

BITS Pilani, Pilani Campus
2nd Sem. 2019-20
CS F211 Data Structures & Algorithms

=====

Lab VI

=====

Topics: Recursion vs. Iteration - Running Time and Space Usage, File I/O and Sorting Large data.

Programming Environment: C on Linux

Exercise 1: [Expected Time: 80 minutes.]

- a) Implement an iterative version of the *merge* operation that takes two sorted arrays *Ls1* and *Ls2* and returns a single sorted array *Ls* such that all elements of *Ls1* and *Ls2* are included in *Ls* and the length of *Ls* is the sum of lengths of *Ls1* and *Ls2*. Assume that space for *Ls* is allocated outside and passed to *merge* i.e. the type signature for *merge* would be of the form:

void merge(Element Ls1[], int sz1, Element Ls2[], int sz2, Element Ls[])

[Note: Please note that each element in the array is a student record of the form *<Name,CGPA>*, which you will have to read from the given sample file - "1024.txt". Name is a string containing 10 characters and CGPA is a float value. Merging has to be performed on CGPA as the key.

End of Note]

- b) Implement a recursive version of Merge Sort algorithm that uses the *merge* procedure defined above to sort an array. Note that you have to copy the elements of the resultant sorted array into the original array every time *merge* is called.
- c) Implement an iterative version of Merge Sort algorithm that uses the *merge* procedure defined in (a) to sort an array. Note that you have to copy the elements of the resultant sorted array into the original array every time *merge* is called.

Exercise 2: [Expected Time: 30 minutes]

- a) Write a Makefile with targets *compRecMS*, *compIterMS*, *runRecMS*, *runIterMS*, and *compare* for compiling code for recursive MergeSort, compiling code for iterative MergeSort, running Recursive MergeSort, running Iterative MergeSort and comparing the two procedures respectively.
- b) Measure the time taken and space used by the two procedures in b) and c) for multiple inputs of different sizes. [Note: At this stage, the only difference in space usage between the versions of Merge Sort is the space used by the recursive calls.
- End of Note.]

Exercise 3: [Expected Time: 70 minutes]

- a) If you have a large file of records to be sorted you cannot bring all the records in memory for sorting. So write a procedure which does the following:
- repeat {

- ```

 read K records from the input file into an array
 sort the array using MergeSort
 store the result in a new temporary file
} until (all elements in the input file are sorted);
/* K is the size of the array that can be allocated in memory */

```
- b) Write a mergeFiles procedure that performs a merge operation where the input lists are in files and the output has to be a new file:
- ```

open file F1;
open file F2;
open file Out;
read record r1 from F1;
read record r2 from F2;
repeat {
    if (r1.key <= r2.key) {
        write r1 to Out; read r1 from F1;
    } else {
        write r2 to Out; read r2 from F2;
    }
} until (F1 is empty OR F2 is empty);
while (F1 is not empty) { read r1 from F1; write r1 to Out; }
while (F2 is not empty) { read r2 from F2; write r2 to Out; }
close files F1, F2, and Out.

```
- c) Write a mergeSort procedure which sorts a large file of size M records by using your solution in (a) to generate M/K files and then repeatedly merges two files at a time using your solution in (b) to obtain a single sorted file.
- You can use the sample file provided - "10240.txt", which consists of 10,240 student records containing <Name, CGPA>. You can choose various value of K such as "64", "256", "512" & "1024" and then compare their running time.

Assignment for Lab 6: Solve Exercise 2, put all the required files in a folder, name it with your ID number, and upload it.