The homework is due in class (during the first 10 minutes) on Thursday, 25th October (or before). The problems are from the book by Rosen, 6th edition (sections 3.2, 3.3). As mentioned in the course syllabus, there are no electronic submissions and no late homeworks will be accepted unless you have an illness spanning the full period from the time the homework was assigned until it was due (and I shall need to see a medical practitioner’s certificate to that effect). Standard academic honesty rules apply. You can discuss problems with one another, but the solutions turned in should be entirely your own. Cases of plagiarism will be dealt with strictly. Also, make sure you write your name and section number on the first homework sheet and also staple all the sheets together. UNSTAPLED OR UNNAMED HOMEWORKS WILL NOT BE GRADED.

1 Time Complexity

1. Section 3.2: 2(a),(c),(d),(f) [Give reasons for all your answers], 6

2. Section 3.2: 18 [Hint: This is very similar to the problem where we proved \( n! \) is \( O(n^n) \)], 44 [Hint: First prove that \( f(x) \) is \( O(x^n) \), as done in class. Then prove that \( f(x) \) is \( \Omega(x^n) \)].

3. Section 3.2: 40

2 Numerical Algorithms
Section (3.3): 4, 7, 8

3 Searching and Sorting

1. Ternary search is a method that locates a particular element \( q \) in a sorted list given as \( \{a_1, a_2, ..., a_n\} \) by splitting the list into three sublists of equal size (or approximately equal size). The list is split at two indices, which we call as \( s_1 \) and \( s_2 \) respectively. Based on a comparison of the elements \( a_{s_1} \) and \( a_{s_2} \) with \( q \), we decide in which of the three sublists, \( q \) could possibly be found. We throw away the other two sublists and continue the same process on the selected sublist until the element \( q \) is found, or the list is empty. Write this procedure in pseudo-code. Analyze its worst case time complexity. Is this algorithm faster/slower than binary search in terms of time complexity (ignoring constants)?
2. Insertion sort is a procedure which sorts an array of \( n \) numbers by successively maintaining smaller sorted arrays and adding new elements in such a way that the new array remains sorted after inclusion of the new element. Suppose \( \{a_1, a_2, \ldots, a_n\} \) is the array of numbers to be sorted. It starts off by building an array with just one element \( a_1 \). The next element \( a_2 \) is added either before or after \( a_1 \) such that this array with two elements is sorted in the correct order. The same procedure is applied to \( a_3 \) to produce a correctly sorted array with three elements. This whole procedure is repeated for \( n \) elements in total. Write a pseudo-code for insertion sort and analyse its time complexity. There are two aspects to this algorithm: firstly locating where to insert, and secondly re-arranging the array appropriately. Do not blindly copy the pseudo-code from the book, but work it out yourself and show the two aforementioned components clearly.

4 Optional Problems, which you should do, but do not need to submit

1. Supplementary exercise: 8 (page 259)

2. Supplementary exercise: 16 (page 259) [Hint: Look at the table of standard summation in chapter 2]

3. Let \( f(n) = n^{0.01} \) and \( g(n) = \log(n) \). Can I say \( f(n) \) is \( O(g(n)) \)? Can I say \( g(n) \) is \( O(f(n)) \)? Why?