(a) The merge can be done by examining the elements of the two input lists `A` and `B` left to right. The smaller element is copied into the result list. The member function is given below. Variables `ca` and `cb` act as cursors that move through `A` and `B` left to right. `ct` is a cursor that keeps track of the location for the next element of the result. The code throws an exception if it is unsuccesful. This happens iff there isn't enough space in the result list.

```
template <class T>
LinearList<T>& LinearList<T>::
    Merge(const LinearList<T>& A, const LinearList<T>& B)
{// Merge the two sorted lists A and B
   int al = A.Length();
   int bl = B.Length();
   length = al + bl;  // length of result
   if (length > MaxSize) throw NoMem();
                      // inadequate space for result

   int ca = 0; // cursor for A
   int cb = 0; // cursor for B
   int ct = 0; // cursor for *this
   while ((ca < al) && (cb < bl)) {
      if (A.element[ca] <= B.element[cb])
         element[ct++] = A.element[ca++];
      else element[ct++] = B.element[cb++];
      }

   // take care of left overs
   if (ca == al)  // A is finished
      for (int q = cb; q < bl; q++)
         element[ct++] = B.element[q];
   else for (int q = ca; q < al; q++)
           element[ct++] = A.element[q];

   return *this;
}
```

(b) When there is inadequate space for the result, an exception is thrown and the complexity is $\Theta(1)$. Assume we have enough space. In each iteration of the `while` loop the value of `ct` increases by one. On exiting this loop, `ct` < `al+bl`. The time spent in the `for` loops is either O(`al`) or O(`bl`). So, the complexity is O(`al+bl`). Also, since all `al+bl` elements are moved into the result list, the complexity is Ω(`al+bl`). As a result, the complexity is Θ(`al+bl`).

(c) The test program and output can be found in the files `elist.h`, `merge.cpp`, and `merge.out`.