#Aufgabe 2

1. **[2,3,…,n-1,n,1]**

    - Der letze Eintrag (1) wird in `n` Schritte nach Anfang des Arrays verschiebt ###> `n` Vergleichen

2. **[1,2,1,2,…,1,2]**

    - Der zweiter einer-Eintrag (index Position 3) brauchen 1 Schritte wegen einer (2) Eintrag, der bevor steth. Der näßchste einer-Eintrag wurde 2 Vergleichen brauchen usw
    - d.h für den `n`-te einer-Eintrag, es wurde `n` Vergleichen brauchen, die Summe ist damit: ###> $\sum_{i=1}^{n} i$ Vergleichen

3. **[3,2,1,6,5,4,…,n,n-1,n-2]**

    - Es braucht 1 Vergleichen, um (2) und (3) zu tauschen, und noch 2 Vergleichen für (1) nach Anfang zu schieben
    - d.h um einer sortierte (1), (2), (3) zu erzeugen brauchen wir ingesamt 3 Vergleichen
    - Letzte Schleife ist (n), (n-1), (n-2), brauchen wir folglich ingesame 3*n Vergleichen

###> `3*n` Vergleichen

#Aufgabe 3

1. **f(n) = n^3/2**

    - **behauptung**: f(n) is O(n^2/3)

    ```
    c = 300, n = 100. für alle n>= 100: n^3/2 = 100^3/2 <= 300*100^2/3
    ```

    - **behauptung**: f(n) is Omega(n^2/3)

    ```
    c = 1, n = 1. für alle n>= 1: n^3/2 >= 1*n^2/3
    ```

    # => f(n) is Theta(g(n))

2. **f(n) = 10*n^2 + log(n)^2**

    - **behauptung**: f(n) is O(n^2)

    ```
    c = 300, n = 100. für alle n>= 100: 10n^2 +log(n)^2 = 10*100^2 +log(100)^2 <= 300*(n^2)
    ```

    # => f(n) is O(g(n))

3. **f(n) = 2^n**

    - **behauptung**: f(n) is Omega(2^(n+4)+3)

    ```
    c = 0.05, n = 0. für alle n>=10: 2^n = 2^10 >= 0.05(2^(n+4)+3)
    ```

    # => f(n) is Omega(g(n))

4. **f(n) = 2^n**

    - **behauptung**: f(n) is O(10n!)

    ```
    c = 0.2, n = 4. für alle n>=4: 2^n = 2^4 <= 0.2(10n!)
    ```

    # => f(n) is O(g(n))