BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
First Semester 2010-2011
CS / IS C363 Data Structures and Algorithms
[Section 4 – Thursday, 19th August]
Lab Sheet – 2 [Duration: 165 minutes]

Problem

Brief:

- Implement a insetion sort, mergesort and quicksort algorithms and compare their performance.

Objective:
In this lab, comparatively analyze the performance of various sorting algorithms discussed in lecture.

Partition Procedure: This procedure must choose a pivot, p, uniformly randomly and use it for partitioning. (The Partition procedure can be referred from text book, or lecture slides.)

Description:

The program must take as input a file containing integers. The number of elements in the list of initegers is determined by the number of elements stored in the file. The program then invokes the methods for sorting elements in the list using the same set of input parameters in non-increasing order. It determines the time taken by each of the method for the same value of k. It then stores the time values along with the value of k used onto a file for plotting graph. Note that the size of the list of integers must be sufficiently large enough for getting any meaningful time values for comparison.

Step 1:
Create new type definitions: (**`List.h`**)

(a) Create a type definition for the type **`Element`**, which is represented as a structure containing an integer.
(b) Create a type definition for **`List`** of **`Element`**s, which is a dynamic array.
(c) Create a type definition for the type **`Location`**, which is used to maintain the location of an element.

Step 2:
Create header file for operations: (**`ListOps.h`**)

(d) **`void print(List list, int list_ size)`**: This function will find the kth smallest element using method 1 described above and returns it
(e) **`Location doPartition(List list, Location start, Location end):`** This function will uniformly randomly select a pivot from within the **`List`** list[**`start`**...**`end`**], then partition it into two sub lists as described above. This function returns the location in which pivot element is finally stored.
(f) **`Element Sort(List list, int list_ size):`** This function will sort the data in the list in non-increasing order.

Step 3:
Create two implementation files; method1.c, method2.c and method3.c.

In method1.c, implement the function described in (f) above using Insertion Sort algorithm.
In method2.c, implement the function described in (f) above using Merge Sort algorithm.
In method3.c, implement the function described in (e) and (f) above.
Implement the function (d) in a file named print.c

If you wish to implement recursive versions, then you must create separate helper functions in which the recursive versions are implemented; but you must not change the names/prototypes of the functions described above.

Step 4: driver file.
Create a driver file (**`evaluate.c`**) for meeting the objectives of this problem. The problem description above outlines the actions to be performed by the driver file. The names of the input file, the size(number of lines) of the input file and output file must be read as command-line arguments. The first parameter will be the input file and the second parameter is the output file. If second parameter is not specified, then the output needs to be printed in standard output. The input file must have only one integer in each line. The output file must be a comma separated file(.csv) containing the fields value of n, time needed by method 1, time needed by method 2 and time needed by method 3. The driver performs the following steps:

1. The program must take as input, a file containing integers stored in it; with one integer in each line. (Input file must be created by you.)
2. The program must read the integers from this input file, and store the integers in a dynamically created array.
3. for the values of N, ranging from 1000 to 100000, repeat the steps 4 through 9
4. note current time.(t_0)
5. Call the method1 for sorting the list.
6. note the current time (t_1). The time for computing step 5 is t_1 - t_0.
7. Call the method2 for sorting the list.
8. note the current time (t_2). The time for computing step 5 is $t_2 – t_1$.
9. Call the method3 for sorting the list.
10. note the current time (t_3). The time for computing step 7 is $t_3 – t_2$.
11. Store the time taken for the three methods and the value of N used in an output file.
12. Plot a graph with three curves one for each method with time taken on the y-axis and N on the x-axis.

You may use rand() function to generate random values, and store it in file. Please use a separate c file for this code. (You can also use the script provided) Refer man rand for details. Appendix 1 describes one of the methods for computing time consumed by a function.

Step 5: Performance evaluation.
Use the output file generated to plot a graph for comparatively evaluating the three methods. You may use xgraph, openoffice spread sheet, MS Office Excel, google charts, or any other method. Please name the file containing the plot as **`plot`**, with appropriate extension.

Support files: List.h, ListOps.h, input file(say, input.txt)

Deliverables: method1.c, method2.c, method3.c, print.c, evaluate.c, make file, output file (say, output.txt), plot file.

Appendix 1:

Measuring Execution Time:

```
   #include <sys/time.h>

   int main(int argc,char **argv)
   {           ......
// Declaration of timeval variables which will capture the system time
   long time_start, time_end;
   struct timeval tv;

//  Write all the initialization code here

 //  Time measurement begins here
   gettimeofday(&tv,NULL); /*gets the current system time into variable tv */
   time_start  = (tv.tv_sec *1e+6) + tv.tv_usec;
// calculating start time in seconds

//  Write the code to be analyzed here

   gettimeofday (&tv, NULL);
   time_end = (tv.tv_sec *1e+6) + tv.tv_usec;
 // calculating end time in seconds
// (time_end-time_start) gives the total execution time
   return (0);
}
```

Exercise:

Try the same problem, but using linked list instead of dynamic arrays.