27 December 2012

Analysis of Algorithm Project

All project members have to be present at the time of the demo.

You need to submit your report and Java programs, including a readme.txt file that explains how to run your program.

Part I

You are not allowed to use language libraries to sort the array!

Consider the following algorithms:
Group 1: Selection Sort, Bubble sort, Insertion Sort
Group 2: Merge Sort, Quick Sort

Choose one algorithm from Group 1 and one from Group 2. Implement the algorithms (Choose one from C, CSharp or Java). Your algorithm should take the array size as an input. You need to generate the array elements randomly. Your program should take a switch for choices i and ii.

i) Run the programs, with a properly inserted counter for the number of key comparisons, on 20 random arrays of sizes 1000, 2000, 3000, ..., 20000.

Size n	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12000	13000	14000	15000	16000	17000	18000	19000	20000
Average Key Comparison (Algorithm 1)																				
Avergae run time (msec)																				
Average Key Comparison (Algorithm 2)																				
Avergae run time (msec)																				

Plot the graph for the data above: one for the average key comparison and one for average run time. Analyze the data obtained to form a hypothesis about the algorithms' average-case efficiencies: determine the f(n) for the average key comparison and the average run time using interpolation techniques. Comment on the test results. What do you expect if input size is 25000 for each algorithm? Find the average cases of the algorithms for input size 25000. How much are they different from the results you find using f(n)?

ii) (To test your programs' correctness.) It should write the input array, the output arrays produced by each algorithm and the result into a file named output.txt. Arrays should be printed into files in columns. The first column should be input array, the second and the third columns should be the results of the algorithms. Seperate the columns using tab character (\t).

Let A be the input array, B and C be the results of the algorithms you have chosen. So the file format should be as follows:

Algorithm:	Algorithm1	Algorithm2
Array Size:	(result)	(result)
Running Time (msec) :	(result)	(result)
No. of Key Comparsions:	(result)	(result)

Input	Algorithm1	Algorithm2
A[0]	B[0]	C[0]
A[1]	B[1]	C[1]

....
....

Part II

Implement the Horspool's algorithm, the Boyer-Moore algorithm, and the brute-force string matching algorithm and run an experiment to compare their efficiencies for matching: Determine the number of key comparisons made by each algorithm. Your algorithm should take an input file: first line should contain the text and the second line should contain the pattern. For example:
BESS _ KN E W_AB O UT_BAOBABS
BAOB A B

Your output file should contain the results for each algorithm.

Brute-Force String Matching
Number of key comparison:
Run time (msec):

Horspool's algorithm

Shift Table:
Character c
Shift t(c)

Number of key comparison:
Run time (msec):

Boyer-Moore algorithm

Bad-symbol Shift Table:
Character c
Shift t(c)

Good-suffix Shift Table:

k	pattern	d2

Number of key comparison:
Run time (msec):

Part III

Implement the matrix chain problem using the following two approaches:

Approach 1: Multiply matrices in the order they are given.
Approach 2: Determine the order of matrix multiplication using dynamic programming.

You should randomly generate matrices of the given number.

Write the input and output (result) matrices into files. You shold also have a separate outfile file in the following format:

Matrix sizes: A1 n1 x m1 ; A2 n2 x m2; ......

In-order multiplication
Number of multiplication:
Run time (msec):

Multiplication with Dynamic Programming

Table for dynamic programming

Matrices with paranthesis: ((A1(A2A3)....)(....))
Number of multiplication:
Run time (msec):