
SWE	Softwareentwicklung 3 Medizin- und Bioinformatik	WS 22/23, Übung 2
------------	---	------------------------------

Name: Lena Oppitz:**Aufwand in h:** 10**Punkte:** _____ **Kurzeichen Tutor/in:** _____

Inhalt

Beispiel 1 External-Mergesort	2
Lösungsidee	2
File_manipulator	2
Merge_sorter	2
Code	3
Testfälle	3
Fill randomly (file_manipulator)	3
Copy (file_manipulator)	3
sort (merge_sorter)	4

Beispiel 1 External-Mergesort

Lösungsidee

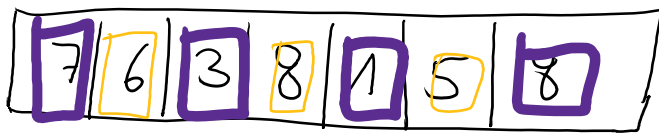
Mithilfe des External-Mergesorts können auch Datenmengen bzw. Files sortiert werden, die nicht alle im Speicher gleichzeitig Platz haben.

File_manipulator

Die Klasse file_manipulator managend die Files und soll es möglich machen möglichst einfach, Files zu teilen, kopieren und auszugeben.

Partition

Es muss immer ein Element aus dem source File gelesen werden. Dieses wird dann entweder in g0 oder in g1 gespeichert. Der Wechsel ist am einfachsten zu implementieren, wenn jedes Element mitgezählt wird und immer mit %2 auf 0 und 1 (Positionen im Vektor von * auf filestream) gewechselt wird. Diese Schleife soll solange wiederholt werden, bis das src File leer ist.



File g0
File g1

Copy

Durch das File iterieren und jedes Element bis das File leer ist in ein anderes File schreiben.

Print

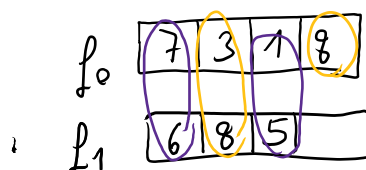
Immer einen Wert aus dem File einlesen und diesen auf der Konsole ausgeben. So lange bis das src File leer ist.

Merge_sorter

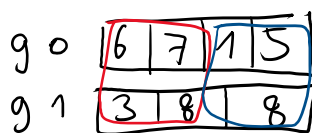
Sort

Diese Funktion soll durch unterschiedliche Funktionsaufrufe und einer Abfrage zwischen den Files wechseln. Die Funktion soll hier merge aufrufen.

Merge ① Partition

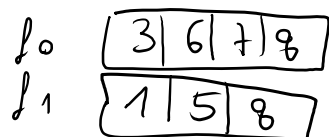


run length = 1



sort und in File schreiben

run-length = 2



1 Lauf length = 4
1 Restlauf 6

Es soll so lange gemerged werden, bis die Lauflänge so groß ist, wie die Anzahl der Elemente im Originalfile. Dann kann davon ausgegangen werden, dass alle Elemente sortiert sind. Pro Run soll immer in ein File geschrieben werden. Danach muss gewechselt werden (kann ähnlich umgesetzt werden wie bei Partition mit $\text{counter} \% 2$). Es soll immer ein Element aus dem 0er und eins aus dem 1er File gelesen werden. Diese müssen dann verglichen werden. Das Kleinere soll geschrieben werden und ein neues in diese Variable eingelesen werden, bis beide Files, bis die Lauflänge erreicht wird. Ist der Run vorbei müssen die jeweiligen Lauflängenzähler resetet werden. Da der lesende und der schreibende Zugriff immer getauscht werden, ist es sinnvoll die streams in Pointer auf Vektoren zu speichern. Das Standardzeichen des << oder >> Operators ist immer ein Leerzeichen.

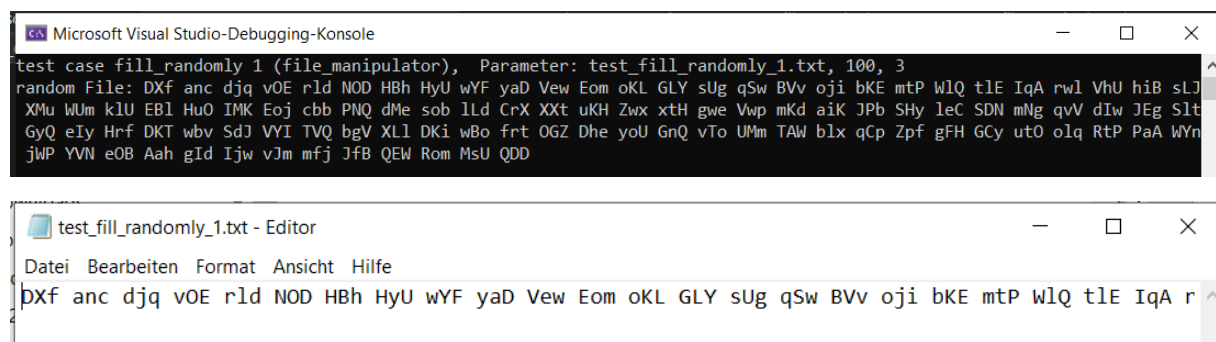
Code

SWE_Oppitz_UE02

Testfälle

Fill randomly (file_manipulator)

Testfall 1: test_fill_randomly_1.txt, 100, 3



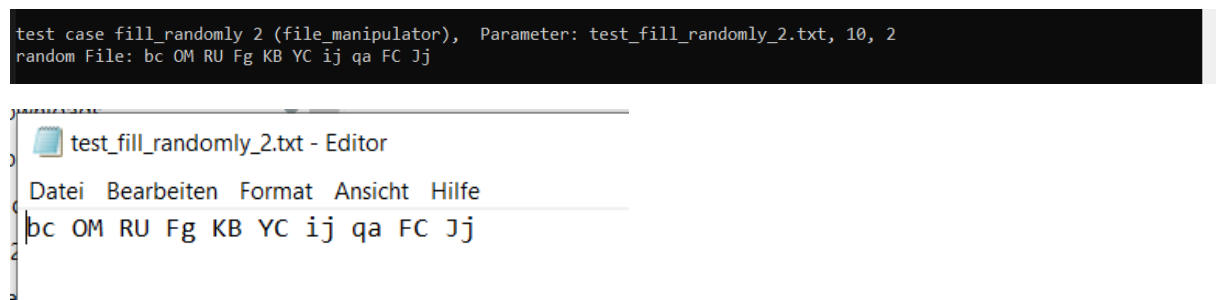
```

Microsoft Visual Studio-Debugging-Konsole
test case fill_randomly 1 (file_manipulator), Parameter: test_fill_randomly_1.txt, 100, 3
random File: DXf anc djq vOE rld NOD HBh HyU wYF yaD Vew Eom oKL GLY sUg qSw BVv oji bKE mtP WlQ tLE IqA rwl VhU hiB sLJ
XMU wUm kLU EBl HuO IMK Eoj cbb PNQ dMe sob lLd CrX Xxt uKH Zwx xth gwe Vwp mKd aiK JPb SHy leC SDN mNg qvV dIw JEg Slt
GyQ eIy Hrf DKT wbv SdJ VYI TVQ bgV XLl DKi wBo frt OGZ Dhe yoU GnQ vTo Umm TAW blx qCp Zpf gFH GCy utO olq RtP PaA WYn
jWP YVN eOB Aah gId Ijw vJm mfj JfB QEW Rom MsU QDD

test_fill_randomly_1.txt - Editor
Datei Bearbeiten Format Ansicht Hilfe
DXf anc djq vOE rld NOD HBh HyU wYF yaD Vew Eom oKL GLY sUg qSw BVv oji bKE mtP WlQ tLE IqA r

```

Testfall 2: test_fill_randomly_2.txt, 10, 2



```

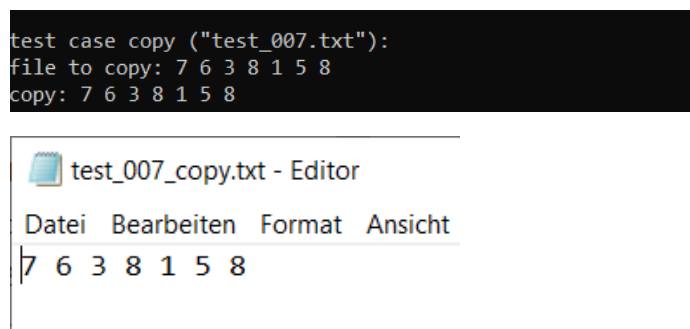
test case fill_randomly 2 (file_manipulator), Parameter: test_fill_randomly_2.txt, 10, 2
random File: bc OM RU Fg KB YC ij qa FC Jj

test_fill_randomly_2.txt - Editor
Datei Bearbeiten Format Ansicht Hilfe
bc OM RU Fg KB YC ij qa FC Jj

```

Copy (file_manipulator)

Testfall 1:



```

test case copy ("test_007.txt"):
file to copy: 7 6 3 8 1 5 8
copy: 7 6 3 8 1 5 8

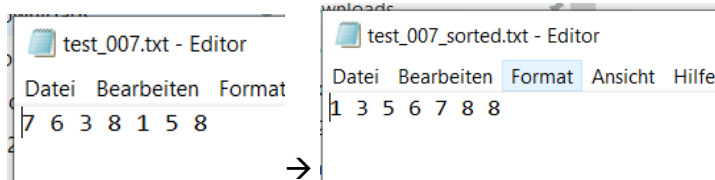
test_007_copy.txt - Editor
Datei Bearbeiten Format Ansicht
7 6 3 8 1 5 8

```

sort (merge_sorter)

Testfall 1: mit Zahlen und doppelten 8

```
test case sort 1 ("test_007.txt"):
File to sort : 7 6 3 8 1 5 8
Sorted File: 1 3 5 6 7 8 8
```



Testfall 2: nicht existierendes File

```
test case sort 2 ("not_existing_file.txt"):
File to sort :
```

Es wird kein neues File erstellt.

Testfall 3:

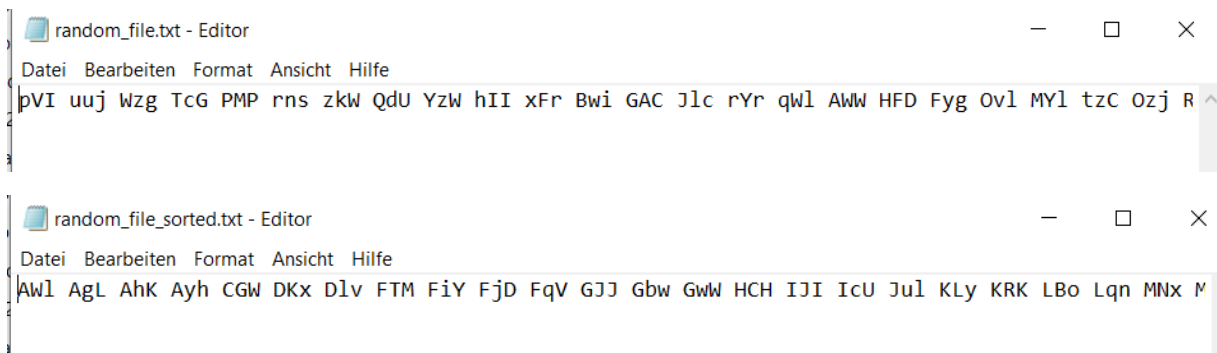
```
test case sort 3 ("file_is_empty.txt"):
File to sort :
```

Es wird kein neues File erstellt.

Testfall 4:

```
test case sort 4, gerade Zahl an Woertern ("random_file.txt"):
File to sort : pVI uuJ Wzg TcG PMP rns zkw QdU YzW hII xFr Bwi GAC Jlc rYr qWl AWW HFD Fyg OvI MYl tzC Ozj Rpo BfK JuX D
qR XnJ IUs pyC ILh olr KNV IyK jEC zDy AGt WIi LZc dvX Sfv dzP IkG sbb mSh ckJ LGY XvK Dqa DUm bdO Eth nTB fWI bTO bSN L
rk JsU JtG eaX mjZ vmc UYw TjX Pwd oeI DTX vvG afn Dxz Ljk jJv pJy NcI kqS NZS yVQ pyo dVB VzK gYJ iVr KJJ sLZ CbJ nOc C
OM quX TPR gBP dED yoF UMW rQk qAR MZX uIC rIB Set DNq

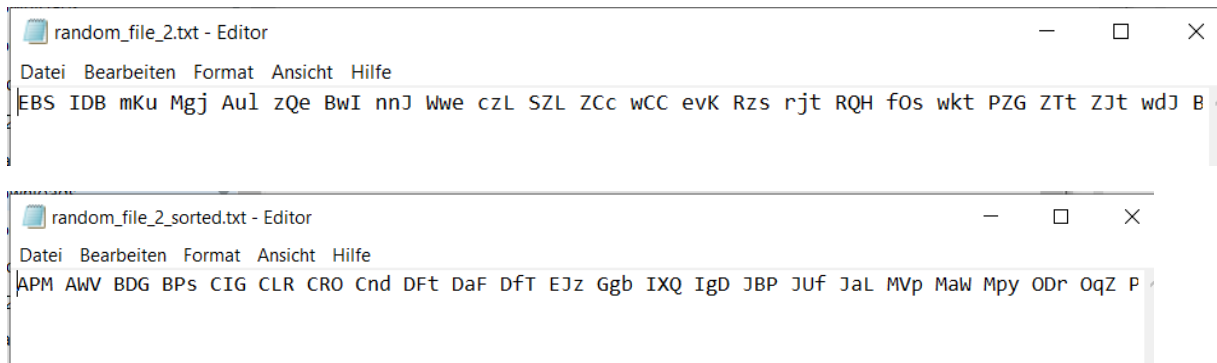
Sorted File: AGt AWW BfK Bwi COM CbJ DNq DTX DUm DqR Dqa Dxz Eth Fyg GAC HFD ILh IUs IkG IyK Jlc JsU JtG JuX KNV KJJ LGY
Ljk Lrk MYl MZX NZS NcI OvI Ozj PMP Pwd QdU Rpo Set Sfv TPR TcG TjX UMW UYw VzK WIi Wzg XnJ XvK YzW afn bSN bTO bdO ckJ
dED dVB dvX dzP eaX fWI gBP gYJ hII iVr jEC jJv kqS LZc mSh mjZ nOc nTB oeI olr pJy pVI pyC pyo qWl qAR quX rIB rQk rYr
rns sLZ sbb tzC uIC uuJ vmc vvG xFr yVQ yoF zDy zkw
```



Testfall 5:

```
test case sort 5, ungerade Anzahl an Woertern ("random_file_2.txt"):
File to sort : EBS IDB mKu Mgj Aul zQe BwI nnJ Wwe czL SZL ZCc wCC evK Rzs rjt RQH fOs wkt PZG ZTt ZJt wdJ BDs zcl IFx Q
xd eHK Zau QsY eTP yIf uHX haF TML Sgc Mjb aCB WAt VGI FWY bts KgQ KPH CUF iEj mpU rgW WrJ ErU XfO tAo grm tXg WTe avf h
yy Fng Xtq oSL SKh DoS gjD yYW bCB goa fsh vXU Jbt kPQ HZB Hxm mNC SNR lIh Zyh Teu orN ySL QSP zEX DOe ZfY ZRJ DWi Zin S
xZ AeC faX ged OYQ YkF zgd dVs vzj VBm OkE bwv Dxx SvM mbg

Sorted File: AeC Aul BDs BwI CUF DOe DWi DoS Dxx EBS ErU FWY Fng HZB Hxm IDB IFx Jbt KPH KgQ Mgj Mjb OYQ OkE PZG QSP QsY
Qxd RQH Rzs SKh SNR SZL Sgc SvM SxZ TML Teu VBm VGI WAt WTe WrJ Wwe XfO Xtq YkF ZCc ZJt ZRJ ZTt Zau ZfY Zin Zyh aCB avf
bCB bts bwv czL dVs eHK eTP evK fOs faX fsh ged gjD goa grm haF hyy iEj kPQ lIh mKu mNC mbg mpU nnJ oSL orN rgW rjt tAo
tXg uHX vXU vzj wCC wdJ wkt yIf yYW ySL zEX zQe zcl zgd
```

**Testfall 5:**

```
test case sort 6, negative Zahlen und doppelte Zahlen ("negativ.txt"):
File to sort : 5 -1 1 6 34 1424 -33 4 -7 -23 3 565 7 5 99 -99

Sorted File: -1 -23 -33 -7 -99 1 1424 3 34 4 5 5 565 6 7 99
```

