Mergesort is probably the first sub-quadratic sorting algorithm. It was devised by John von Neumann in 1945. It is an excellent example of the divide-and-conquer algorithmic paradigm.

Conceptually, this scheme works as follows:

```python
if the given (unsorted) list is of length one,
    then it is already sorted, so return;
otherwise
    
    divide the list into two sublists of about half the size;
    sort each sublist recursively by applying merge sort;
    merge the two (sorted) sublists into one sorted list;
    return;
```

Mergesort incorporates two main ideas that lead to an improved performance:

1. A small list will take fewer steps to sort than a large list.
2. Fewer steps are required to construct a sorted list from two sorted lists than from two unsorted lists. For example, you only have to traverse each list once if they're already sorted. (An example and an implementation appear below.)

**Method of attack:** Suppose we have an array $A$ consisting of $n$ integers $A_0$ through $A_{n - 1}$, and let $c$ be the integer part of $n / 2$. Apply merge sort to the sub-arrays $A_0 .. A_c$ and $A_{c+1} .. A_{n-1}$. When the two halves are returned, they will have been sorted. They can now be merged together to form a sorted array.

An example appears next.
Example: Perform *merge-sort* on the following sequence of integers that is to be sorted into ascending order: 56, 34, 65, 70, 19, 21, 95, 12.

Note: Elements appearing in the red in the foregoing example correspond to a (sub)array that is sorted by itself.

<table>
<thead>
<tr>
<th>Data Structure:</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Complexity:</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Space Complexity:</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

An implementation in C++ appears next.
void mergeSort(DataType theArray[], int first, int last);
/* * sorts the items in an array into ascending order.
 * Pre: theArray[first..last] is an array.
 * Post: theArray[first..last] is sorted in ascending order.
 * theArray is the given array.
 * first is the first element to consider in theArray.
 * last is the last element to consider in theArray.
 */

void merge(DataType theArray[], int first, int mid, int last);
/* * merges two sorted array segments theArray[first..mid] and
 * theArray[mid+1..last] into one sorted array.
 * Pre: first <= mid <= last.
 * The subarrays theArray[first..mid]
 * and theArray[mid+1..last] are each sorted in increasing order.
 * Post: theArray[first..last] is sorted.
 * theArray is the given array.
 * first is the beginning of the first segment in theArray.
 * mid is the end of the first segment in theArray.
 * mid + 1 marks the beginning of the second segment.
 * last is the last element in the second segment in theArray.
 * Note: This function merges the two subarrays into a temporary
 * array and copies the result into the original array theArray.
 */

int main()
{
    DataType A[MAX_SIZE];
    ifstream inp;
    inp.open("input.dat"); // Input comes from a file.

    int i = 0;
    while(i < MAX_SIZE)
    {
        inp >> A[i];
        i++;
    }
    inp.close();

    mergeSort(A, 0, MAX_SIZE-1);

    ofstream outp; outp.open("output.dat"); // Output goes to a file.
    i = 0;
    while(i < MAX_SIZE)
    {
        outp << A[i] << " ";
        i++;
    }
    outp << endl;
    outp.close();
    return 0;
} // end of main
void mergeSort(DataType theArray[], int first, int last)
{
    if (first < last)
    {
        int mid = (first + last)/2;    // index of midpoint
        mergeSort(theArray, first, mid); // recursively sort left half.
        mergeSort(theArray, mid+1, last); // recursively sort right half.
        merge(theArray, first, mid, last); // merge the two.
    } // end if
} // end mergesort

void merge(DataType theArray[], int first, int mid, int last)
{
    DataType tempArray[MAX_SIZE]; // temporary array

    int first1 = first;       // beginning of first subarray
    int last1  = mid;         // end of first subarray
    int first2 = mid + 1;     // beginning of second subarray
    int last2  = last;        // end of second subarray

    // while both subarrays are nonempty, copy the
    // smaller item into the temporary array
    int index = first1; // next available location in tempArray
    for (; (first1 <= last1) && (first2 <= last2); ++index)
        { // Invariant: tempArray[first..index-1] is in order
            if (theArray[first1] < theArray[first2])
                { tempArray[index] = theArray[first1];
                  ++first1;
                }
            else
                { tempArray[index] = theArray[first2];
                  ++first2;
                } // end if
        } // end for

    // finish off the first subarray, if necessary
    for (; first1 <= last1; ++first1, ++index)
        { // Invariant: tempArray[first..index-1] is in order
            tempArray[index] = theArray[first1];
        }

    // finish off the second subarray, if necessary
    for (; first2 <= last2; ++first2, ++index)
        { // Invariant: tempArray[first..index-1] is in order
            tempArray[index] = theArray[first2];
        }

    // copy the result back into the original array
    for (index = first; index <= last; ++index)
        { theArray[index] = tempArray[index];
        } // end merge