

Inhaltsverzeichnis:

1. External-Mergesort

- 1.1. Lösungsidee
- 1.2. Testfälle

1. External-Mergesort

1.1. Lösungsidee:

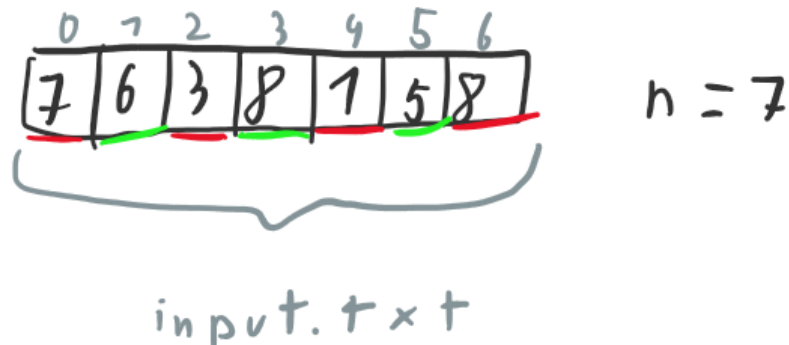
Dem externen Mergesort liegt der File Manipulator, welche verschiedene Dateioperationen durchführen kann, zugrunde:

- **Einlesen und in der Konsole ausgeben:** Eine Datei wird eingelesen und auf der Konsole ausgegeben. Zusätzlich werden die ausgegebenen „Gruppen“ (getrennt durch Leerzeichen oder Zeilensprünge) gezählt.
- **Datei kopieren:** Eine Datei wird kopiert, indem nacheinander kleinere Gruppen an Zeichen eingelesen und in eine neue Datei geschrieben werden.
- **Datei zufällig füllen:** Es wird eine Datei mit zufällig aufeinander folgenden Zeichen erstellt. Diese Zeichen sind entweder Groß- oder Kleinbuchstaben, welche zufällig generiert werden. Erst wird generiert, ob es sich um einen Groß- oder Kleinbuchstaben handelt, danach um welchen. Der User kann die Anzahl der Zeichen und die Größe der „Gruppen“ (getrennt durch Leerzeichen) wählen.
- **Datei partitionieren:** Eine Datei wird in mehrere Dateien partitioniert, indem die einzelnen Gruppen jeweils abwechselnd in die Ausgabefiles geschrieben werden. Bei 2 Partitionen wird die erste Gruppe in das erste File, die zweite in das zweite, die dritte wieder in das erste und so weiter geschrieben.
- **Inhalt alterierend in Dateien ausgeben:** Ähnlich wie beim Partitionieren wird ein Inhalt in Gruppen abwechselnd in verschiedene Files geschrieben. Die Gruppen können hier auch größer sein als beim Partitionieren.

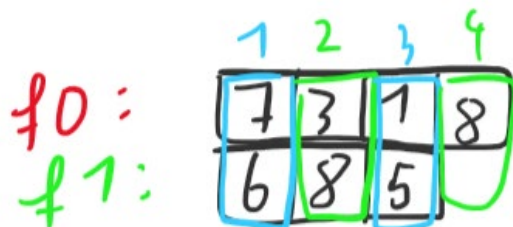
Der externe Mergesort dient dazu, größere Dateien, die nicht in den RAM passen würden, mithilfe von Partitionierung zu sortieren. Der User kann auswählen, ob er numerisch (nur Zahlen als Input möglich) oder nach Zeichen (1,12,3 anstatt von 1,3,12) aufsteigend oder absteigend sortieren möchte.

Festgelegt für das Programm ist, dass die Eingabedatei in 2 Partitionen unterteilt wird. Der User kann, wenn er möchte, den Namen der Ausgabedatei wählen. Es werden zwei Sätze von je 2 Partitionen benötigt. Als erster Schritt wird die Eingabedatei in den ersten Satz partitioniert. Falls die Datei leer ist, wird eine Fehlermeldung ausgegeben.

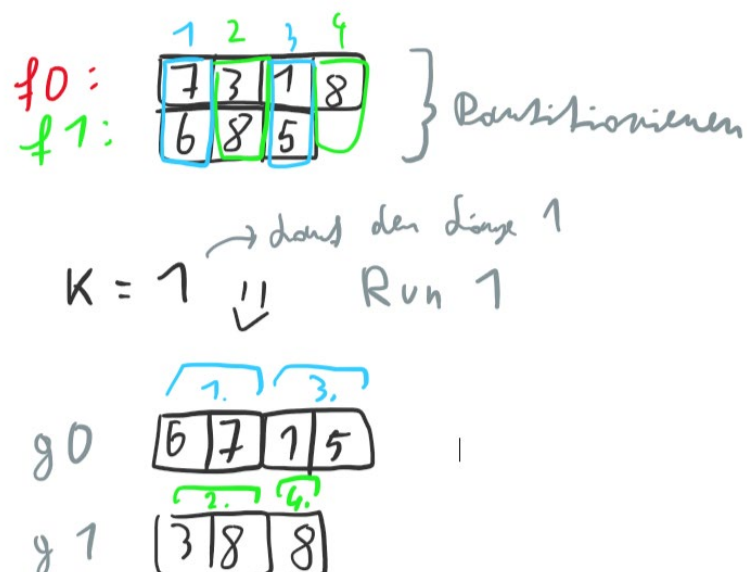
Das Sortieren findet in „Runs“ statt. Der erste Run beginnt mit dem Wert 1. Dieser Wert verdoppelt sich bei jedem Run. Am besten lässt sich die Funktionsweise des Algorithmus an einem Beispiel illustrieren.



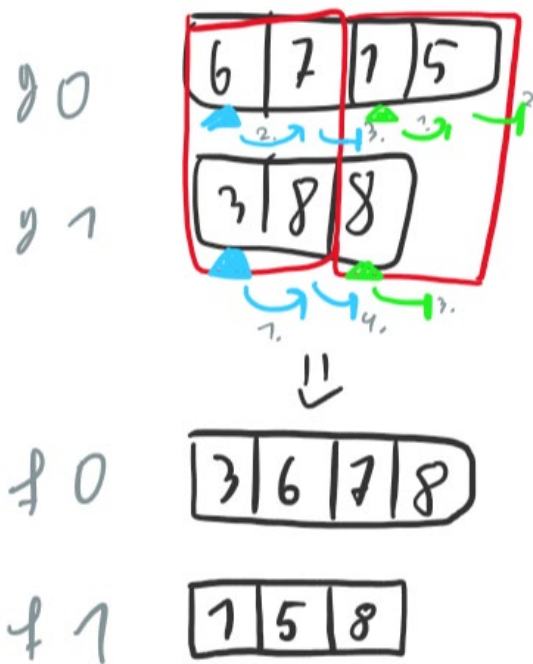
Die Ausgangsdatei umfasst 7 Werte und wurde folgendermaßen partitioniert (in f0 und f1).



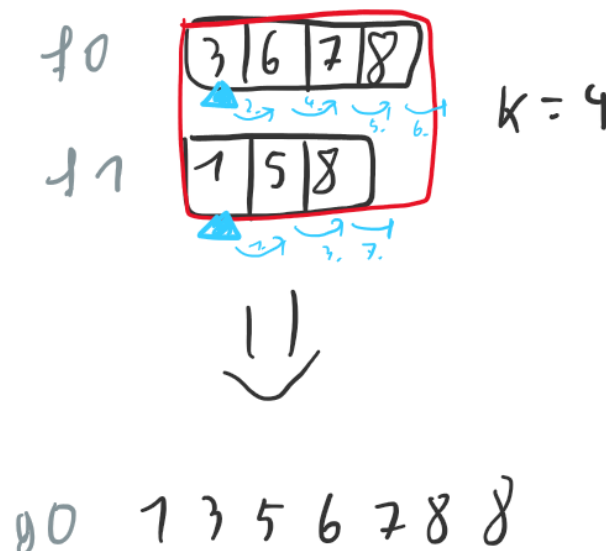
Der Wert des Runs gibt Aussage über die Anzahl der Elemente, die aus einer der Dateien entnommen werden. Wie durch die grünen und blauen Felder illustriert, werden das erste Element aus File 1 und das erste Element aus File 2 „gruppiert“. Diese beiden Elemente werden in das erste File im zweiten Satz der Partitionen (hier: g0, g1) geschrieben, wobei das kleinere Element als erster in die Datei geschrieben wird. Es wird von aufsteigender Sortierreihenfolge ausgegangen.



Beim nächsten Run beträgt der Wert 2 ($1 \cdot 2$). Es werden aus jeder Datei im zweiten Satz je 2 Zeichen verglichen und in eine Datei aus dem ersten Satz geschrieben. Da die Zeichen einer Datei, die in einem Run eingelesen werden (hier: 2) untereinander sortiert sind, kann man wie durch die grünen und blauen Pfeile gezeigt mittels jeweiliger Indizes vorgehen und diese um 1 erhöhen, wenn die kleinere Zahl aus der zum Index gehörigen Datei stammt.



Nach einem Wert von 4 beim nächsten Lauf ist die Datei vollständig sortiert. Da die Datei in Zyklen eingelesen wird, die dem Wert des Laufs entsprechen (also immer nur die Dateien aus beiden Dateien die „gruppiert“ werden), ist der Datensatz nach dem Lauf, der nur einen Zyklus benötigt, um alle Werte einzulesen, fertig sortiert.




Beim Einlesen der Werte für einen Lauf, ist wichtig darauf zu achten, dass beim letzten Zyklus des Laufs die Gruppe nicht „vollständig“ sein muss, also die zweite Datei weniger Werte als die erste Datei oder sogar keine Dateien im dem letzten Zyklus enthält. Die Container sind dementsprechend anzupassen, um zu vermeiden, dass 0 statt der fehlenden Werte in den Datensatz eingefügt wird.

Das Sortieren der Werte eines Einlesezyklus funktioniert folgendermaßen: Aus den beiden Wertegruppen aus den Files soll eine aufsteigende Reihenfolge in den nächsten Partitionensatz geschrieben werden. Es werden die ersten Indizes verglichen, der kleinere Wert wird an die erste Position geschrieben und der Index der Werte aus dieser Datei springt zum nächsten Wert. Wenn die Indizes einer Datei erschöpft sind, werden automatisch alle Werte der anderen Datei nacheinander (da sie untereinander im Run (Wert des Runs = n-viele Zeichen hintereinander in einer Datei sind sortiert) geordnet sind) geschrieben bis auch diese Indizes erschöpft sind. Dann wird der nächste Zyklus des Runs eingelesen und in die andere Datei des zum Schreiben benutzen Partitionensatzes geschrieben.

Neben den Einlesezyklen eines Runs, die wie oben erwähnt bestimmen, wann der Datensatz fertig sortiert ist, wird auch die Anzahl der Runs gezählt. Dadurch ob diese gerade oder ungerade ist, kann man feststellen in welchem Partitionensatz der fertig sortierte Datensatz (immer in der ersten Partition) steht. Diese Partition wird kopiert und anschließend in der Konsole ausgegeben.


1.2. Testfälle:

Fall 1: Leere Datei

 Microsoft Visual Studio Debug Console

```
ERROR: File could not be sorted. Ensure your file contains more than 0 values and meets the required standards.
```


Fall 2: Datei existiert nicht

 Microsoft Visual Studio Debug Console

```
ERROR: No compatible file found.
```

```
ERROR: File could not be sorted. Ensure your file contains more than 0 values and meets the required standards.
```

Fall 3: Leerer Eingabedateiname


 Microsoft Visual Studio Debug Console

```
ERROR: No compatible file found.
```

Fall 4: Leerer Ausgabedateiname

```
ERROR: Non-valid name for destination file.
```

Fall 5: Beispiel aus Lösungsidee

 Microsoft Visual Studio Debug Console

```
Original dataset:
```


```
7 6 3 8 1 5 8
```

```
Sorted dataset:
```

```
1 3 5 6 7 8 8
```

Das Beispiel wird richtig sortiert und ausgegeben (aufsteigend, numerisch)

Fall 6: Kein Ausgabedateiname

 Microsoft Visual Studio Debug Console

```
Original dataset:
```

```
7 6 3 8 1 5 8
```

```
Sorted dataset:
```

```
1 3 5 6 7 8 8
```

Die Datei wird unter einem Default-Dateinamen gespeichert, es wird trotzdem korrekt sortiert.

In folgenden Testfällen wird jeweils (außer es ist explizit anders erwähnt) aufsteigend und numerisch sortiert.

Fall 7: Nicht nur einzelne Zeichen oder Zahlen bei numerischem Sortieren

Microsoft Visual Studio Debug Console

```
Original dataset:  
6 5 4 3 2 7 8 1 5 Das ist ein Testfall 7 8 9 1 t 5 a d f 5 6 g  
ERROR: Numeric sort is activated, ensure your file only contains numeric values.
```

Fall 8: Menge mit 1 Element sortieren

Microsoft Visual Studio I

```
Original dataset:  
7  
Sorted dataset:  
7
```

Fall 9: Menge mit 2 Elemente sortieren

Microsoft Visual Studio

```
Original dataset:  
12 4  
Sorted dataset:  
4 12
```

Fall 10: Menge mit 3 Elementen sortieren


Microsoft Visual Studio Debu

```
Original dataset:  
19 5 7  
Sorted dataset:  
5 7 19
```

Fall 11: Rückwärts sortierte Menge sortieren


Microsoft Visual Studio Debug Console

```
Original dataset:  
20 17 16 15 14 13 10 9 8 7 6 5 4 3 2 1  
Sorted dataset:  
1 2 3 4 5 6 7 8 9 10 13 14 15 16 17 20
```

Fall 12: Bereits sortierte Menge sortieren Microsoft Visual Studio Debug Console


```
Original dataset:
1 2 3 4 5 6 7 8 9 10 13 14 15 16 17 20

Sorted dataset:
1 2 3 4 5 6 7 8 9 10 13 14 15 16 17 20
```

Fall 13: Menge mit mehreren doppelten Werten sortieren Microsoft Visual Studio Debug Console


```
Original dataset:
5 5 6 6 7 8 9 1 2 4 5 2 3 4 5 1 2 3 4 5 6 7 8 7 8 7 8 9 8 0 7 8 9 8 7 6 3 4 7 5 6 8 7 4 2 1 5 2 3 6 5 4 5 7 8 6

Sorted dataset:
0 1 1 1 1 2 2 2 2 2 3 3 3 3 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 8 9 9 9
```

Fall 14: Integer und Character gemischt sortieren mit aktiviertem numerischem Sortieren Microsoft Visual Studio Debug Console


```
Original dataset:
a g 5 6 t r 4 j 8 9 7 5 g r e w a 3 4 n b g a s t 4 e r 7 8 9 7 6 3 4 5 6 2 1 2 3 4 1 j k l m n a d

ERROR: Numeric sort is activated, ensure your file only contains numeric values.
```

Fall 15: Sortieren mit negativen Zahlen Microsoft Visual Studio Debug Console

```
Original dataset:
-50 53 45 9 1 -151 -5 -3 0 5 -4 -6 -7 21

Sorted dataset:
-151 -50 -7 -6 -5 -4 -3 0 1 5 9 21 45 53
```

Fall 16: Absteigend sortieren Microsoft Visual Studio Debug Console

```
Original dataset:
6 8 9 1 7 6 5 4 1 2 8 9 7 12 42 35 89 64 21 10 41 21 2 5 7 8

Sorted dataset:
89 64 42 41 35 21 21 12 10 9 9 8 8 8 7 7 7 6 6 5 5 4 2 2 1 1
```

Fall 17: Absteigend sortieren mit negativen Zahlen

```
Original dataset:
-50 53 45 9 1 -151 -5 -3 0 5 -4 -6 -7 21

Sorted dataset:
53 45 21 9 5 1 0 -3 -4 -5 -6 -7 -50 -151
```


Fall 18: Leerzeichen und Zeilenumbrüche in Ausgangsdokument

	6	8	7	8	5	5	34
5	5						
65		7					

```
Original dataset:
6 8 7 8 5 5 34 5 5 65 7

Sorted dataset:
5 5 5 5 6 7 7 8 8 34 65
```

Fall 19: Nach Zeichen sortieren (nicht numerisch, aufsteigend)

```
Microsoft Visual Studio Debug Console
Original dataset:
Hallo das ist ein Testfall mit verschiedenen Woertern und Zahlen die nicht numerisch sortiert werden. Einige weitere Testwoerter: Aejhdafl AAhfadsfjklad aafd 631 13 1 125 4 5 3 21 2 test

Sorted dataset:
1 125 13 2 21 3 4 5 631 AAhfadsfjklad Aejhdafl Einige Hallo Testfall Testwoerter: Woertern Zahlen aafd das die ein ist mit nicht numerisch sortiert test und verschiedenen weitere werden.
```

Wie man sehen kann, werden auch die Zahlen nicht numerisch sondern nach Zeichen sortiert. Großbuchstaben werden vor Kleinbuchstaben sortiert.

Fall 20: Nach Zeichen sortieren (nicht numerisch, absteigend)


```
Original dataset:
Hallo das ist ein Testfall mit verschiedenen Woertern und Zahlen die nicht numerisch sortiert werden. Einige weitere
Testwoerter: Aejhdafl AAhfadsfjklad aafd 631 13 1 125 4 5 3 21 2 test

Sorted dataset:
werden. weitere verschiedenen und test sortiert numerisch nicht mit ist ein die das aafd Zahlen Woertern Testwoerter
: Testfall Hallo Einige Aejhdafl AAhfadsfjklad 631 5 4 3 21 2 13 125 1
```


Fall 21: Veranschaulichung nicht-numerisches Sortieren mit Zahlen

```
Microsoft Visual Studio Debug Console
Original dataset:
7 8 9 4 5 6 7 1 2 3 4 5 12 14 17 124 1251235 12545 24 467 46 47 410 798 0 -124 -5 -123 -11 -10 -15 0


Sorted dataset:
-10 -11 -123 -124 -15 -5 0 0 1 12 124 1251235 12545 14 17 2 24 3 4 4 410 46 467 47 5 5 6 7 7 798 8 9
```


Fall 27: Ein Element aufteilen Microsoft Visual Studio Debug Console

```
Original file: 5  
Amount of partitioned values: 1  
Partition 1: 5  
Partition 2:
```

Fall 28: Mehrere Elemente aufteilen Microsoft Visual Studio Debug Console

```
Original file: 8 5 4 5 6 7 8 9 8 7 6 4 3 2 1 9 8 7 6 5 4  
Amount of partitioned values: 21  
Partition 1: 8 4 6 8 8 6 3 1 8 6 4  
Partition 2: 5 5 7 9 7 4 2 9 7 5
```

Fall 29: Leeres File aufteilen Microsoft Visual Studio Debug Console

```
Original file:  
Amount of partitioned values: 0  
Partition 1:  
Partition 2:
```