Please review carefully the instructions given for Homework 1. They apply to this assignment, too.

Please hand in your answers to the following problems. Problem numbers, where indicated, are from the seventh edition of the Rosen text.

1. (6 points) Let \( T = t_1 t_2 \ldots t_n \) and \( P = p_1 p_2 \ldots p_m, \) \((m \leq n)\) be two strings composed of letters from (say) the English alphabet. We wish to decide whether \( P \) is a substring of \( T, \) i.e., whether the letters of \( P \) appear consecutively in \( T. \) (For instance, if \( T \) is MINNESOTA and \( P \) is INNE, then \( P \) is a substring of \( T; \) if \( P \) is INEN or INES, then it is not.) This is a common task in text editors. Your goal is to develop a simple algorithm that outputs “yes” or “no” depending on whether or not \( P \) is a substring of \( T. \) Your algorithm should make no more than \( m \times n \) comparisons between letters from \( P \) and \( T. \)

(a) Describe your approach in a few sentences and give pseudocode. Your style of pseudocode should be similar to the one in the text or seen in class; do not give full-blown C/C++/Java, etc.

(b) Analyse your algorithm and show that it makes no more than the stated number of comparisons between letters of the two strings. (Ignore any other type of operation.)

2. (8 points) Let \( A[1:n] \) be an array containing a sequence of distinct integers in some arbitrary order. We wish to find both the minimum and the maximum element in \( A. \)

(a) Give an algorithm for this problem that makes just one pass over \( A \) and determines the minimum and maximum using a total of at most \( 2n - 2 \) comparisons between elements of \( A. \) (Ignore any other type of operation.) Describe your approach in a few sentences, give pseudocode, and establish the stated bound on the number of comparisons made by your algorithm. (See Problem 1’s statement for the style of pseudocode to use.)

(b) Suppose now that you are allowed to make more than one pass over \( A. \) Show how to find both the minimum and the maximum using \( 3(n/2) - 2 \) element comparisons. (For simplicity, assume that \( n \) is even here.) You may explain your approach in words but be sure that it is clear (pseudocode is not required). Establish the stated bound on the number of comparisons.

3. (5 points) Use the definition of “\( f(n) = O(g(n)) \)” rather than any known theorem(s), to establish each of the following bounds. Show your work, including the witnesses \( c \) and \( k. \) (Use the definition of “big-O” given in class instead of the one on page 205 of the text.)

- \( 2^n + 17 \) is \( O(3^n) \)
- \( n^2 + 2n + 3 \) is \( O(n^3) \) but \( n^3 \) is not \( O(n^2 + 2n + 3) \)
4. (5 points) Give a big-$O$ bound for each of the functions below and justify using the appropriate theorem(s) from Sec. 3.2. Show your work. (Do not expand the expressions below; instead apply the appropriate theorem(s) to the subexpressions directly and simplify/combine the results.)

- $(n^2 + n)(1 + \log_2 n) + (n^3 + 19)$
- $(2^n + n^2)(n + 3^n)$

5. (5 points) Determine the exact number of times the print operation is executed by the function below. (Simplify your expression as much as possible.) In your calculation, reference each print operation by its line number and count the number of times it is executed. Also express your count using big-$O$ notation. Assume that $n > 0$.

```plaintext
1. function Friendly(n)
2.   for i = 1 to n do
3.     print("hello")
4.   end
5.   for j = i+1 to n do
6.     print("hello")
7.   end
8.   for k = 1 to n-1 do
9.     print("hello")
10. end
11. for m = 1 to 10 do
12.   print("hello")
13. end
14. end
```

6. (6 points) Let $A[1 : n]$ be an array containing a sequence of distinct integers. Consider the following algorithm, called selection sort, to sort $A$ in increasing order: In the first pass, the smallest element is found and moved to the first position. In the second pass, the smallest of the remaining elements is found and moved to the second position. And so on.

Express this algorithm in psuedocode, determine the exact number of comparisons made between elements of $A$ (simplifying your expression as much as possible), and then express this using big-$O$ notation. Ignore any other type of operation done by the algorithm. (See Problem 1’s statement for the style of pseudocode to use.)