ECE368 Fall 2016 Homework #2

IMPORTANT:

- Do NOT leave your name or Purdue ID on this homework.
- Write your homework security number at the TOP of EACH page.

Read and sign the ***Academic Honesty Statement*** that follows:

"In signing this statement, I hereby certify that the work on this exercise is my own and that I have not copied the work of any other student while completing it. I understand that, if I fail to honor this agreement, I will receive a score of zero for this exercise and will be subject to further disciplinary action."

`Homework security number:`

Please acknowledge any people who have helped you with this homework.

Question	Credits
1	
2	
3	
4	

1. (25 points) Does the bidirectional bubble sorting algorithm reduce the number of swaps when compared to the original bubble sort? Does it reduce the total number of comparisons required to sort an array? Justify your answers.

2. (25 points) Consider the following procedure acting on an array $A[1..n]$.

```
INSERTION_SORT(A[1..n])                          Cost   Times
1.      for j ← 2 to n                           C1
2.          key ← A[j]                           C2
3.          i ← j − 1                            C3
4.          while i > 0 and A[i] > key           C4
5.              A[i+1] ← A[i]                    C5
6.              i ← i − 1                        C6
7.          A[i+1] ← key                         C7
```

(a) Let t_j denote the number of times the while loop test in line 4 is executed for that value of j. Fill in for each line of instruction, the number of times the instruction is executed.

(b) Derive the expression for the running time of INSERTION_SORT in terms of n, C_i, and t_j.

(c) What is t_j for the best-case scenario, i.e., when the running time of the algorithm is the smallest. Use that to derive the expression for the best-case running time of INSERTION_SORT in terms of n and C_i. What is the best-case time complexity of INSERTION_SORT using the big-O notation?

(d) What is t_j for the worst-case scenario, i.e., when the running time of the algorithm is the largest. Use that to derive the expression for the worst-case running time of INSERTION_SORT in terms of n and C_i. What is the worst-case time complexity of INSERTION_SORT using the big-O notation?

3. (25 points) Let $A[1..n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, the pair (i, j) is called an *inversion* of A.

(a) List all the inversions of the array $\langle 2, 3, 8, 6, 1 \rangle$.

(b) What array with elements from the set $\{1, 2, ..., n\}$ has the most inversions? How many does it have?

(c) What is the relationship between the running time of `INSERTION_SORT` (see question 2) and the number of inversions in the input array? Justify your answer.

(d) Suppose we are comparing implementations of insertion sort and merge sort (a more advanced sorting algorithm, which we will learn about later in the semester) on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in $64n \log_2 n$ steps. For which values of n does insertion sort beat merge sort?

4. (25 points) Consider the following procedure that performs multiplication of two upper triangular matrices $A[1..n][1..n]$ and $B[1..n][1..n]$.

```
MATRIX_MULTIPLY(A[1..n][1..n], B[1..n][1..n])   Cost   Times
1.      for i ← 1 to n                           C1
2.          for j ← i to n                       C2
3.              c_ij ← 0                         C3
4.              for k ← i to j                   C4
5.                  c_ij ← c_ij + a_ik · b_kj    C5
6.      return C                                 C6
```

As the product $C[1..n][1..n]$ is also upper triangular, we assume that we do not have to be concerned with the strictly lower triangular entries of the matrix.

(a) Fill in for each line of instruction, the number of times the instruction is executed.

(b) Derive the expression for the running time of `MATRIX_MULTIPLY` in terms of n and C_i. What is the complexity of the algorithm using the big-O notation?