Ex 1:

(a) BubbleSort

3	5	7	11	41	13	17	22	19	58
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

(b) SelectionSort

3	5	7	11	17	41	19	22	58	13
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

(c) InsertionSort

7	11	17	22	41	3	19	5	58	13
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Ex 2:

58	41	19	22	17	3	11	5	7	13
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Ex 3:

(a) Bubble sort

(b) Selection sort

(c) Insertion Sort

Ex 4:

Nbr. of comparisons: 4950 = N(N-1)/2

Ex 5:

To sort the file in descending order, collect the queues from Queues[9] DOWNTO Queues[0].

Ex 6:

(a) for the best case. O(*N*)
(b) for the worst case. O(*N*)

Ex 7:

LSD Radix sort does not provide correct results when there are negative elements.

```
Radix Sort(A,d)
        Let arrays P and N to present positive and negative integers respectively.
        j = 0; k = 0
        for i = 0 to A.length {
                if A[i] < 0 {
                        N[j] = A[i] * (-1)
                        j =  j + 1
                 }
                Else {
                        P[k] = A[i]
                        K = k +1
                }
         }
        RadixSort (P, d)
        RadixSort (N, d)
        Reverse(N)
        For m = 1 to N.length
             N[m] = N[m] * (-1)
        Concatenate (N, P)
```

Ex 8:

$O(N^2)$ since the array is already sorted.

Ex 9:

1. *The Base-Case Question*: The base case occurs when First >= Last. In this case, the part of the list left to sort contains 0 or 1 element; that is, it is already sorted. The answer is yes.

2. *The Smaller-Caller Question:* Each of the two recursive calls sorts half of the list. In each recursive call, the list is again split in half, until the "half" being sorted has 0 or 1 element. At this point, as we verified with the base-case question above, we have reached the smallest case, and no further recursive calls are made. The answer is yes.

3. *The General-Case Question:* Assuming that the first recursive call to MergeSort successfully sorts the first half of the list and that the second recursive call successfully sorts the second half, the whole list is sorted when the two sorted halves are merged into one. Assuming that procedure Merge completes this task successfully, the answer is yes.

Ex 10:

Only main needs to be changed, so just the relevant portion of its code is included here.

```
        cout << "Enter minimum chunk size (<= " << MAX_ITEMS << "): ";
        int minchunck;
```

```
cin >> minchunk;
int maxthreads = thread::hardware_concurrency();
chunk = MAX_ITEMS / maxthreads;
if (minchunk >= chunk)
    chunk = minchunk;
start = chrono::system_clock::now(); // Record start time
ParallelMergeSort<int>(numbers, 0, MAX_ITEMS-1, temp, chunk); // Run sort
end = chrono::system_clock::now(); // Record end time
chrono::duration<float> elapsed = end-start; // Report time
cout << "Execution time in seconds = " << elapsed.count() << "\n";
return 0;
```