Ricorsione

Numero triangolare

Iterativa

```
public class TriangularNumbers {
    // Metodo iterativo per calcolare il numero triangolare
    public static int triangularIterative(int n) {
        int sum = 0;
        for (int i = 1; i ≤ n; i++) {
            sum += i;
        }
        return sum;
    }

    // Metodo main per testare la funzione iterativa
    public static void main(String[] args) {
        int n = 5; // Cambia questo valore per testare con un numero
diverso
        System.out.println("Il numero triangolare di " + n + " è: " +
triangularIterative(n));
    }
}
```

Ricorsiva

```
public class TriangularNumbers {
    // Metodo ricorsivo per calcolare il numero triangolare
    public static int triangularRecursive(int n) {
        if (n == 0)
            return 0;
        else
            return n + triangularRecursive(n - 1);
    }

    // Metodo main per testare la funzione ricorsiva
    public static void main(String[] args) {
```

```
        int n = 5; // Cambia questo valore per testare con un numero
diverso
        System.out.println("Il numero triangolare di " + n + " è: " +
triangularRecursive(n));
    }
}
```

Parole palindrome

Iterativa

```
public class Palindrome {
    // Metodo iterativo per verificare se una parola è un palindromo
    public static boolean isPalindromeIterative(String word) {
        int start = 0;
        int end = word.length() - 1;
        // Continua fino a quando start è minore di end
        while (start < end) {
            // Se i caratteri alle posizioni start ed end non
corrispondono, non è un palindromo
            if (word.charAt(start) != word.charAt(end))
                return false;
            // Avanza start e diminuisci end
            start++;
            end--;
        }
        // Se il ciclo è completato senza ritornare false, allora è un
palindromo
        return true;
    }

    // Metodo main per testare la funzione iterativa
    public static void main(String[] args) {
        String word = "radar"; // Cambia questa parola per testare con una
diversa
        if (isPalindromeIterative(word))
            System.out.println(word + " è un palindromo.");
        else
            System.out.println(word + " non è un palindromo.");
    }
```

```
}
```

Ricorsiva

```
public class Palindrome {
    // Metodo ricorsivo per verificare se una parola è un palindromo
    public static boolean isPalindromeRecursive(String word) {
        // Caso base: una stringa vuota o di lunghezza 1 è un palindromo
        if (word.length() ≤ 1)
            return true;
        // Verifica se il primo e l'ultimo carattere sono uguali
        if (word.charAt(0) ≠ word.charAt(word.length() - 1))
            return false;
        // Chiamata ricorsiva con la sottostringa escludendo il primo e
l'ultimo carattere
        return isPalindromeRecursive(word.substring(1, word.length() -
1));
    }

    // Metodo main per testare la funzione ricorsiva
    public static void main(String[] args) {
        String word = "radar"; // Cambia questa parola per testare con una
diversa
        if (isPalindromeRecursive(word))
            System.out.println(word + " è un palindromo.");
        else
            System.out.println(word + " non è un palindromo.");
    }
}
```

Conteggio numeri

Iterativo

```
public class IterativeCounter {
    // Metodo iterativo per contare da 1 a N
    public static void countIterative(int n) {
        for (int i = 1; i ≤ n; i++) {
            System.out.print(i + " ");
```

```
        }
    }

    // Metodo main per testare la funzione iterativa
    public static void main(String[] args) {
        int N = 5; // Cambia questo valore per testare con un numero
diverso
        System.out.print("Conteggio iterativo da 1 a " + N + ": ");
        countIterative(N);
    }
}
```

Ricorsivo

```
public class RecursiveCounter {
    // Metodo ricorsivo per contare da 1 a N
    public static void countRecursive(int n) {
        if (n > 0) {
            countRecursive(n - 1);
            System.out.print(n + " ");
        }
    }

    // Metodo main per testare la funzione ricorsiva
    public static void main(String[] args) {
        int N = 5; // Cambia questo valore per testare con un numero
diverso
        System.out.print("Conteggio ricorsivo da 1 a " + N + ": ");
        countRecursive(N);
    }
}
```

Altri esercizi

Torre di Hanoi

Il problema della Torre di Hanoi è un classico esempio di problema ricorsivo. L'obiettivo è spostare tutti i dischi da un piolo di partenza a un altro piolo, rispettando le seguenti regole:

1. È possibile spostare solo un disco alla volta.
2. È possibile spostare un disco solo se si trova in cima a un piolo.
3. Un disco più grande non può essere posizionato sopra un disco più piccolo.

Il problema può essere risolto utilizzando la ricorsione.

1. Scrivi un programma Java che implementi la soluzione ricorsiva del problema della Torre di Hanoi.
2. Crea poi la versione iterativa

Ricorsiva

```
public class HanoiTowerRecursive {
    // Metodo ricorsivo per risolvere il problema della Torre di Hanoi
    public static void hanoiRecursive(int n, char from, char to, char aux)
{
        if (n == 1) {
            System.out.println("Sposta il disco 1 da " + from + " a " +
to);
            return;
        }
        hanoiRecursive(n - 1, from, aux, to);
        System.out.println("Sposta il disco " + n + " da " + from + " a "
+ to);
        hanoiRecursive(n - 1, aux, to, from);
    }

    // Metodo main per testare la funzione ricorsiva
    public static void main(String[] args) {
        int n = 3; // Numero di dischi
        System.out.println("Soluzione Ricorsiva:");
        hanoiRecursive(n, 'A', 'C', 'B');
    }
}
```

Iterativa

```
import java.util.Stack;

public class HanoiTowerIterative {
```

```java
    // Metodo iterativo per risolvere il problema della Torre di Hanoi
    public static void hanoiIterative(int n, char from, char to, char aux)
{
        Stack<Integer> source = new Stack<>();
        Stack<Integer> destination = new Stack<>();
        Stack<Integer> auxiliary = new Stack<>();

        // Riempie il piolo di partenza con i dischi
        for (int i = n; i ≥ 1; i--) {
            source.push(i);
        }

        // Calcola il numero totale di mosse
        int totalMoves = (int) (Math.pow(2, n) - 1);

        // Se il numero di dischi è dispari, scambia piolo ausiliario e
piolo di destinazione
        if (n % 2 ≠ 0) {
            char temp = aux;
            aux = to;
            to = temp;
        }

        // Esegue le mosse in base al numero totale di mosse
        for (int i = 1; i ≤ totalMoves; i++) {
            if (i % 3 == 1) {
                moveDisc(source, destination, from, to);
            } else if (i % 3 == 2) {
                moveDisc(source, auxiliary, from, aux);
            } else if (i % 3 == 0) {
                moveDisc(auxiliary, destination, aux, to);
            }
        }
    }

    // Metodo per spostare un disco da un piolo all'altro
    public static void moveDisc(Stack<Integer> from, Stack<Integer> to,
char fromPole, char toPole) {
        if (!from.isEmpty() && (to.isEmpty() || from.peek() < to.peek()))
{
            to.push(from.pop());
            System.out.println("Sposta il disco " + to.peek() + " da " +
fromPole + " a " + toPole);
```

```
        } else if (!to.isEmpty() && (from.isEmpty() || to.peek() <
from.peek())) {
            from.push(to.pop());
            System.out.println("Sposta il disco " + from.peek() + " da " +
toPole + " a " + fromPole);
        }
    }

    // Metodo main per testare la funzione iterativa
    public static void main(String[] args) {
        int n = 3; // Numero di dischi
        System.out.println("Soluzione Iterativa:");
        hanoiIterative(n, 'A', 'C', 'B');
    }
}
```

Generazione di permutazioni

Scrivi un programma Java che generi tutte le possibili permutazioni di una stringa data. Una permutazione di una stringa è un riarrangiamento delle sue lettere. Ad esempio, le permutazioni della stringa "abc" sono: "abc", "acb", "bac", "bca", "cab" e "cba".

1. Scrivi una funzione ricorsiva che prenda una stringa come input e generi tutte le sue permutazioni.
2. Creane poi la versione iterativa

Ricorsiva

```
public class PermutationsRecursive {

    // Funzione ricorsiva per generare tutte le permutazioni di una
stringa
    public static void generatePermutations(String str) {
        generatePermutationsRecursive("", str);
    }

    private static void generatePermutationsRecursive(String prefix,
String remaining) {
        int n = remaining.length();
        if (n == 0) {
            System.out.println(prefix);
```

```
        } else {
            for (int i = 0; i < n; i++) {
                generatePermutationsRecursive(prefix +
remaining.charAt(i),
                        remaining.substring(0, i) + remaining.substring(i
+ 1, n));
            }
        }
    }

    // Metodo main per testare la funzione ricorsiva
    public static void main(String[] args) {
        String str = "abc";
        System.out.println("Permutations:");
        generatePermutations(str);
    }
}
```

Iterativa

```
public class PermutationsIterative {

    // Funzione iterativa per generare tutte le permutazioni di una
stringa
    public static void generatePermutations(String str) {
        int[] indexes = new int[str.length()];
        char[] chars = str.toCharArray();
        int n = str.length();

        while (true) {
            System.out.println(new String(chars));

            int i = n - 1;
            while (i > 0 && indexes[i - 1] ≥ indexes[i]) {
                i--;
            }

            if (i == 0) {
                break;
            }

            int j = n - 1;
```

```
            while (indexes[j] ≤ indexes[i - 1]) {
                j--;
            }

            swap(chars, i - 1, j);

            j = n - 1;
            while (i < j) {
                swap(chars, i, j);
                i++;
                j--;
            }
        }
    }

    // Metodo per scambiare due caratteri in un array di caratteri
    private static void swap(char[] chars, int i, int j) {
        char temp = chars[i];
        chars[i] = chars[j];
        chars[j] = temp;
    }

    // Metodo main per testare la funzione iterativa
    public static void main(String[] args) {
        String str = "abc";
        System.out.println("Permutations:");
        generatePermutations(str);
    }
}
```

Calcolo MCD

Massimo Comune Divisore (MCD) è il più grande numero intero che divide esattamente due numeri senza lasciare un resto.

L'algoritmo di Euclide è un metodo efficiente per calcolare il MCD di due numeri interi. Si basa sull'idea che il MCD di due numeri non cambia se sottraiamo il più piccolo dal più grande, finché entrambi i numeri sono diversi da zero. Quando uno dei numeri diventa zero, l'altro numero è il MCD dei due numeri originali.

Ricorsiva

```java
public class RecursiveGCD {
    // Funzione ricorsiva per calcolare il MCD di due numeri utilizzando
l'algoritmo di Euclide
    public static int gcdRecursive(int a, int b) {
        if (b == 0) {
            return a;
        } else {
            return gcdRecursive(b, a % b);
        }
    }

    // Metodo main per testare la funzione ricorsiva
    public static void main(String[] args) {
        int num1 = 48;
        int num2 = 18;
        int mcd = gcdRecursive(num1, num2);
        System.out.println("Il Massimo Comune Divisore di " + num1 + " e "
+ num2 + " è: " + mcd);
    }
}
```

Iterativa

```java
public class IterativeGCD {
    // Funzione iterativa per calcolare il MCD di due numeri utilizzando
l'algoritmo di Euclide
    public static int gcdIterative(int a, int b) {
        while (b ≠ 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    // Metodo main per testare la funzione iterativa
    public static void main(String[] args) {
        int num1 = 48;
        int num2 = 18;
        int mcd = gcdIterative(num1, num2);
        System.out.println("Il Massimo Comune Divisore di " + num1 + " e "
```

```
 + num2 + " è: " + mcd);
     }
 }
```

`