Aufgabe 1. Lege ein Modul `mymath.c` / `mymath.h` an, in dem du die bisher geschriebenen Funktionen auslagerst.

Wir brauchen im folgenden eine Potenzfunktion, die zwei Fließkommazahlen als Argumente akzeptiert. Falls du diese Funktion gestern geschrieben hast, sollte sie jetzt im `mymath`-Modul verfügbar sein. Diese Funktion wird aber vermutlich zu langsam sein, daher gibt es die Funktion

```
double pow(double x, double y);
```

in der Systemheader `<math.h>`. Im Skript findest du bei Interesse im Anhang eine Referenz einiger Systembibliotheken.

Aufgabe 2. Implementiere die Riemann'sche Zeta-Funktion für $s \in \mathbb{N}$:

$$\zeta(s) := \sum_{k=1}^{\infty} \frac{1}{k^s}$$

Aufgabe 3. In dieser Aufgabe geht es um Sortieralgorithmen. Definiere dir ein Test-Array mit einer *festen* Anzahl von Einträgen mit denen du den Algorithmus testest. Lagere diesen Algorithmus noch nicht in eine Funktion aus (du weiß ja noch gar nicht, wie man Arrays an Funktionen übergibt).

a) Implementiere folgenden Sortieralgorithmus: Sortiere das kleines Element an die erste Stelle, dann das zweitekleinste Element an die zweite Stelle usw.

b) Der obige Sortieralgorithmus hat Komplexität $\mathcal{O}(n^2)$ (wobei n die Anzahl der Elemente ist). Aus theoretischer Sicht sind Sortieralgorithmen bis zu $\mathcal{O}(n \log(n))$ realisierbar. Wenn man nun aber die größer der zu sortierenden Eintrag einschränkt (z.B. sei die größte zu sortierende Zahl 20000) ist es sogar möglich einen *linearen* Sortieralgorithmus zu implementieren, also $O(n)$. Dazu stellt man sich für jede Zahl einen leeren "Bucket" (Korb) vor. Dann geht man die Liste der zu sortierenden Einträge durch und für ein Verkommen der Zahl k einen Ball in den k-ten Bucket. Danach geht man die Buckets vom ersten bis zu letzten durch. Ist am k-ten Bucket angekommen und es liegen j Bälle darin, dann schreibe sukzessive j mal die Zahl k in die zu sortierende Liste. Da die Bälle genau den zu sortierenden Zahlen entsprechen ist die Liste nachher sortiert.