**Write a brief report on what you found, what you did, and what other changes to the code you'd recommend**

**NOTE – This whole document is a report and contains suggestions and implementations by me to address various issues/bugs.**

First of all I tried to see if the files were being read correctly. I noticed that in each of the files, the last word wasn't being read. So I found out that listOut was being used only if endofFile is false but when endofFile is false, it wouldn't insert the last word. So I moved listOut.push\_back one line above and modified the if to include fileIn.close and a break statement.

Next, I experimented with DoSingleThreaded and noticed that if string a is a prefix of string b, then a is printed below b but in the correct files it is reversed. So in the return I changed it to i== \_first.length(). This gave the correct output for SingleAscending.txt

Moved to SingleDescending. I created a new commonSort function that takes 2 strings and the type of sort and return either true or false depending on the sorting and the string. This made it easier to code and reduced the if statments. I created a new class for SingleDescending called AlphabeticalDescendingStringComparer and swapped the return statements. Compared with the answer and found that prefixes were in the wrong order. I changed the final return to (i == \_second.length()) and it worked.

For the last one, I noticed a pattern that the strings were sorted in reverse order. So I reversed the strings and applied SingleAscending sort function and changed the last return to (i == \_second.length())

**Improve performance as much as possible on both single-threaded and multi-threaded versions; speed is more important than memory usage**

To improve performance I did a number of changes. One was to introduce merge sort since it has complexity O(nlogn). Another optimization to merge sort would be to multi thread it. Merge sort can be naturally multi threaded since it is a divide and conquer algorithm. Each divide can be sent to a separate thread. For the program, I divided the main array into 2 parts and assigned each part to 1 thread.

In addition, I avoid calls to the copy constructor. This is called whenever an array is returned. To avoid this, I passed arrays by reference and then modified them in place.

Vector push back is inherently slow since it doesn’t know how much memory to allocate. But what if we don’t know the vector size? I used vector reserve with an approximate value. This will only reserve memory for the vector and can avoid costly copies when the vector is full after push back. Emplace back is another improvement since it avoid a copy.

File I/O is notoriously slow. It is faster to read/write 1000 bytes at once rather than 1 byte a thousand times. Since the text files were small, I read them as a whole chunk and then passed them into memory. This used 1 file i/o call and another call for the size. Thus, only 2 calls were made for read. For write, I merged all data into 1 string and wrote the whole string at once. This gave me 1 call.

‘new’ allocates memory on the heap. Since it is a heap, programs can ask for as much memory as needed but at the cost that it is not O(1). When trying to call the stringsorter methods in the classes, using new and delete obviously does not result in a memory leak but the heap allocation is slower. Instead, we use stack memory since the stack is sequential. Stack memory is small so large dynamic variables should not be initialized. Since we only declare an integer, it fits perfectly and moreover, the memory is free automatically. In C++ this is called automatic storage.

Initially, I experimented with mutexes but found them to be very slow. This is because if there if there is contention between threads, many syscalls are made and these round-trip calls to the kernel can be 1000s of CPU cycle [1]. Instead, I allocated n vectors where n is the number of files. Although, this was very fast, I suspected another problem: False sharing.

False sharing occurs when several threads write to the same cache line and the CPU thinks that that they’re shared which causes performance degradation. One way to avoid this is to pad the variable so that it uses 1 full cache line. For the simultaneous writes, I created n padded vectors and the padding is 1024 bytes. It is possible for false sharing to occur in vectors since they might be allocated contiguous spaces but since the vectors are empty, the CPU will not understand if they are meant to be in the same cache line of not [2][3] [See fig 1]. Orange line is unpadded and blue is padded. We notice a slight difference improvement in performance when padded vectors are used.

Below charts are clocks times for 100 calls of the same function

**Figure 1**

**Figure 2**

**Figure 3**

**Figure 4**

Also added these 3 lines to unlink printf and scanf

ios\_base::sync\_with\_stdio(0);

cin.tie(NULL);

cout.tie(NULL);

For additional performance analysis, I used Visual Studio’s profiler which showed my the hot paths and helped in general performance improvements.

**Improve safety and stability; fix memory leaks and handle unexpected input and edge cases**

In order to fix memory leaks, I made of use of crtdbg which helps in analyzing memory.

I avoided the new keyword (Initially I used ‘delete’ which frees memory).

Oddly enough, I found that directory\_iterator seems to have memory issues. I used crtdbg and got the following output

0 bytes in 0 Free Blocks.

352 bytes in 11 Normal Blocks.

8492 bytes in 15 CRT Blocks.

0 bytes in 0 Ignore Blocks.

0 bytes in 0 Client Blocks.

Largest number used: 131377 bytes.

Total allocations: 3493618 bytes.

Detected memory leaks!

Dumping objects ->

{201} normal block at 0x0107AF50, 112 bytes long.

Data: < h > 98 B3 08 01 68 AA 07 01 00 00 00 00 00 00 00 00

{200} normal block at 0x0107A6A0, 48 bytes long.

Data: <../InputFiles\Ne> 2E 2E 2F 49 6E 70 75 74 46 69 6C 65 73 5C 4E 65

{199} normal block at 0x0108B4E8, 8 bytes long.

Data: < > A4 AF 07 01 00 00 00 00

{194} normal block at 0x0107A548, 48 bytes long.

Data: <../InputFiles\Ne> 2E 2E 2F 49 6E 70 75 74 46 69 6C 65 73 5C 4E 65

{193} normal block at 0x0108AE58, 8 bytes long.

Data: < > 88 AF 07 01 00 00 00 00

{188} normal block at 0x0107A818, 48 bytes long.

Data: <../InputFiles\Ne> 2E 2E 2F 49 6E 70 75 74 46 69 6C 65 73 5C 4E 65

{187} normal block at 0x0108ADB0, 8 bytes long.

Data: <l > 6C AF 07 01 00 00 00 00

{182} normal block at 0x0107AA68, 48 bytes long.

Data: <../InputFiles\Ne> 2E 2E 2F 49 6E 70 75 74 46 69 6C 65 73 5C 4E 65

{181} normal block at 0x0108B398, 8 bytes long.

Data: <P > 50 AF 07 01 00 00 00 00

{166} normal block at 0x0108B2F0, 8 bytes long.

Data: <H > 48 FC CF 00 00 00 00 00

{164} normal block at 0x0108B0C0, 8 bytes long.

Data: <l > 6C FC CF 00 00 00 00 00

Object dump complete.

I attempted to check

DoSingleThreaded(fileList, ESortType::AlphabeticalAscending, "SingleAscending");

DoSingleThreaded(fileList, ESortType::AlphabeticalDescending, "SingleDescending");

DoSingleThreaded(fileList, ESortType::LastLetterAscending, "SingleLastLetter");

DoMultiThreaded(fileList, ESortType::AlphabeticalAscending, "MultiAscending");

DoMultiThreaded(fileList, ESortType::AlphabeticalDescending, "MultiDescending");

DoMultiThreaded(fileList, ESortType::LastLetterAscending, "MultiLastLetter");

But found no memory leaks.

Windows uses \r\n as line delimiter. This can cause read problems. I used \_WIN32 and \_\_unix\_\_

To check if the OS is unix or windows (CRLF vs LF).

I added directory and file checks to see if input directory is a real directory and also if it contains any text files.

One important security changes can be made which checks the magic number of a file. The magic number is the signature of the file and is found by reading the first 4 bytes. The problem with text files is that they do not have a magic number. Moreover, If you add characters that match the ascii representation of the magic number of a png, the program will think the text file is a png. I have left this for future implementation but found this important to mention.

Exceptions in child threads are not handled in the main thread and thus can cause issues. Instead, exception handling should be done inside a thread and the exception should be sent to a queue that can be displayed to a user.

References:

1. <https://stackoverflow.com/a/66506828>
2. [False sharing - Wikipedia](https://en.wikipedia.org/wiki/False_sharing)
3. [c++ - Can vector cause false sharing - Stack Overflow](https://stackoverflow.com/questions/70661070/can-vector-cause-false-sharing)