Aufgabe 3.
1. In dem folgenden Java Ausdruck ist die Anzahl von p Aufrufen zu berechnen.

```
for (int i = 0; i < n; i++) {
      for(int j = i; j < n; j++) {
          a[j] = a[j] + p(i,j) + p(j,i);
     }
}
for (int j = 0; j < m; j++){
   a[j] = a[j] + p(j,j);
}
```

Es ist offensichtlich, dass die Aufgabe in 2 Haelfte aufgeteilt werden kann: In
1)

```
for (int i = 0; i < n; i++) {
      for(int j = i; j < n; j++) {
          a[j] = a[j] + p(i,j) + p(j,i);
     }
}
```

…und in
2)

```
for (int j = 0; j < m; j++){
   a[j] = a[j] + p(j,j);
}
```

Bei der zweiten Haelfte ist alles offensichtlich. Die Anzahl von Aufrufen ist linear und haengt von m ab.

Fuer die erste Haelfte hab ich die Fkt f (RandomArray.java) geschrieben, die Anzahl von p-Aufrufen in der Abhängigkeit von n aufzaehlt.

Die Tabelle, die ich mithilfe von dieser Fkt bekommen habe

n	0	1	2	3	4	5	6	7 ...
f(n)	0	2	6	12	20	30	42	56...

Das heisst, das f(n) = n*(n+1)

Dann die geschlossene Formel sieht folgender Weise aus:

g(n,m) = n*(n+1) + m.

2. Mit % bezeichne ich weiter matematisches „gehoert zu“
g(n,m) = n^2+n+m
fuer n>m gilt
g(n,m)%O(n^2), denn
n^2+n+m<=n^2+n^2=2*n^2, wobei C=2 aus der Def-on von O-Notation
fuer n<m gilt g(n,m)%O(n^2+m)

3. Der Speicherbedarf waechst quadratisch mit Wachstum von n und linear mit Wachstum von m.