipedia.org/wiki/Tower_of_Hanoi