Vanessa Wahlmüller

Inhaltsverzeichnis

Arbeitsaufwand: 7 h

Vanessa Wahlmüller

Beispiel 1

Lösungsidee

Es sollen zwei Klassen merge_sorter und file_manipulator implementiert werden. Die Klasse merge_sorter soll mit Hilfe des file_manipulators Dateien nach Art des Merge Sort Algorithmus sortieren. Der file_manipulator soll Dateien kopieren, sie mit Zufallswerten füllen, in eine oder mehrere Dateien aufspalten und den Inhalt der Datei ausgeben können. Durch dessen Verwendung soll der merge_sorter externe Datenmengen sortieren können.

Funktionsweise des Merge Sort Algorithmus:

Der Merge Sort Algorithmus nimmt einen Größeren Datensatz und partitioniert diesen. Dadurch verringert sich die Anzahl der zu vergleichenden Datensätze, bis nur zwei übrig sind, welche leicht miteinander verglichen werden können. Das Vergleichen von zwei Datensätzen findet beim erneuten zusammenfügen der Datensätze statt.

Beim externen vergleichen soll ein eingelesener Datensatz (aus einer Datei) in zwei Dateien aufgespalten werden. Der ganze Datensatz soll mithilfe von vier selbst erstellten Dateien sortiert werden. Beim erstmaligen Aufteilen werden in die Dateien jeweils der zweite Wert aus der Ursprungsdatei (getrennt durch Leerzeichen) platziert. Als nächstes werden jeweils die ersten beiden Werte aus beiden Dateien genommen verglichen und in eine der beiden weiteren Dateien gespeichert, dies passiert mit allen vorhandenen Werten. Bei ungerader Werte Anzahl wird der Rest einfach in der betreffenden Datei angefügt. Bei der nächsten Überprüfung werden mehr Werte verglichen, bis die Werte sortiert sind.

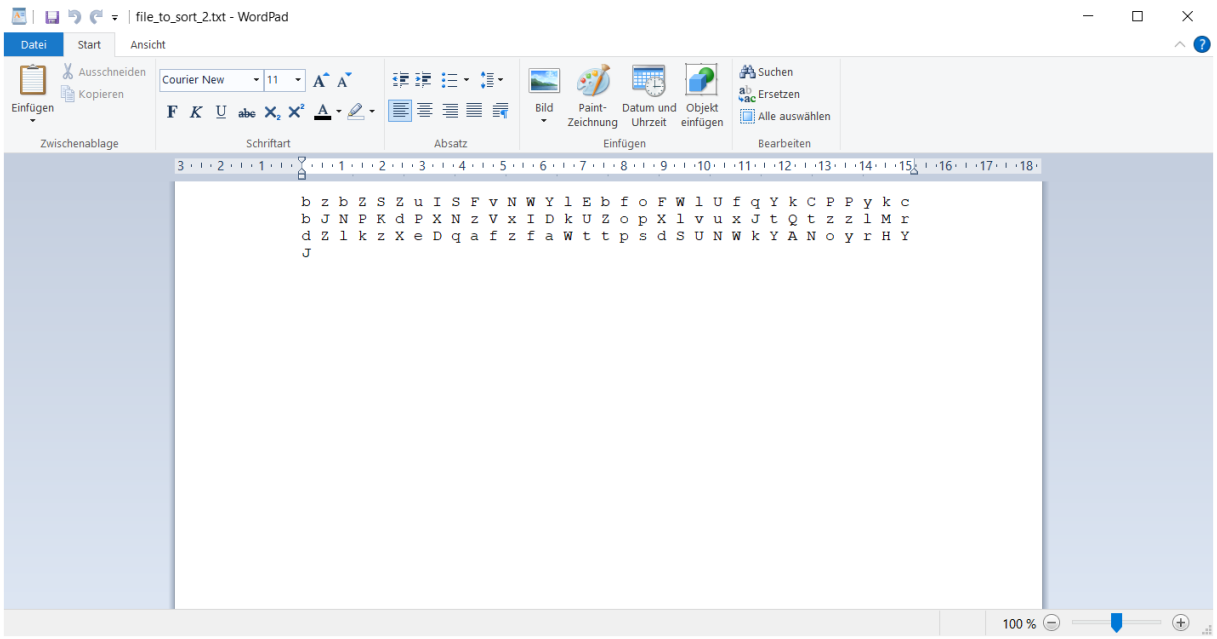
Vanessa Wahlmüller

Testfälle

```
void test_1();
```

Es werden zwei Dateien mit Zufälligen Werten erstellt. Danach wird die Funktion partition auf eine der Dateien angewendet. Diese teilt die Datei in zwei Dateien auf. Leerzeichen in verwendeten Dateien fungieren als Trennzeichen für die Aufteilung.

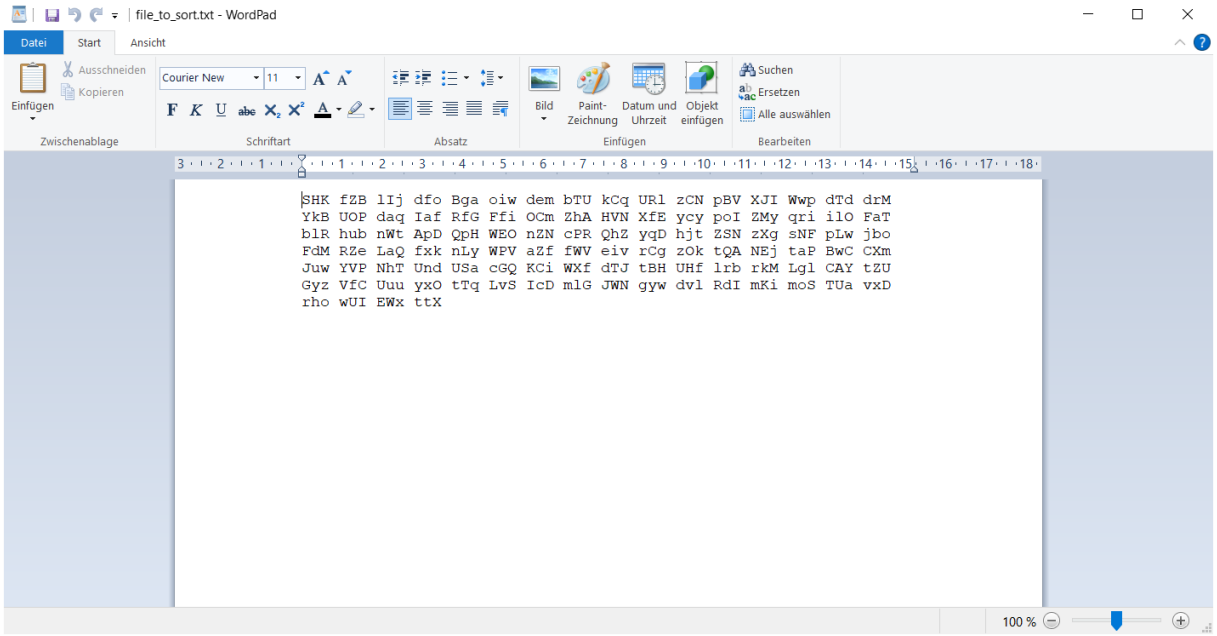
```
file_manipulator::cont_t out{ "f0.txt", "f1.txt" };
file_manipulator::fill_randomly("file_to_sort.txt", 100, 3);
file_manipulator::fill_randomly("file_to_sort_2.txt");//do not open with editor

file_manipulator::partition("file_to_sort.txt", out);
```



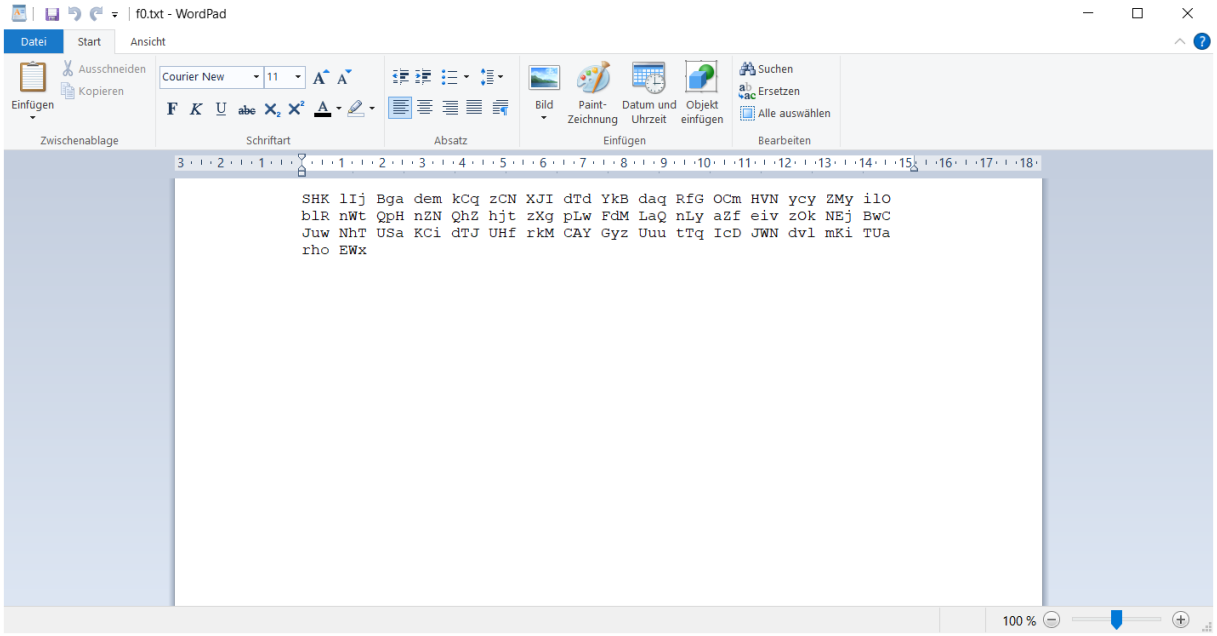
Arbeitsaufwand: 7 h

Vanessa Wahlmüller

Datei für partition:

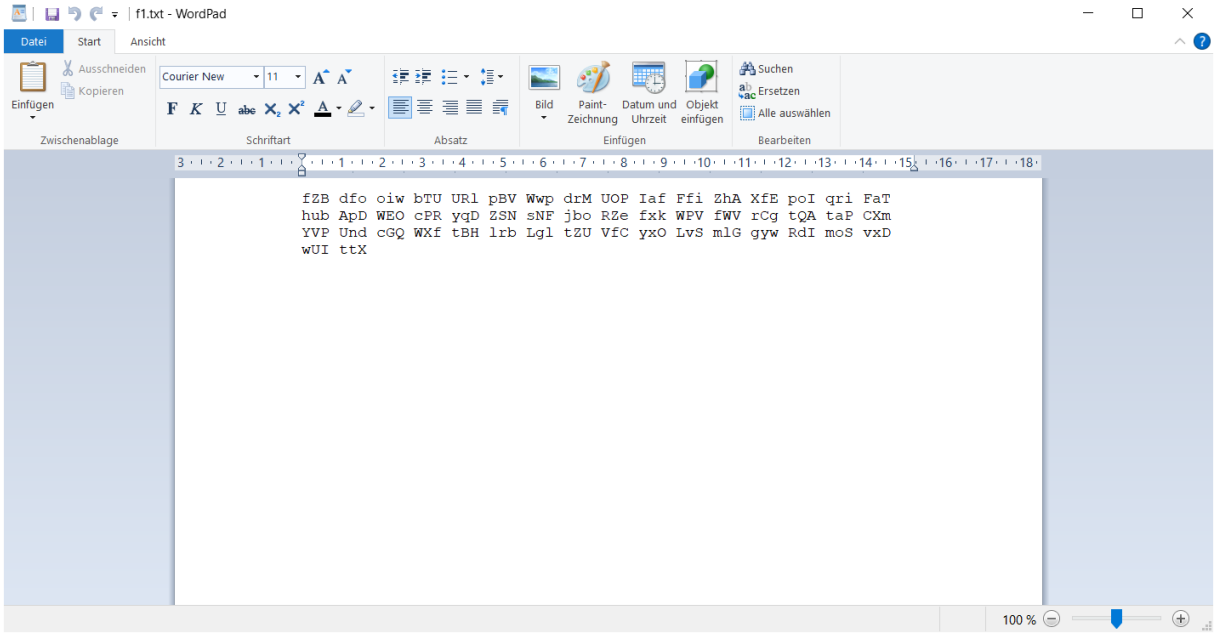


Dateien die nach der Aufteilung entstanden sind:



Arbeitsaufwand: 7 h

Vanessa Wahlmüller



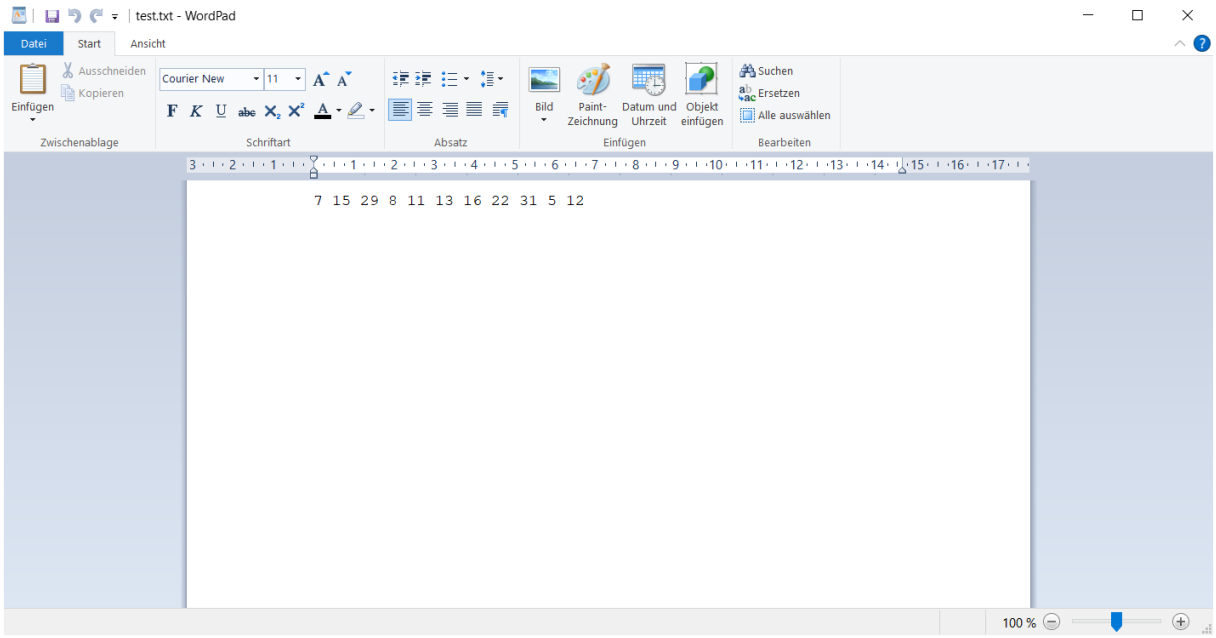
Arbeitsaufwand: 7 h

Vanessa Wahlmüller

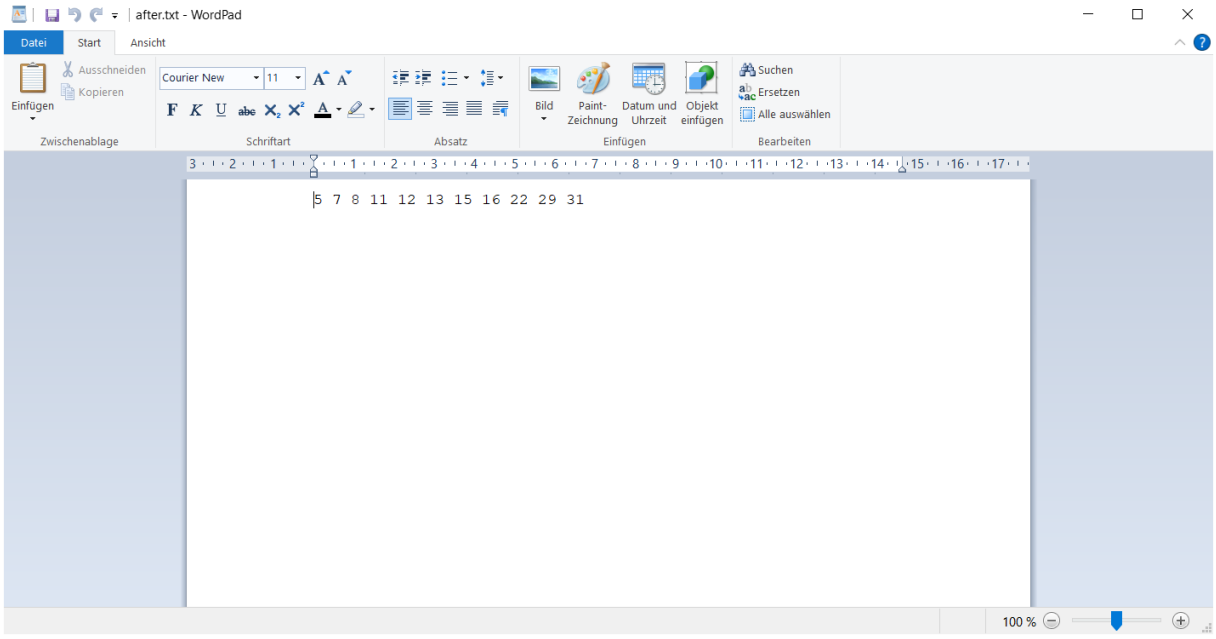
```
void test_2();
```

Es wird eine Datei eingelesen und durch die Funktion sort Mithilfe von vier eigenen Dateien sortiert. Die Funktion sort kopiert den fertig sortierten Datensatz in eine neue Datei und löscht die Hilfs Dateien.

Ausgangs Datei Test.txt:



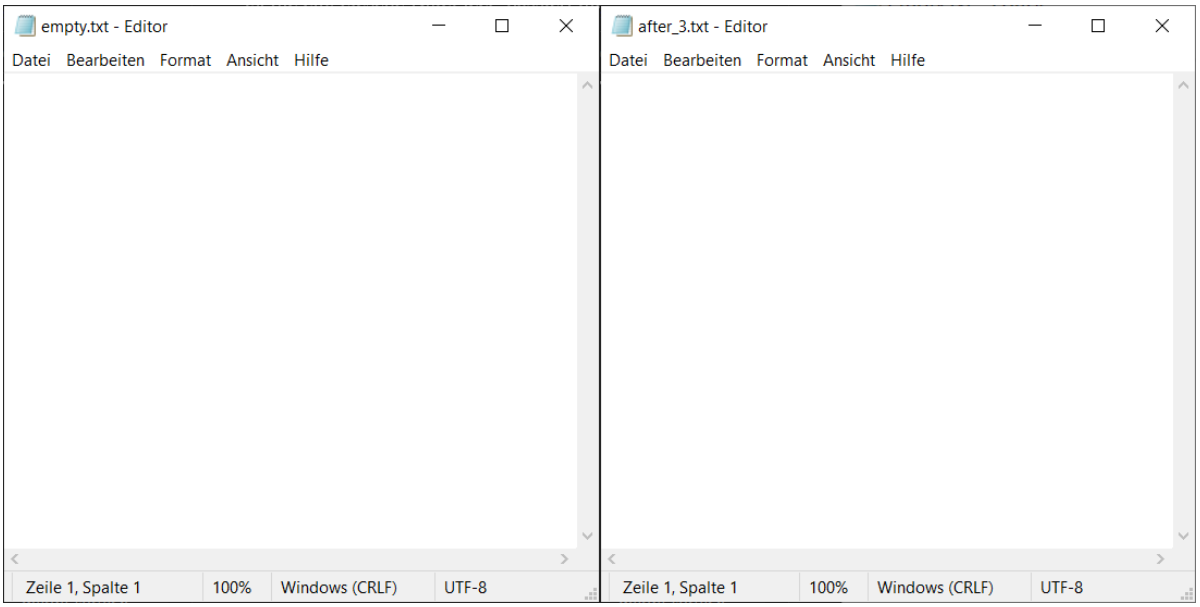
Fertig sortierte Datei:



Arbeitsaufwand: 7 h

`void test_3();`

Ist die eingelesene Datei leer, passiert nichts außer das eine leere after Datei erstellt wird.

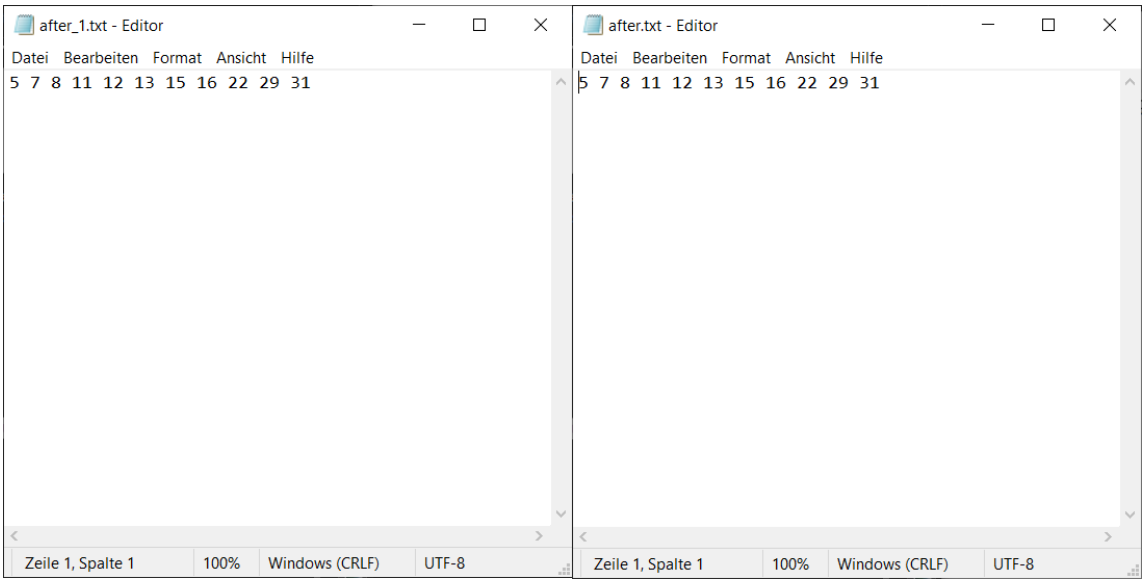


`void test_4();`

Tests mit Dateien die andere Werte als Zahlen aufweisen werfen Fehler aus und können nicht durchgeführt werden.

`void test_5();`

Test mit bereits sortierter Datei. Eingabe und Ausgabe sind gleich, es findet keine Sortierung statt.

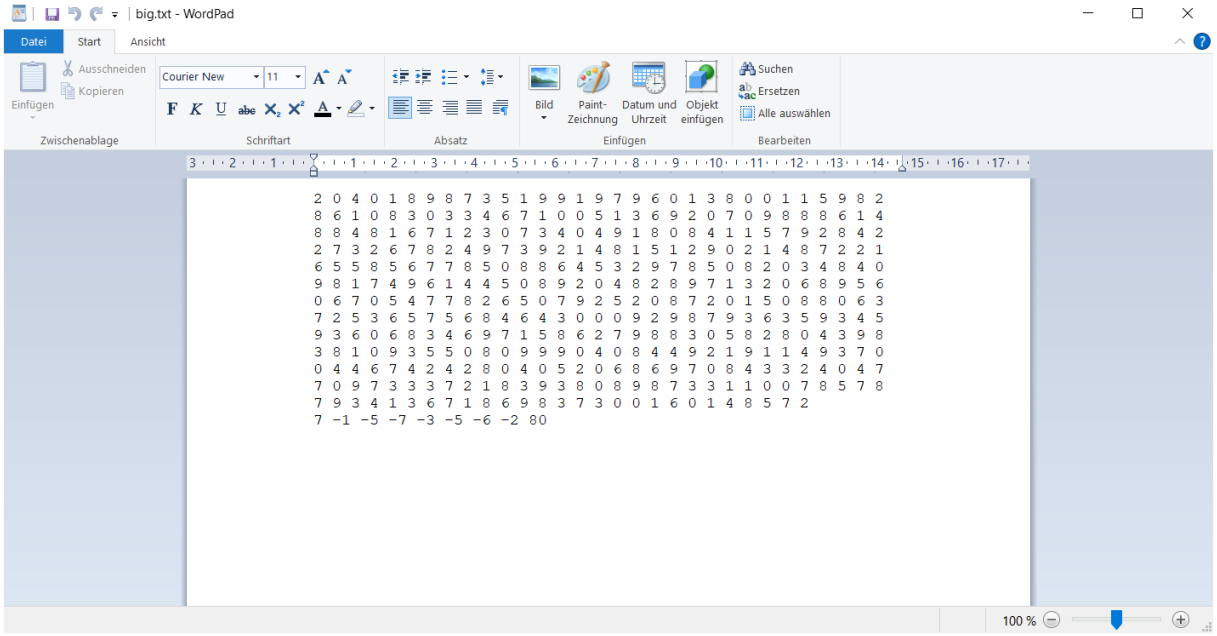


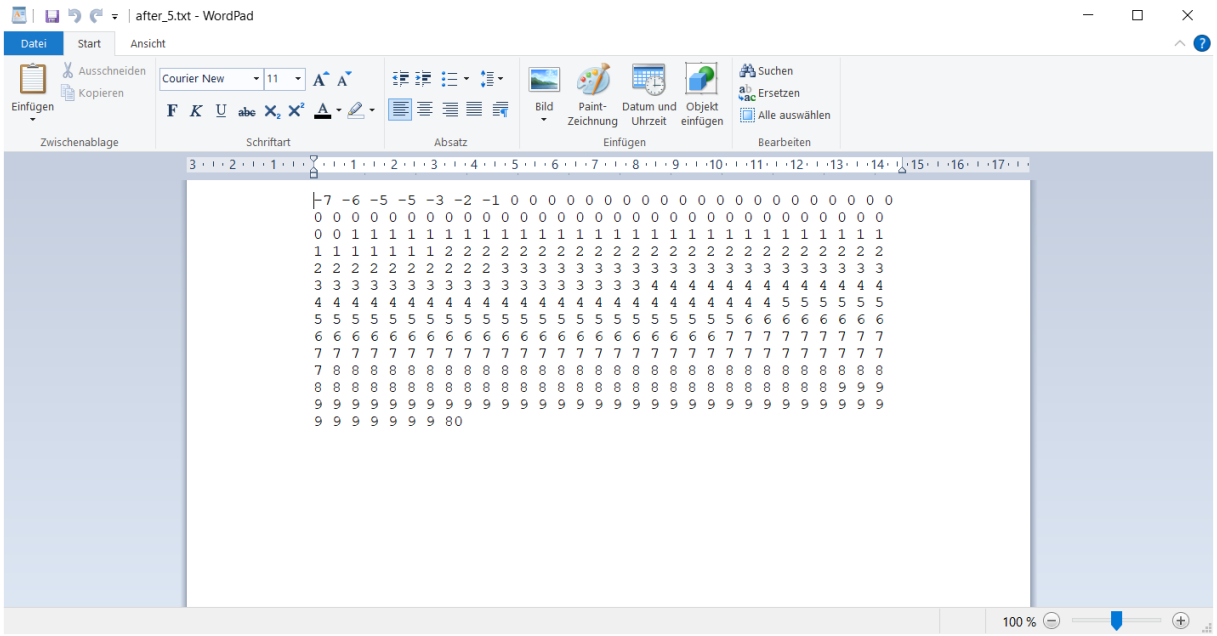
Arbeitsaufwand: 7 h

Vanessa Wahlmüller

```
void test_6();
```

Test des Sortierens einer größeren Datenmenge. Positive als auch negative Zahlen befinden sich in der Datei.





Arbeitsaufwand: 7 h