Solution for 2024-2025 DSA Exam, second sitting, Code 1

Write the data structures and algorithms necessary to sort a file containing integer numbers separated by a space.

The program will have 3 or 4 command line arguments.

The first argument represents the operations to be performed and can be either D (Distribute) (4 points) or S (Sort) (4 points).

If the first argument is D then there are 3 other arguments representing as follows:

- **The second argument is the file containing the initial unsorted, space separated, values**
- **The third argument is an output file.**
- **The forth argument is another output file.**

In case the first argument is D, the program will distribute already existing sorted sequences, alternatively, from the input file to the two output files. A sorted sequence is considered a sequence of numbers for which the current value is not less than the previous one. When a sequence is interrupted then, the next sequence will be written in the other file, alternatively. (i.e. first sequence in the first file, second sequence in the second file, third sequence in the first file (after the first sequence already there), forth sequence in the second file and so on...).

Example: ./yourprogram D ../tests/test1input test1output1 test1output2
The program should write sorted sequences from ../tests/test1input alternatively into test1output1 and test1output2.

If the first argument is S (sort) then there are two other arguments:

- **The first command line argument will be the path of a file containing the original, unsorted integers. The values are separated by a space character.**
- **The second command line argument will be the name of a file which will be created in the current working directory by your code.This file will also be the file which will contain the sorted integer after your program has executed. After code execution, this file will contain the integers, separated by a space character in ascending order. After the last value there will be a space character, right before the end of file marker.**

Example: ./yourprogram S ../tests/test1input test1output

Your program should sort the values in ../tests/test1input into test1ouput

Constraints:

- **There is no known information regarding the size of the file.**
- **There should be no array declared in your code**
- **There should be no extra memory allocated statically or dynamically, except for a few working variables.**

Assume the arguments exist and are correct both in number and in value(s).

Assume the input file exists, is not empty, and it contains numbers separated by space.

Output files might or might not exist in the current working directory. If they exist they need to be overwritten. If they do not exist they will need to be created.

Code must compile. Any errors and the subsequent failure of the compilation process will result in your submitted solution to be marked with 0 (zero) points.

Code must run and produce correct output. If the code compiles but no correct output is generated your submitted solution will be marked with 1 (one) point.
Violating the constraints will result in massive points deductions which can be up to 9 points.

Up to 2 (two) points will be given based on the code quality and legibility, complexity of solution and constraints.

```c
#include<stdio.h>
#include<string.h>

int split(char* src, char* temp1, char* temp2){
        int prev = 0, current = 0;
        FILE *source = fopen(src, "r");
        FILE *f1 = fopen(temp1, "w");
        FILE *f2 = fopen(temp2, "w");
        FILE* dest = f1;
        char sorted = 1;
 -----------------------------------------------
        fscanf(source, "%d ", &current);
        prev = current;
        fprintf(dest, "%d ", current);

        while(!feof(source)){
                fscanf(source, "%d ", &current);
                if(prev > current){
                        dest = (dest == f1) ? f2 : f1;
                        sorted =0;
                }
                fprintf(dest, "%d ", current);
                prev = current;
        }

        fclose(source);
        fclose(f1);
        fclose(f2);
return sorted;
}
```

Function for the Distribute requirement

Keep track of previous value so you can compare if the current value is or is not part of an ordered sequence.

Whenever the ordered sequence is interrupted we use the ternary operator (? :) to toggle the current output file. You can replace it with an if statement if you consider it's more comfortable.

If there is more than one ordered sequence it means that the input file is not sorted. We will return 1 or 0 to signal if the input file is sorted or not

Function for the Sort requirement

We merge the two files in which we alternatively distributed ordered sequences.
It is a natural merge sort algorithm with three sequences (files).

```c
void merge (char *destfile, char *temp1, char *temp2){
        FILE *dest = fopen(destfile, "w");
        FILE *f1 = fopen(temp1, "r");
        FILE *f2 = fopen(temp2, "r");
        int f1value, f2value, f1prev, f2prev;

        fscanf(f1,"%d",  &f1value);
        f1prev = f1value;
        fscanf(f2,"%d",  &f2value);
        f2prev = f2value;

        while(!feof(f1) && !feof(f2)){
                if(f1prev > f1value && f2prev > f2value){
                        f1prev = f1value;
                        f2prev = f2value;
                }

                if( !feof(f1) && f1prev <= f1value && (f2prev > f2value || (f2prev <= f2value && f1value <= f2value))){
                        fprintf(dest, "%d ", f1value);
                        f1prev = f1value;
                        fscanf(f1, "%d", &f1value);
                }

                if(!feof(f2) && f2prev <= f2value && (f1prev > f1value || (f1prev <= f1value && f1value > f2value))){
                        fprintf(dest, "%d ", f2value);
                        f2prev = f2value;
                        fscanf(f2, "%d", &f2value);
                }
        }

        if(!feof(f1))  fprintf(dest, "%d ", f1value);
        if(!feof(f2))  fprintf(dest, "%d ", f2value);

        while(!feof(f1)){
                fscanf(f1, "%d", &f1value);
                if(feof(f1)) break;
                fprintf(dest, "%d ", f1value);
        }

        while(!feof(f2)){
                fscanf(f2, "%d", &f2value);
                if(feof(f2)) break;
                fprintf(dest, "%d ", f2value);
        }

        fclose(dest);
        fclose(f1);
        fclose(f2);
}
```

While none of the two files to be merged have not yet reached the end

If we are at a new sequence in both of the files then initialise previous values for both.

If the current value from the first file is still in sequence (not less than the previous value) and the second file is out of sequence or the second file is in sequence but the first file value is not greater than the second one, write the first file current value and read next

Same but for the second file current value

If we have not reached end of file, we have the last value not written yet.

Write the rest of the file if the other file has reached the end

```c
int main(int argc, char **argv){
        char *action = argv[1];

        if(!strcmp(action, "D")){
                char *inputFile = argv[2];
                char *out1File = argv[3];
                char *out2File = argv[4];

                split(inputFile, out1File, out2File);
        return 0;
        }

        if(!strcmp(action, "S")){
                char *inputFile = argv[2];
                char *outputFile = argv[3];

                split(inputFile, "temp1", "temp2");

                do{
                        merge (outputFile, "temp1", "temp2");

                }while(!split(outputFile, "temp1", "temp2"));

                remove("temp1");
                remove("temp2");
}
return 0;
}
```

First argument is D - Distribute (split)

First argument is S - Sort

Notice the use of the split function to determine when the sorting algorithm ends.