Homework 1 program report

Student: Dong Liang

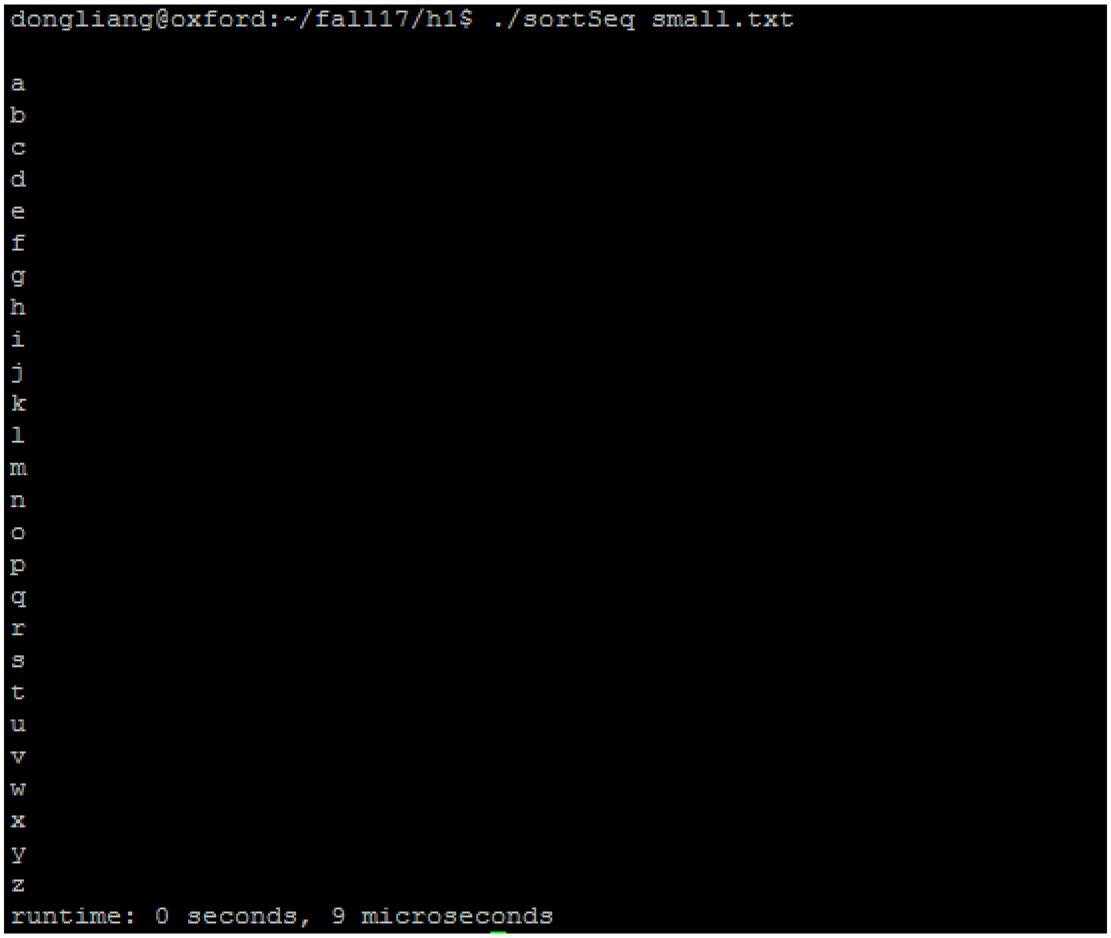
Net-id: dongliang

Test Environment: off-campus PC through on Lectura and ssh to oxford.

Test for quicksort

Description: The quicksort function has been practiced several times in 345 and 445. That means the code I implemented for this program is solid. But this is a C programming project and the array to be sorted is for Strings. So, I did test for quicksort function.

Test file used: A text file named small.txt which contains letters in lines.



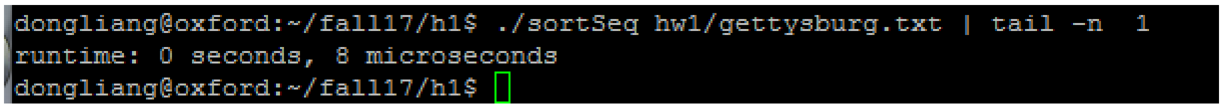
```
dongliang@oxford:~/fall17/h1$ ./sortSeq small.txt

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
runtime: 0 seconds, 9 microseconds
```

Test for sortSeq

Description: This one is supposed to be the lowest one among the three. But the expect performance may not appear until the size of the file gets large.

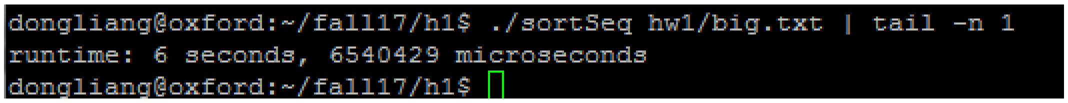
Test files used: gettysburg.txt big.txt bigger.txt

gettysburg.txt



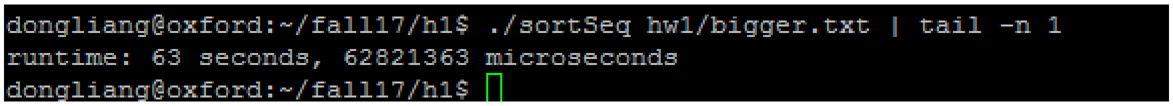
```
dongliang@oxford:~/fall17/h1$ ./sortSeq hw1/gettysburg.txt | tail -n  1
runtime: 0 seconds, 8 microseconds
dongliang@oxford:~/fall17/h1$
```

big.txt



```
dongliang@oxford:~/fall17/h1$ ./sortSeq hw1/big.txt | tail -n 1
runtime: 6 seconds, 6540429 microseconds
dongliang@oxford:~/fall17/h1$
```

bigger.txt



```
dongliang@oxford:~/fall17/h1$ ./sortSeq hw1/bigger.txt | tail -n 1
runtime: 63 seconds, 62821363 microseconds
dongliang@oxford:~/fall17/h1$
```

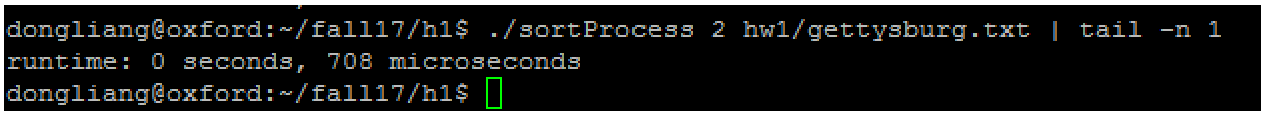
Test for sortProcess

Description: This one is supposed to be faster than sortSeq for big files.

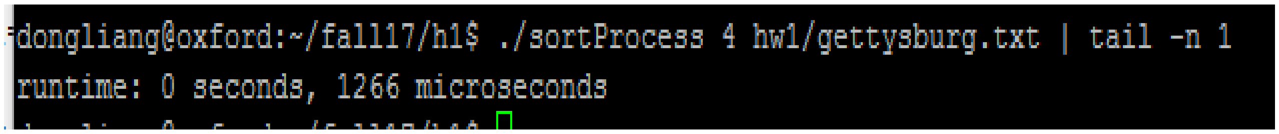
Using different numbers of processes working on the same file.

Test files used: gettysburg.txt big.txt bigger.txt

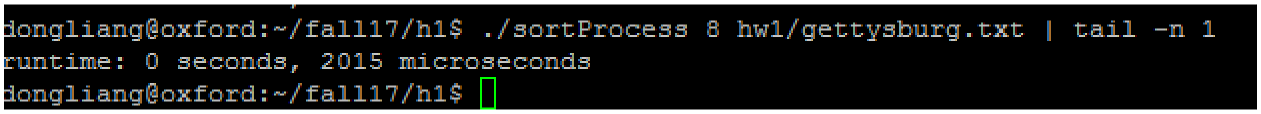
gettysburg.txt



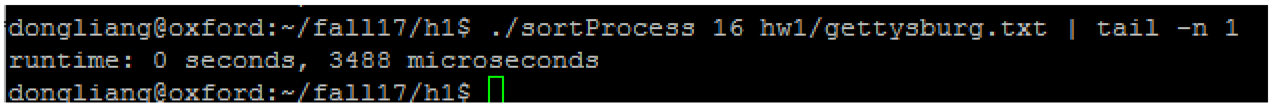
```
dongliang@oxford:~/fall17/h1$ ./sortProcess 2 hw1/gettysburg.txt | tail -n 1
runtime: 0 seconds, 708 microseconds
dongliang@oxford:~/fall17/h1$
```



```
dongliang@oxford:~/fall17/h1$ ./sortProcess 4 hw1/gettysburg.txt | tail -n 1
runtime: 0 seconds, 1266 microseconds
```

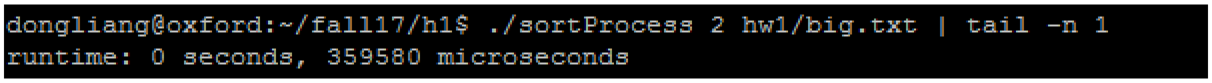


```
dongliang@oxford:~/fall17/h1$ ./sortProcess 8 hw1/gettysburg.txt | tail -n 1
runtime: 0 seconds, 2015 microseconds
dongliang@oxford:~/fall17/h1$ 
```

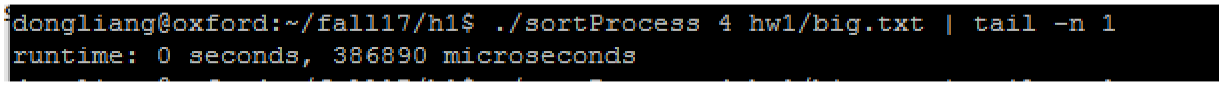


```
dongliang@oxford:~/fall17/h1$ ./sortProcess 16 hw1/gettysburg.txt | tail -n 1
runtime: 0 seconds, 3488 microseconds
dongliang@oxford:~/fall17/h1$ 
```

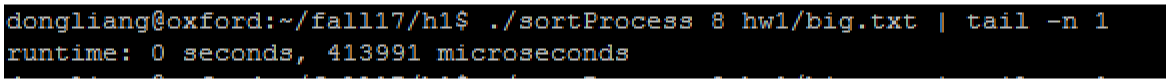
big.txt



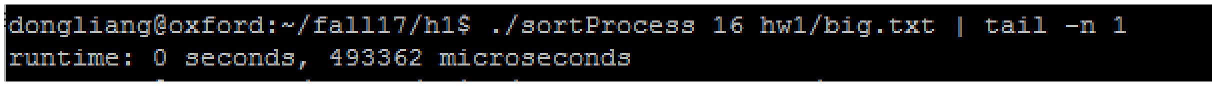
```
dongliang@oxford:~/fall17/h1$ ./sortProcess 2 hw1/big.txt | tail -n 1
runtime: 0 seconds, 359580 microseconds
```



```
dongliang@oxford:~/fall17/h1$ ./sortProcess 4 hw1/big.txt | tail -n 1
runtime: 0 seconds, 386890 microseconds
```

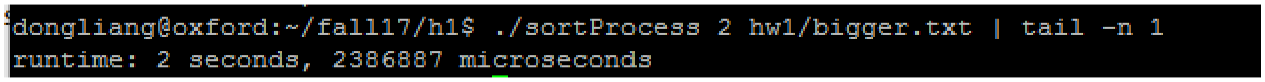


```
dongliang@oxford:~/fall17/h1$ ./sortProcess 8 hw1/big.txt | tail -n 1
runtime: 0 seconds, 413991 microseconds
```

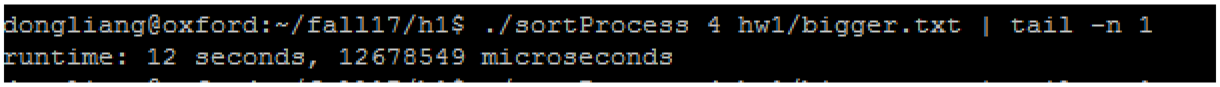


```
dongliang@oxford:~/fall17/h1$ ./sortProcess 16 hw1/big.txt | tail -n 1
runtime: 0 seconds, 493362 microseconds
```

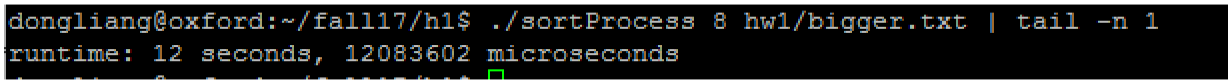
bigger.txt



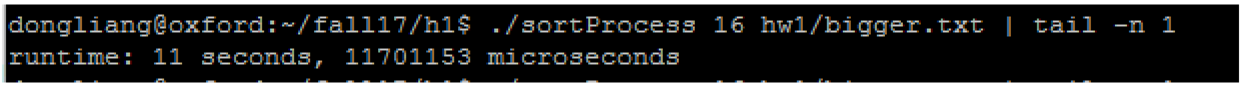
```
dongliang@oxford:~/fall17/h1$ ./sortProcess 2 hw1/bigger.txt | tail -n 1
runtime: 2 seconds, 2386887 microseconds
```



```
dongliang@oxford:~/fall17/h1$ ./sortProcess 4 hw1/bigger.txt | tail -n 1
runtime: 12 seconds, 12678549 microseconds
```



```
dongliang@oxford:~/fall17/h1$ ./sortProcess 8 hw1/bigger.txt | tail -n 1
runtime: 12 seconds, 12083602 microseconds
```



```
dongliang@oxford:~/fall17/h1$ ./sortProcess 16 hw1/bigger.txt | tail -n 1
runtime: 11 seconds, 11701153 microseconds
```

Analysis and conclusion

More processes using doesn't mean better performance for small size files. Had no time for testing super-size files (the down of lectura on Monday evening was hurt). Or, my algorithm of the process implements sucked…

Compare to sortSeq

The numbers say that for small size files, sortSeq is good enough. But the multi-process does help the performance of big size files.

Compare to sortThread

Will be covered in sortThread session.

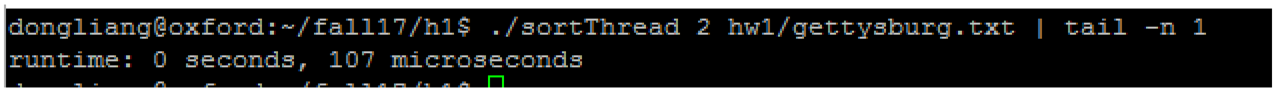
Test for sortThread

Description: This one is supposed to be faster than sortSeq for big size files and I will figure it out if it is also faster than sortProcess.

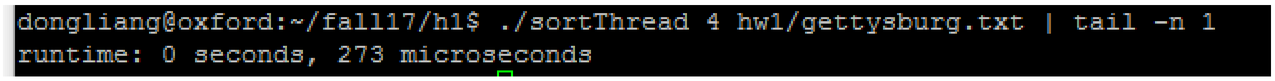
Using different numbers of threads to work on the same file.

Test files used: gettysburg.txt big.txt bigger.txt

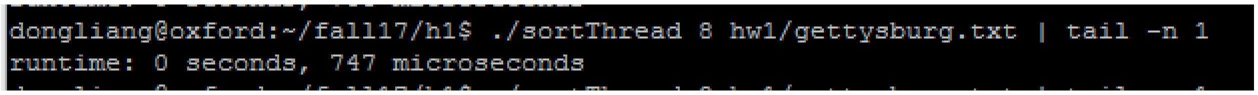
gettysburg.txt



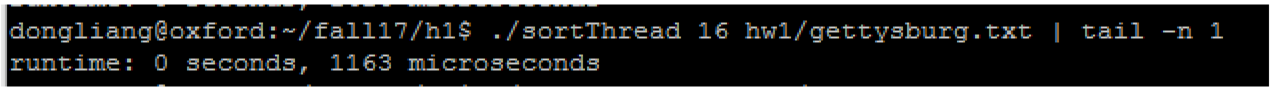
```
dongliang@oxford:~/fall17/h1$ ./sortThread 2 hw1/gettysburg.txt | tail -n 1
runtime: 0 seconds, 107 microseconds
```



```
dongliang@oxford:~/fall17/h1$ ./sortThread 4 hw1/gettysburg.txt | tail -n 1
runtime: 0 seconds, 273 microseconds
```

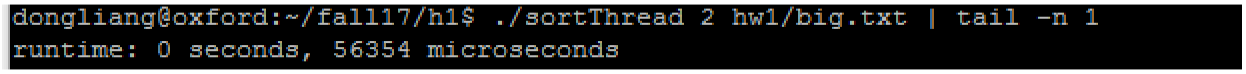


```
dongliang@oxford:~/fall17/h1$ ./sortThread 8 hw1/gettysburg.txt | tail -n 1
runtime: 0 seconds, 747 microseconds
```

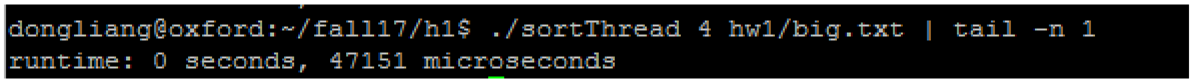


```
dongliang@oxford:~/fall17/h1$ ./sortThread 16 hw1/gettysburg.txt | tail -n 1
runtime: 0 seconds, 1163 microseconds
```

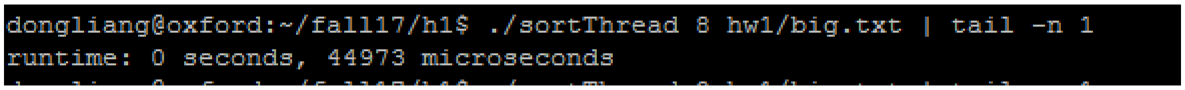
big.txt



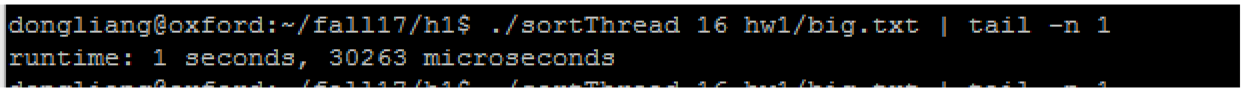
```
dongliang@oxford:~/fall17/h1$ ./sortThread 2 hw1/big.txt | tail -n 1
runtime: 0 seconds, 56354 microseconds
```



```
dongliang@oxford:~/fall17/h1$ ./sortThread 4 hw1/big.txt | tail -n 1
runtime: 0 seconds, 47151 microseconds
```

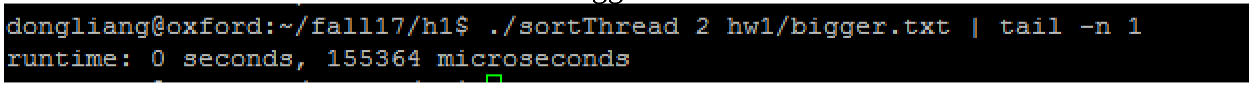


```
dongliang@oxford:~/fall17/h1$ ./sortThread 8 hw1/big.txt | tail -n 1
runtime: 0 seconds, 44973 microseconds
```

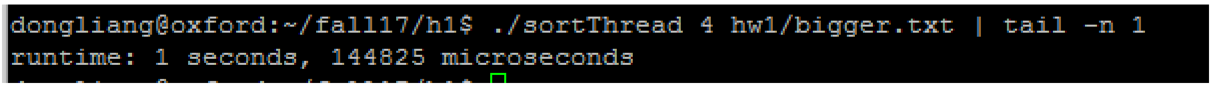


```
dongliang@oxford:~/fall17/h1$ ./sortThread 16 hw1/big.txt | tail -n 1
runtime: 1 seconds, 30263 microseconds
```

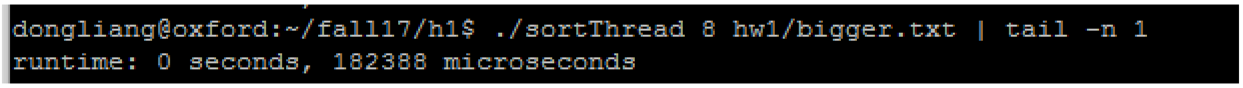
bigger.txt



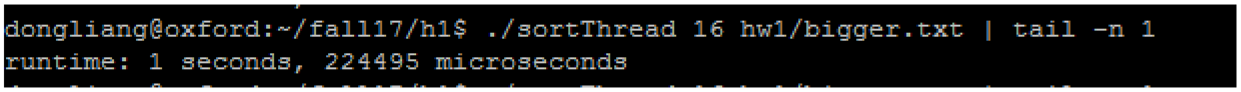
```
dongliang@oxford:~/fall17/h1$ ./sortThread 2 hw1/bigger.txt | tail -n 1
runtime: 0 seconds, 155364 microseconds
```



```
dongliang@oxford:~/fall17/h1$ ./sortThread 4 hw1/bigger.txt | tail -n 1
runtime: 1 seconds, 144825 microseconds
```



```
dongliang@oxford:~/fall17/h1$ ./sortThread 8 hw1/bigger.txt | tail -n 1
runtime: 0 seconds, 182388 microseconds
```



```
dongliang@oxford:~/fall17/h1$ ./sortThread 16 hw1/bigger.txt | tail -n 1
runtime: 1 seconds, 224495 microseconds
```

Analysis and conclusion

SortThread was expected to be the fastest one by me. And it does. Also, the more threads used, the better performance will be present.

Compare to sortSeq

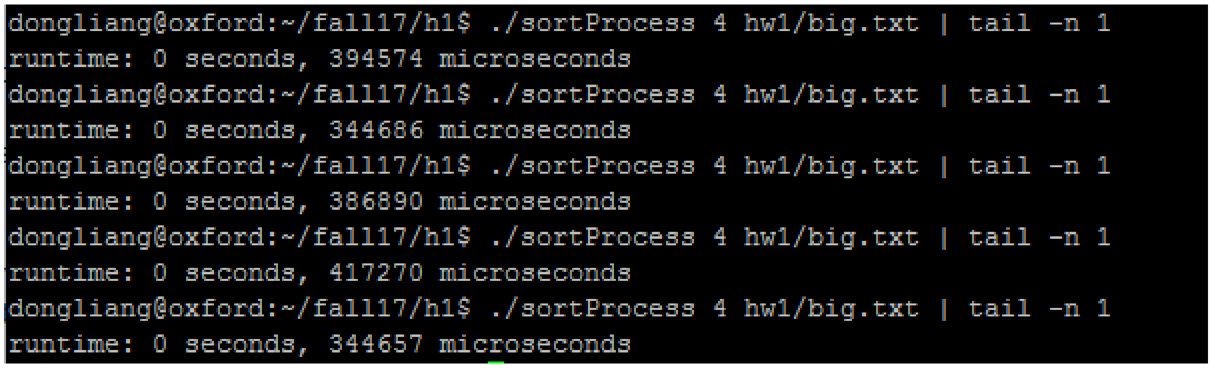
Still, using threads doesn’t help for small files. But sortThread is much faster than sortSeq when the test files become lager.

Compare to sortProcess

SortThread beats sortProcess :)

Note:

For each test, I run the same command for 5 times and pick the one that cost the median time to create the short cut. Like this:



```
dongliang@oxford:~/fall17/h1$ ./sortProcess 4 hw1/big.txt | tail -n 1
runtime: 0 seconds, 394574 microseconds
dongliang@oxford:~/fall17/h1$ ./sortProcess 4 hw1/big.txt | tail -n 1
runtime: 0 seconds, 344686 microseconds
dongliang@oxford:~/fall17/h1$ ./sortProcess 4 hw1/big.txt | tail -n 1
runtime: 0 seconds, 386890 microseconds
dongliang@oxford:~/fall17/h1$ ./sortProcess 4 hw1/big.txt | tail -n 1
runtime: 0 seconds, 417270 microseconds
dongliang@oxford:~/fall17/h1$ ./sortProcess 4 hw1/big.txt | tail -n 1
runtime: 0 seconds, 344657 microseconds
```

I picked the time 386890 microseconds as the result.