Exercise 1: repetition/perspektiv/tænkeopgave.

```
public static String a(String b) {
  ArrayList<String> c = new ArrayList<>();
  String[] d = b.split(" ");
  int e = 0;

  if (d != null && d.length > 0) {
    for (; e < d.length; e = e + 1) {
      c.add(0, d[e]);
    }
  }
  Object[] f = c.toArray();
  String[] g = new String[e];
  for(e = 0; e < f.length; e=e+1) {
    g[e] = f[e].toString();
  }
  String h = "";
        for(String i : g) {
    h = h.toString() + " " + i.toString();
    }
    return h.toString();
  }
```

1. Hvad gør metoden?
2. Hvad er redundant ved koden?
3. Er der noget ved ovenstående, der er oplagt ineffektivt?
4. Kunne navngivningen være bedre (og i så fald hvordan)?
5. Hvor kort og/eller effektiv kan du selv skrive en metode, med samme input/output?

End of Exercise 1

Exercise 2: Rekursion. What is the output generated by the following program. Find out before running the program.

```
public class Recursion01 {

    public static void main(String[] args) {
        whatever(12000);
    }

    private static void whatever(int i) {
        if (i > 0) {
            System.out.println(i);
            whatever(i / 2);
        }
    }
}
```

End of Exercise 2

Exercise 3: Rekursion. Ændr koden og få raketten til at starte på tallet nul. Skriv løsningen i hånden, før du checker på computeren.

```
public class Countdown {

    public static void main(String[] args) {
        countdown(10);
        System.out.println("Rocket startetd");
    }

    private static void countdown(int i) {
        System.out.println(i);
        countdown(i-1);
    }
}
```

End of Exercise 3

Exercise 4: Rekursion og iteration. Husk opgave 5.4:
Lav en klasse `GCD` med en offentlig metode `gcd`:
`public static int gcd(int a, int b)`
som tager to heltal som input og giver deres største fællesdivisor som output.
Man kan benytte Euklids algoritme:

$$\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$$
$$\text{GCD}(a, 0) = |a|$$

Lav to løsninger:

- En iterativ (løkkebaseret)
- En rekursiv (funktionskaldsbaseret)

End of Exercise 4

Exercise 5: Rekursion og effektivitet. Betragt følgende program:

```
public class Fibonacci {
  public static int fib(int n) {
    if(n<0) {
        throw new IllegalArgumentException("intet "+n+"'te Fibbonaccital");
    }
    if(n==0) return 1;
    if(n==1) return 1;
    return fib(n-1) + fib(n-2);
  }
}
```

Test programmet ved at kalde det med forskellige tal.
Hvornår begynder programmet at være virkelig irriterende langsomt?
Skriv et nyt og bedre rekursivt program: `public static int fibII(int n)`
Hint: Man kan bruge denne hjælpefunktion:

```
private static int[] fibHelper(int n) {
        // beregn og returner arrayet [F(n), F(n-1)]
        // hvor F(i) er det i'te Fibonaccital
}
```

End of Exercise 5

Exercise 6: Binær søgning opgave i rekursion og iteration
Følgende er en variant af obligatorisk opgave 2 delopgave 1.3.
Lav den offentlige metode `find(int holger, int[] array)` som returnerer hvorvidt holger findes i tabellen `array`.

```
public class BinarySearch {
  public static boolean find(int t, int[] a) {
        ...
  }
}
```

Opgavens kerne: Lav *både* en rekursiv *og* en iterativ løsning.

Bonusopgave: (∗) Udvid en af dine løsninger til en generisk variant, der håndterer flere typer:

```
public static<T extends Comparable<T>> boolean find(T t, T[] a) {
  ...
}
```

End of Exercise 6

Exercise 7: Flettesortering
I denne opgave laver vi vores eget rekursive program til at sortere arrays.
Programmet tager et array som input, og outputter et nyt, sorteret array.

```
public static int[] fletteSortering(int[] rodetArray){
  // 1. del i to halve
  // 2. løs hver halvdel rekursivt ved kald til fletteSortering(...)
  // 3. flet løsningerne sammen og returner
} // ?: hvor skal man checke for basistilfælde henne?
```

Basistilfældet er et array med 0 til 1 elementer: det er allerede sorteret.
Alle andre (større) arrays behandles på samme måde:

1. Split arrayet på midten og dan to nye arrays,
2. sorter hvert array rekursivt,
3. og flet de sorterede arrays sammen.

For at flette to arrays sammen kan man bruge løsningen fra obligatorisk opgave 2 delopgave 1.2 `merge`.
For at splitte kan man med fordel benytte Arrays.copyOfRange:

```
static int[] copyOfRange(int[] original, int from, int to)
```

fra `java.util.Arrays`.

Bonusspørgsmål: Hvor mange rekursive kald foretagers på et array af længde 8?

End of Exercise 7

Marge Sort