Lab H Design

MergeSort code from: http://www.geeksforgeeks.org/iterative-merge-sort/
Written by Shivam Agrawal (ref: http://csg.sph.umich.edu/abecasis/class/2006/615.09.pdf)
Supplement: https://en.wikipedia.org/wiki/Merge_sort

Recursive MergeSort():

```
/* Function to merge the two haves arr[l..m] and arr[m+1..r] of array arr[] */
void rMerge(int array[], int left, int middle, int right);

/* l is for left index and r is right index of the sub-array of arr to be sorted */
void rMergeSort(int array[], int left, int right);
```

Iterative MergeSort():

```
/* Function to merge the two haves arr[l..m] and arr[m+1..r] of array arr[] */
void iMerge(int array[], int left, int middle, int right);

// Utility function to find minimum of two integers
int min(int x, int y);


/* Iterative mergesort function to sort arr[0...n-1] */
void iMergeSort(int array[], int n);
```

Main Function:

```
Int *unsortedArray;
Int *recusriveArray;
Int *iterativeArray;

POPULATE A FILE WITH DIGITS

Open an ostream file
Prompt user for number of digits, save number as arraySize

For (int i = 0 ; I < arraySize; i++)
     outputFile << random number << " ";

close the file

OPEN FILE AS ifstream & EXTRACT THE DIGITS

Open file

//Create dynamic array of ints the size of arraySize
     unsortedArray = new int[arraySize];

//use the only effing working way of getting ints
while (inputFile >> unsortedArray[index]) {
     index++
}

copy the unsorted array to the arrays to be passed:
recursiveArray = new int[arraySize];
iterativeArray = new int[arraySize];
for (int I = 0; I < arraySize; i++) {
     recursiveArray[i] = unsortedArray[i];
     iterativeArray[i] = unsortedArray[i];
}

close the ifstream file

create and open a new ofstream file for results
```

<u>COMPLETE RECURSIVE MERGESORT</u>

```
Int startClock = clock(); //// make it a double

recursiveMergeSort(recursiveArray, 0, arraySize)

int stopClock = clock();

recursiveTime = (stopClock – startClock)
               /double(clocks_per_second)*1000
               //for milliseconds
cout the time

store clock time in a clock variable and cast it to float for
clock/sec op
```

<u>COMPLETE ITERATIVE MERGESORT</u>

```
Int startClock = clock();

iterativeMergeSort(iterativeArray, 0, arraySize)

int stopClock = clock();

iterativeTime = (stopClock – startClock)
               /double(clocks_per_second)*1000
               //for milliseconds
cout the time
```

<u>Print sorted arrays and times to new output file</u>

```
Print the times
For loop to print one array
newline
fopr loop to print the other array

close the output file and deallocate the memory
```

Testing

During extensive testing and debugging of the found source code for mergeSort I found 2 errors in the code. 1 of them is that the code undershoots or overshoots the array while passing the code around its member functions. So at some point junk integers will be passed and saved or integers may be repeated. Luckily, that's not the point of this lab.

The second problem is that the code may cause segmentation faults when dealing with very large arrays. After discussing this with Ian during his office hours, we decided that this was most likely caused by the array size itself. Numbers this large cannot be held by an int variable and this leads to an eventual segmentation fault. This makes sense since the seg faults seem to only occur on very large arrays (see bottom of attached FLIP screenshot).

I did a lot of my research on my mac, because the way I was initially counting time (using clock and then using algebra to calculate milliseconds) would print 0s on flip regardless of what I did. I finally implemented chrononos to account for that. During testing on my mac the time counts were erroneous. Whether I used clock() or chromos to keep track of the execution time of both functions, iterative was only faster initially. Once the arrays grew to very large numbers, beyond 10,000 elements, the recursive function became faster. Because these array sizes were under 1,000,000 elements the time difference was always fairly marginal but recursive mergeSort() did prove faster on the larger arrays. This is counter-intuitive since the increased function header should have led to an increase in processing time. Somehow I think my Macbook was accounting for that (see attached appendix with mac terminal output arrays are omitted due to extreme length).

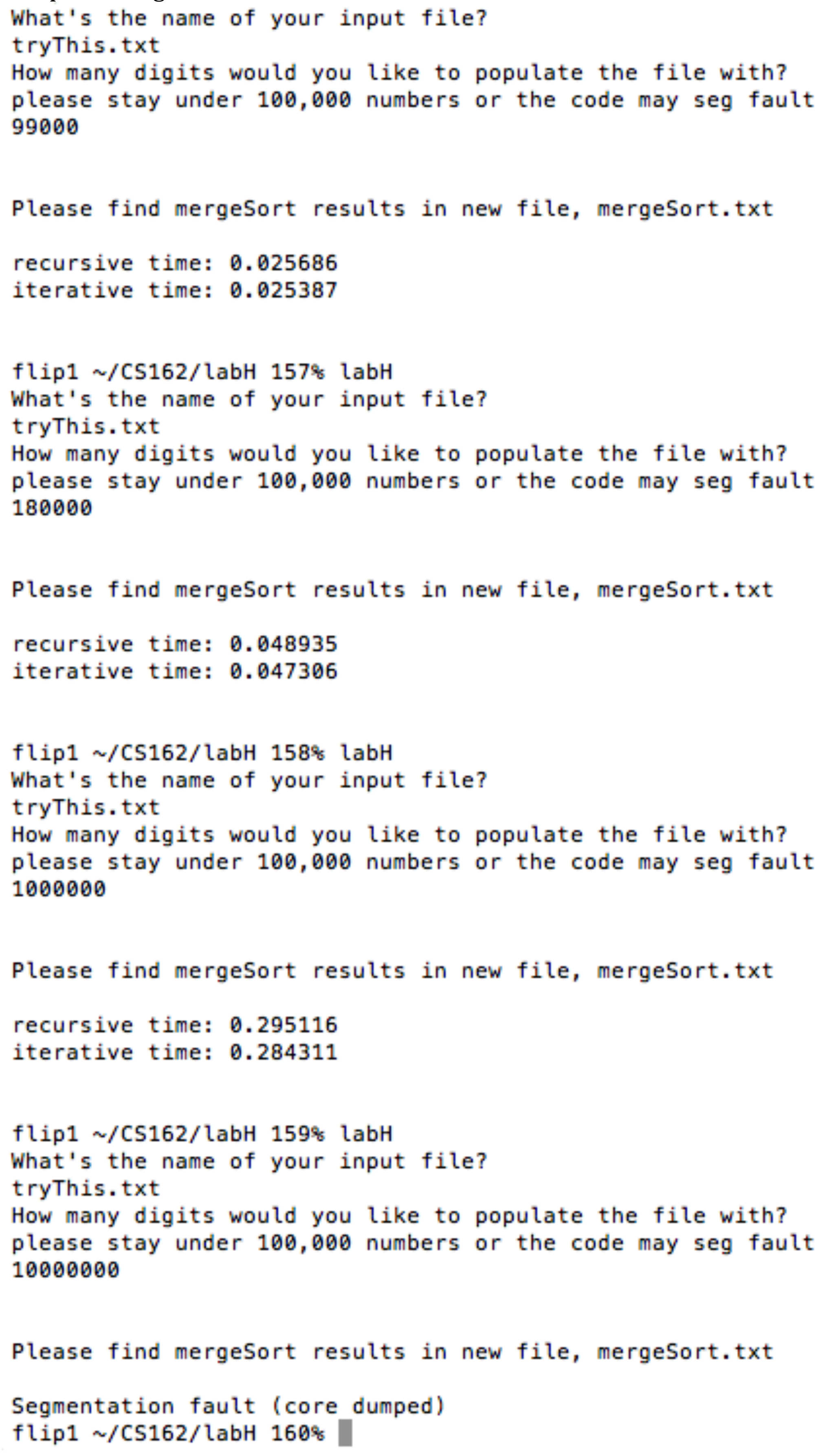
When I tested the program on flip, using chronos, the timing function finally worked as intended. When I tested the functions on flip the iterative mergeSort was consistently faster (even though it was not by a lot). It also quickly became apparent that the function had 2 limitations: the array size may not to be too small (10 or less is especially erroneous) or too large (over about 2,091,000) , either of these conditions tends to result in a segmentation fault.

Sample size (number digits)	Time (recursive)	Time per digit (recursive)	Time (iterative)	Time per digit (iterative)
10	0.000002s	0.0000002000s	0.000002s	0.0000002000s
100	0.000016s	0.0000001600s	0.000015s	0.0000001500s
1,000	0.000189s	0.0000001890s	0.000176s	0.0000001760s
10,000	0.002368s	0.0000002368s	0.002290s	0.0000002290s
100,000	0.026255s	0.0000002626s	0.025560s	0.0000002556s
1,000,000	0.296754s	0.0000002968s	0.284372s	0.0000002844s
2,000,000	0.612468s	0.0000003062s	0.589898s	0.0000002949s

*arrays larger than 2,000,000 or smaller than 100 elements are likely to cause a segmentation fault

As evidenced in the table above, as the size of the array grows to the function limit not only does the function take more time, but the function spends more time dealing with each number. However, when testing on flip, it is apparent that the recursive function is conclusively slower. Marginal as it may be. Thus the lab has successfully shown the effect function header can have when recursive functions are handling very large data files, whereas iterative functions can save time and memory space. This allows busy systems to run more functions at a time and run them faster (since theoretically there is more memory space available and the faster run-time means the program can move on faster.

Sample testing on FLIP:



```
What's the name of your input file?
tryThis.txt
How many digits would you like to populate the file with?
please stay under 100,000 numbers or the code may seg fault
99000


Please find mergeSort results in new file, mergeSort.txt

recursive time: 0.025686
iterative time: 0.025387


flip1 ~/CS162/labH 157% labH
What's the name of your input file?
tryThis.txt
How many digits would you like to populate the file with?
please stay under 100,000 numbers or the code may seg fault
180000


Please find mergeSort results in new file, mergeSort.txt

recursive time: 0.048935
iterative time: 0.047306


flip1 ~/CS162/labH 158% labH
What's the name of your input file?
tryThis.txt
How many digits would you like to populate the file with?
please stay under 100,000 numbers or the code may seg fault
1000000


Please find mergeSort results in new file, mergeSort.txt

recursive time: 0.295116
iterative time: 0.284311


flip1 ~/CS162/labH 159% labH
What's the name of your input file?
tryThis.txt
How many digits would you like to populate the file with?
please stay under 100,000 numbers or the code may seg fault
10000000


Please find mergeSort results in new file, mergeSort.txt

Segmentation fault (core dumped)
flip1 ~/CS162/labH 160% 
```

Testing on Mac (saved output from files):

```
Time to complete recursive MergeSort on 18360 digits: 3.397
Time to complete iterative MergeSort on 18360 digits: 3.626

Time to complete recursive MergeSort on 10 digits: 0.006
Time to complete iterative MergeSort on 10 digits: 0.004

Time to complete recursive MergeSort on 100 digits: 0.029
Time to complete iterative MergeSort on 100 digits: 0.024

Time to complete recursive MergeSort on 1,000 digits: 0.323
Time to complete iterative MergeSort on 1,000 digits: 0.279

Time to complete recursive MergeSort on 10,000 digits: 1.856
Time to complete iterative MergeSort on 10,000 digits: 1.889

 Time to complete recursive MergeSort on 100,000 digits: 21.202
 Segmentation fault: 11
```