CSE 2012- Design and Analysis of Algorithms
In-lab Practice Sheet 2
(Divide-Conquer-Combine)
Merge-sort Algorithm

Practice makes you Perfect

1. (a) Translate the merge-sort algorithm (A) discussed in the class into a program (P) and execute the same for the following inputs: (i):(0,1,2,3,4,5) (ii)(5, 5.5, 6, 3.723, 1.23, 8.88).

   (b) Run the code with different inputs and record the output of your exceution in the form of a table as follows. Here n is the size of the problem . Since $T(n) \in \theta(n.lgn)$, Take $T(n)$ as $n * lg\ n)$. $t(n)$ is the time taken by your program for executing the input of size n, as computed through your code. particular input Choose 50 different values for the size of the problem , n in an increasing magnitude and run your code. Please note the difference between $T(n)$ and $t(n)$.

Size of the problem (n)	T(n)	t(n) in seconds

   - Plot a graph $n\ \ Vs\ \ T(n)$.
   - Plot a graph $n\ \ Vs\ \ t(n)$.
   - Plot a graph $T(n)\ \ \ Vs\ \ \ t(n)$. Based on the above graphs, record your inferences related to the behaviour of $T(n)$ as n increases. Based on your experiment, express $T(n)$ in terms of O-notation and the Ω notation. Similarly, express $t(n)$ in terms of O-notation and the Ω notation.

   (c) We know that, for the merge-sort algorithm, $T(n)\ \ \in\ \ \theta(n * lg\ n)$ Based on your experiments, compute the values for c_1, c_2 and n_o to justify '$T(n)\ \in\ \theta(nlg\ n)$'

2. Given n 2-dimensional points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, Design an algorithm that follows the 'Divide-Conquer-Combine' strategy to

   (a) arrange the $n - points$ in an increasing order of the x coordinates

   (b) arrange the $n - points$ in an increasing order of the y coordinates

   (c) arrange the $n-points$ in decreasing order of the value $(x-coordinate+ y - coordinate)/2)$ coordinates

For each of the above algorithms, execute the algorithm with an appropriate code. Compute $t(n)$ for each of the algorithm.

3. Modify the Merge-sort algorithm (discussed in the class) such that the algorithm takes the input as words (of different lengths) and arrange them in an alphabetical order. Your words will have both lower-case letters as well as the upper-case letters. Compute the time-complexity $t(n)$ of your algorithm in an experimental approach. Compare this algorithm with that of the algrotihm which takes n numbers as inputs and decide which consumes minimium time.

4. Write a code to compute the number of comparisions $C(n)$ executed by the merge-sort algorithm (input: n numbers) for the input of size n. Similar to the time-complexity based on $T(n)$, compute the 'comparison-complexity based on $C(n)$, in an experimental approach. Based on this, compute the time-taken by your machine for one comparision.

5. Your Merge-sort algorithm A uses 'sentinels' for the purpose of 'array-end-marker'. Modify the Merge-sort algorithm and design a new algorithm A' so that A' is without any sentinels but the functions of A and A' are similar. Compute the $t(n)$ for both A and A'.

6. In the 'merge-sort' algorithm, an array of size n, is divided till we get n arrays of size 1. Modify the merge-sort algorithm in such a way that division is not carried out on any subarray of length ¡ 4. That is, the minimum length of all the subarrays should be 3 and for the subarray of length 3, apply any constant-time sorting algorithm. Write the code for the modified algorithm and compare the efficiency of the modified algorithm with that of the original algorithm.

7. In the merge-sort alogrithm, an array is divided into two subarrays whenever the 'divison' is applied. Modify the merge-sort algorithm in such a way that every divison leads to three subarrays. Write the code for the modified algorithm and compare the efficiency of the modified algorithm with that of the original algorithm.

8. Modify the merge-sort algorithm in such a way that the left subarray (got as a result of the first divison) is divided further till we get a subarray of size 1 and the right-subarray (got as a result of the first division) is not subjected to any division. For sorting the right subarray, apply insertion-sort algorithm. Write the modified algorithm A' to sort the n-given numbers

9. Merge-sort algorithm (discussed in the class) works by partioning the input array A recursively into two halves. Here, the partition is based on the position in the array. Instead, design a new algorithm A' where partioning is based on the values in the input array. Compare the performance of A' with that of A.