BITS Pilani, Pilani Campus
2nd Sem. 2020-21
CS F211 Data Structures & Algorithms
==========
Lab V
==========

Topics: Recursion, Measurements: Running Time and Space Usage, Insertion Sorting, File I/O, and Build Tool - Makefile

Exercise 1: **[Expected Time: 40 minutes.]**
Write a procedure to read a large file of records and store it in an array:

a) Assume that each record contains information about a credit card: <Card_Number>, <Bank_Code>, <Expiry_Date>, <First_Name>, <Last_Name> where <Card_Number> is a 16-digit integer, <Bank_Code> is made of 5 characters, <Expiry_Date> is in the form mm/yyyy.
b) Assume that each record is stored in one line of the file.
c) Assume an initial size for the array and allocate dynamically. Resize the array when it is full. [Hint: Use *realloc.* End of Hint.]
d) Measure the time taken for the entire read procedure for different file sizes – start from 10,000 records and go up to 10,000,000 records. For initial testing, use given test files (10, 100, 1000, 10000) and then concatenate them repeatedly to generate requisite number of records.
e) Write the measurements into an output file.

Exercise 2: [Expected Time: 60 minutes]

a) Write an insertInOrder procedure to insert a credit card record (see previous exercise) into an array of credit cards. Credit card number is the key.
b) Write a recursive procedure implementing Insertion Sort (that calls the insertInOrder procedure).
c) Measure the time taken by the insertionSort procedure from (b) for varying list sizes. Plot a curve using these measurements. Compare this curve against an estimated curve [i.e. O(n*n) curve for insertion sorting.]
d) Measure the stack space used by the insertionSort procedure for different list sizes. [i.e. Use the address of a variable in *main* as the bottom of the stack. Use the address of a variable in the current top-of-stack frame and find the difference in addresses. This gives a very good approximation of stack size.]
e) Write the measurements from (c) and (d) to an output file.

Assignment wrt Lab5: Solve Exercise 2 and upload the solution with all the graphs. Do not forget to rename the folder with your BITS ID.

Exercise 3: [Expected Time: 30 minutes]
Although you must have already grasped knowledge about make files during your assignment submission, following is a short introduction to it.

Write a Makefile with targets run, and compile for the program in Exercise 2:

- **compile:** Compile files readRecords.c, insertionSort.c, and measureTimeAndSpace.c (with the appropriate header files) separately. Compile the object files (i.e. .o files) with the main .c file to produce an executable.
- **run:** Run the executable produced by **compile** once with the input file.

Introduction to Makefiles:
Makefile is a linux utility to easily compile and link large programs, typically containing multiple source and header files. Whenever "make" command is run, it searches for a file named "makefile" in the current directory and executes it.

A makefile is a collection of rules, where each rule consists of following:

- *target*: keyword that user can pass as argument to "make".
- *dependencies (prerequisites)*: all the files on which this target depends upon. If they are target themselves and have been modified since last execution then their corresponding rule is first executed
- *commands (recipes)*: all the commands which should be executed if all dependencies of current target are met (i.e. unchanged from last execution)

Syntax: for writing a rule (there should be a tab (not spaces) before each command):

```
target : dependency1   dependency2    dependencyn
     command1
     command2
     commandm
```

Example:
Assume that there are two modules (with a source and header file each) and a driver file. In order to create a makefile with a single target "compileAll", so that it can be executed with "`make compileAll`" command on terminal, a makefile can be created as follows:

```
**********************************************************
compileAll : module1.o module2.o driver.o
     gcc module1.o module2.o driver.o  -o myprogram

module1.o : module1.c module1.h
     gcc  -c  module1.c

module2.o : module2.c module2.h
     gcc  -c  module2.c

driver.o : driver.c module1.h  module2.h
     gcc  -c  driver.c
**********************************************************
```

Exercise 4: [Expected Time: 45 minutes]

a) Modify the Makefile in Exercise 3 to add a target *test* with a parameter that specifies a test distribution. (see b)
b) Generate the following different input distributions and repeat Exercise 2 for each of them:

    - Input list where several duplicate entries
    - Input list where a single entry is replicated a large number of times
    - Input list that is already in sorted order
- Input list that is sorted in reverse order