For the programming problems below, include in your hardcopy submission a listing of your algorithm and of the output. Please follow attached submission instructions.

1. (U & G-required) [20 points] Consider the following algorithm.

   **ALGORITHM Enigma** (A[0..n − 1])
   //Input: An array A[0..n – 1] of integer numbers
   for i ← 0 to n − 2 do
      for j ← i + 1 to n − 1 do
         if A[i] == A[j]
            return false
         return true

   a) [5 points] What does this algorithm do?
   b) [15 points] Compute the running time of this algorithm.

2. (U & G-required) [40 points]
   a) [30 points] Implement in C/C++ a version of bubble sort that alternates left-to-right and right to left passes through the data. For example, if sorting the array [6, 5, 2, 8, 3, 1], the first left-to-right pass will swap elements that are out of order and get the result: [5, 2, 6, 3, 1, 8]. The right-to-left pass begins at element 1 (on position 5) and goes all the way to the beginning of the array. At the end of this pass we have [1, 5, 2, 6, 3, 8]. The next phase works only on the segment [5, 2, 6, 3] as elements 1 and 8 have already been placed at their final location.

   Show how your algorithm sorts the following array: “EASYQUESTION”. Print the status of the array at the end of each left-to-right and right-to-left pass.

   b) [10 points] How many comparisons does this modified version of bubble sort make?
3. (U & G-required) [40 points] (U-required)
The Mergesort algorithm we discussed in class is a recursive, divide-and-conquer algorithm in which the order of merges is determined by its recursive structure. However, the subarrays are processed independently and merges can be done in different sequences. Implement in C/C++ a non-recursive version of Mergesort, which performs a sequence of passes over the entire array doing \( m \)-by-\( m \) merges, doubling \( m \) at each pass. (Note that the final merge will be an \( m \)-by-\( n \) merge, if the size of the array is not a multiple of \( m \). Show how your algorithm sorts the sequence “ASORTINGEXAMPLE”. At the end of each \texttt{merge} step print the values in the resulting subarray.

For example, if sorting [3, 2, 5, 1]:
- At first pass (1-by-1 merges): 3 and 2 are merged \( \rightarrow [2, 3] \)
  5 and 1 are merged \( \rightarrow [1, 5] \)

At next pass (2-by-2 merges): [2, 3] and [1, 5] are merged \( \rightarrow [1, 2, 3, 5] \)

4. (G-Required) [20 points] Use a loop invariant to prove that the following algorithm computes \( a^n \):

```c
Exp \( (a, n) \) {
    i \leftarrow 1
    pow \leftarrow 1
    while ( i \leq n ) {
        pow \leftarrow pow*a
        i \leftarrow i + 1
    }
    return pow
}
```
Extra credit

5. [20 points]
Consider another algorithm for solving the same problem as the one in Homework 2 (problem 1), which recursively divides an array into two halves (call Min2 (A[0..n − 1])):

**ALGORITHM Min2 (A[left..right])**

```plaintext
if left = right return A[left]
else temp1 ← Min2 (A[left..\lfloor(left + right)/2\rfloor])
    temp2 ← Min2 (A[\lceil(left + right)/2\rceil+1..right])
if temp1 ≤ temp2 return temp1
else return temp2
```

a) [10 points] Set up a recurrence relation for the algorithm’s basic operation count and solve it.

b) [5 points] Which of the algorithms Min1 (from Homework 2) or Min2 is faster?