Due Date: 17 September 2022
20% penalty for 1 day late — CS2009: Design and Analysis of Algorithms (Fall 2022)
40% penalty for 2 days late — Assignment 1
Submission not allowed afterwards — Total Marks: 100

1. Show the steps insertion sort uses to sort the following list of integers in the descending order (from the highest to the lowest / biggest to the smallest):

   $$6, 16, 12, 27, 9, 1, 18, 5, 31$$

   Show the value of the key variable, k, at each step. Explain briefly why time complexity of insertion sort is $O(n^2)$. Use Loop invariant to show its correctness. [5 Points]

2. Show the steps merge sort uses to sort the following list of integers in the descending order (from the highest to the lowest / biggest to the smallest): [5 points]

   $$6, 16, 12, 27, 9, 1, 18, 5, 31$$

   Consider the following variation on Merge Sort, that instead of dividing input in half at each step of Merge Sort, you divide into three part, sort each part, and finally combine all of them using a three-way merge subroutine. What is the overall asymptotic running time of this algorithm? [5 points]

3. Repeat for Quick Sort. Use Loop invariant to show its correctness. [5 points]

   $$6, 16, 12, 27, 9, 1, 18, 5, 31$$

4. Suppose you're consulting for a bank that's concerned about fraud detection, and they come to you with the following problem. They have a collection of n bank cards that they've confiscated, suspecting them of being used in fraud. Each bank card is a small plastic object, containing a magnetic stripe with some encrypted data, and it corresponds to a unique account in the bank. Each account can have many bank cards corresponding to it, and we'll say that two bank cards are equivalent if they correspond to the same account.

   It's very difficult to read the account number off a bank card directly, but the bank has a high-tech "equivalence tester" that takes two bank cards and, after performing some computations, determines whether they are equivalent.

   Their question is the following: among the collection of n cards, is there a set of more than n/2 of them that are all equivalent to one another? Assume that the only feasible operations you can do with the cards are to pick two of them and plug them in to the equivalence tester.

   Design an $O(n \log_2 n)$ algorithm to solve this problem. [5 points]

   Design linear-time $O(n)$ algorithm for solving above problem [5 points]

5. Take a sequence of 2n real numbers as input. Design an $O(n \log_2 n)$ algorithm that partitions the numbers into n pairs, with the property that the partition minimizes the maximum sum of a pair. For example, say we are given the numbers (1,3,5,9). The possible partitions are ((1,3),(5,9)), ((1,5),(3,9)), and ((1,9),(3,5)). The pair sums for these partitions are (4,14), (6,12), and (10,8). Thus the third partition has 10 as its maximum sum, which is the minimum over the three partitions. [10 points]

6. Prove $n^3 - 2n + 1 = O(n^3)$. Determine the values of constant c and n_0. [5 Points]
   Prove $5n^2 \log_2 n + 2n^2 = O(n^2 \log_2 n)$. Determine the values of constant c and n_0. [5 Points]

7. Watch the video lecture on Big O, Big Ω and Big Θ notation from

   http://www.youtube.com/watch?v=6Ol2JbwoJp0. Write the summary of the lecture in your words. [10 Points]

8. Use Master Theorem, to calculate the time complexity of the following [15 points]

$$T(n) = 2T(\frac{n}{3}) + c.n^2. \tag{1}$$

$$T(n) = 4T(\frac{n}{3}) + c.n. \tag{2}$$

$$T(n) = 8T(\frac{n}{2}) + c.n^3. \tag{3}$$

9. Use Iteration Method, to calculate the time complexity of the following [10 points]

$$T(n) = 2T(\frac{n}{3}) + n^2, (T(1) = 1). \tag{4}$$

$$T(n) = 4T(\frac{n}{3}) + n, (T(1) = 1). \tag{5}$$

10. For each of the following questions, indicate whether it is T (True) or F (False) and justify using some examples e.g. assuming a function? [15 Points]

    - For all positive $f(n), g(n)$ and $h(n)$, if $f(n) = O(g(n))$ and $f(n) = \Omega(h(n))$, then $g(n) + h(n) = \Omega(f(n))$.
    - Let f (n) and g(n) be asymptotically nonnegative functions, then max(f (n), g(n)) = Θ (f (n) + g(n)).
    - if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, then we have $(f(n))^2 = O(g(n))^2$