Due: Sep 28, 2017, in class. Please send your photocopy of homework papers to TA’s email address liumanni@msu.edu before the deadline.

1. (20 points) Given an array \( A = [9, 3, 7, 12, 0, 2] \), use insertion sort algorithm to sort \( A \) in an increasing order. The algorithm scans \( A \) from left to right by size(\( A \)) = 6 steps, so please list the changes of \( A \) during these 6 steps.

2. (25 points) Array \( A = [a_1, a_2, \cdots, a_n] \) is unsorted, design an \( O(n \log n) \)-time algorithm to report the number of inversions in \( A \). An inversion is a pair of numbers \( a_i \) and \( a_j \) such that \( i < j \) but \( a_i \geq a_j \) (hint: you may use divide-and-conquer idea).

3. (25 points) Given an array \( A = [7, 12, 2, 3, 10, 4, 20] \), build the MaxHeap for \( A \). Please list the updated \( A \) after calling MaxHeapify each time.

4. (15+15=30 points)
   \( 1 \) Suppose an internet company wants to sort a very large set of numbers, and the company has multiple powerful computers to work in parallel. Which sorting algorithm should we choose? Please explain your reason.

   \( 2 \) We know that insertion sorting costs \( \Theta(n^2) \) time for \( n \) numbers, because we just simply assume that each “shift” and “comparison” costs constant time. Here “shift” means shift a number from one location to another in the memory, and “comparison” means compare two numbers. But if we specify the running time for each “shift” and “comparison” to be \( \alpha \) and \( \beta \) respectively (which are not constants), what about the total time complexity of insertion sort?

5. (optional) Simulate the three sorting algorithms we have learned in class, and compare their real running time and space cost in practice. You may randomly generate a number of arrays to sort, and take the average running time and space cost.