1. (2 points) Showing your work, determine a closed form for the worst-case time complexity of the following sort algorithm:

```cpp
template <class Etype> void shrinkSort(Etype A[], int n)
{
    int j, k;
    int minIndex, maxIndex;
    Etype temp;
    for (j = 0; j <= (n/2) - 1; j++)
    {
        minIndex = j;
        maxIndex = n-j-1;
        for (k = j; k <= n-j-1; k++)
        {
            if (A[k] < A[minIndex])
                minIndex = k;
            if (A[k] > A[maxIndex])
                maxIndex = k;
        }
        if (minIndex != j)
        {
            temp = A[minIndex];
            A[minIndex] = A[j];
            A[j] = temp;
        }
        if (maxIndex != n-j-1)
        {
            temp = A[maxIndex];
            A[maxIndex] = A[n-j-1];
            A[n-j-1] = temp;
        }
    }
}
```

2. (2 points each) A binary tree insertion sort inserts elements into a binary search tree and then traverses the tree using an inorder traversal, inserting the elements encountered into an array.

   (a) Showing your work, determine the big-O time complexity of a worst-case situation for sorting a list of \(n\) elements with this algorithm. Give an example of an \(n\)-element list that requires this amount of time.

   (b) Showing your work, determine the big-O time complexity of a best-case situation for sorting a list of \(n\) elements with this algorithm. Give an example of an \(n\)-element list that requires this amount of time.

3. (2 points each) Consider the following algorithm for sorting a list:
   I. First, make a copy of the list to be sorted.
   II. For each element of the copied list, count how many elements of the copied list are less than it (call this number \(\text{count}\)).
   III. Insert the element in question into the \(\text{count}\)-indexed slot of the original list.
   IV. Repeat this process (steps II & III) for every element of the copied list.

   (a) Showing your work, determine the worst-case time complexity of this algorithm (including an example of an \(n\)-element list that requires that worst-case time).

   (b) Note that this algorithm will only work correctly if the list contains no duplicates. Specify the modifications needed to make this algorithm work correctly even in the presence of duplicates.
4. (5 points) The Weiss text illustrates that since there are \( n! \) ways to order \( n \) objects, any comparison-based algorithm to sort \( n \) items will result in a decision tree with at least \( n! \) leaf nodes, which, consequently has a depth of at least \( \lceil \log_2 n! \rceil \). Therefore, any comparison-based algorithm to sort \( n \) items will require \( \lceil \log_2 n! \rceil \) comparisons.

Note that \( \lceil \log_2 4! \rceil = 5 \). Below is the outline of an algorithm that is guaranteed to sort 4 items, \( \alpha_1, \alpha_2, \alpha_3, \) and \( \alpha_4 \) in 5 comparisons.

**Step 1.** Compare \( \alpha_1 \) and \( \alpha_2 \), swapping their values if necessary to make \( \alpha_1 \leq \alpha_2 \).

**Step 2.** Compare \( \alpha_3 \) and \( \alpha_4 \), swapping their values if necessary to make \( \alpha_3 \leq \alpha_4 \). (Note that this guarantees that one of the following is true: \( \alpha_1 \leq \alpha_2 \leq \alpha_3 \leq \alpha_4, \alpha_1 \leq \alpha_3 \leq \alpha_2 \leq \alpha_4, \alpha_1 \leq \alpha_3 \leq \alpha_4 \leq \alpha_2, \alpha_3 \leq \alpha_4 \leq \alpha_2 \leq \alpha_1 \leq \alpha_2 \leq \alpha_1 \leq \alpha_4 \leq \alpha_3 \leq \alpha_2 \) or \( \alpha_3 \leq \alpha_1 \leq \alpha_2 \leq \alpha_4 \).)

**Step 3.** Compare \( \alpha_1 \) and \( \alpha_3 \), swapping their values if necessary to make \( \alpha_1 \leq \alpha_3 \). (Note that this guarantees that one of the following is true: \( \alpha_1 \leq \alpha_2 \leq \alpha_3 \leq \alpha_4, \alpha_1 \leq \alpha_3 \leq \alpha_2 \leq \alpha_4, \alpha_1 \leq \alpha_3 \leq \alpha_4 \leq \alpha_2, \alpha_1 \leq \alpha_4 \leq \alpha_2 \leq \alpha_3 \leq \alpha_2 \) or \( \alpha_1 \leq \alpha_3 \leq \alpha_4 \leq \alpha_2 \).)

**Step 4.** Compare \( \alpha_2 \) and \( \alpha_4 \), swapping their values if necessary to make \( \alpha_2 \leq \alpha_4 \). (Note that this guarantees that one of the following is true: \( \alpha_1 \leq \alpha_2 \leq \alpha_3 \leq \alpha_4 \) or \( \alpha_1 \leq \alpha_3 \leq \alpha_2 \leq \alpha_4 \).)

**Step 5.** Compare \( \alpha_2 \) and \( \alpha_3 \), swapping their values if necessary to make \( \alpha_2 \leq \alpha_3 \). (Note that this guarantees that \( \alpha_1 \leq \alpha_2 \leq \alpha_3 \leq \alpha_4 \).)

Now note that \( \lceil \log_2 5! \rceil = 7 \). Using the same level of detail as above, outline an algorithm that is guaranteed to sort 5 items, \( \beta_1, \beta_2, \beta_3, \beta_4 \), and \( \beta_5 \) in 7 comparisons.

**You must provide your own solutions to these problems in a clearly presented Word document.**

**Obtaining solutions from any outside source is considered academic misconduct.**

**The only person with whom you may discuss these problems is the instructor.**