Ron Cox

HW8
605.202.81
Data Structures

1. Insertion Sort Comparisons and Interchanges

i. Sorted File
Comparisons: Each element is compared once with the previous one. Since it's already sorted, there are n - 1 comparisons and no interchanges.

ii. Reverse Sorted File
Comparisons: Each element needs to be compared with all elements before it, leading to n(n-1)/2 comparisons. Each comparison leads to an interchange, so interchanges are also n(n-1)/2.

iii. Alternating Smallest and Largest Elements
Comparisons and Interchanges: The largest elements have to be moved across the array, resulting in close to O(n^2) comparisons and interchanges as in a worst-case scenario.

2. Shell Sort Comparisons and Interchanges (Increments 2 and 1)

i. Sorted File
Comparisons and Interchanges: Minimal because the data is already in order; only a few comparisons to verify this.

ii. Reverse Sorted File
Comparisons and Interchanges: Fewer than n(n-1)/2 due to the larger initial gaps helping move elements closer to their correct position earlier.

iii. Alternating Smallest and Largest Elements
Comparisons and Interchanges: Fewer than O(n^2) because the gaps will help move elements into place more efficiently.

3. Merging Two Ordered Files

a. m = n and a[i] < b[i] < a[i+1]
Comparisons: Each element from both arrays needs to be compared, leading to n + m comparisons.

b. m = n and a[n] < b[1]
Comparisons: All elements of a are compared before starting with b, resulting in n comparisons.

4. Merging Two Ordered Files with Specific Conditions

a. m = n and a[n/2] < b[1] < b[m] < a[(n/2)+1]
Comparisons: Requires comparing all elements from both arrays, leading to m + n comparisons.

b. m = 1 and b[1] < a[1]
Comparisons: Only one comparison is needed to place b[1] before a[1].

c. m = 1 and a[n] < b[1]
Comparisons: All elements of a are processed before b[1], resulting in n comparisons.