Introduction to algorithms (CS 207) (January-April, 2016): Assignment 2 Due:14^{th} February, 2016

1. Sort the following set of functions in increasing order of their asymptotic growth rates. State clearly the **partial** order, as some subsets of these functions may have identical asymptotic growth rates.

$\log(\log^* n)$	$2^{\log^* n}$	$(\sqrt{2})^{\log n}$	n^2	$n!$	$(\log n)!$
$\left(\frac{3}{2}\right)^n$	n^3	$\log^2 n$	$\log(n!)$	2^{2^n}	$n^{\frac{1}{\log n}}$
$\ln \ln n$	$\log^* n$	$n2^n$	$n^{\log \log n}$	$\ln n$	1
$2^{\log n}$	$(\log n)^{\log n}$	e^n	$4^{\log n}$	$(n+1)!$	$\sqrt{\log n}$
$\log^* \log n$	$2\sqrt{2 \log n}$	n	2^n	$n \log n$	$2^{2^{n+1}}$

2. Solve the following recurrence equations:
   (a) $T(n) = 3T(\frac{3n}{4}) + \frac{1}{n}$
   (b) $T(n) = T(\sqrt{n}) + \log n$
   (c) $T(n) = T(n-1) + 19$
   (d) $T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + \sqrt{n}$

3. (a) Implement the standard equal-split version of merge sort.

   (b) Implement the scan, divide-into-maximal-monotonic-increasing-subarrays, merge variant of merge sort.

4. (a) Name any optimal (in the worst-case) sorting algorithm and state its asymptotic running time.

   (b) Now consider a set of n numbers stored as a sequence in an array A, which needs to be sorted. One possibility is to sort the first f fraction of the sequence, and then sort the last f fraction of the sequence and then to sort the first f fraction again, in each case using the algorithm you stated above as a subroutine to sort the selected fractional subarray. What is the smallest value of f for which this procedure will correctly sort any input sequence? What is the running time of this modified sorting procedure in terms of f and the running time expression you wrote earlier (for the subroutine)? Solve it and express your answer in closed form asymptotic notation.

   (c) Consider an extended version of the above procedure, where you repeat the alternating procedures of sorting the first and last f fraction of the array until the sequence is sorted. What is the smallest value of f, for which this procedure will eventually sort the sequence. Compute the number of iterations required by your algorithm as a function of f.

   (d) Is the running time of these sorting algorithms smaller or larger, asymptotically, than the standard algorithms? If it is larger, then suggest some reason why this algorithm might be useful.

5. The rank of an element in a list of distinct elements is its position in the array in increasing sorted order. Finding the rank of an element takes $\Theta(n)$ time in an unsorted array and takes $\Theta(1)$ time in a sorted array.

   Consider an array A indexed from 1 to n, such that for some unknown index k, $1 \leq k \leq n$, the minimum element is at position k. The elements are placed in increasing order rightwards, starting with the minimum at position k and wrapping around at position n and continuing from position 1 to position $k-1$. Devise a $\Theta(\log n)$ worst case algorithm to find the rank of an element in this scenario. Implement code for this.