The maximum number of comparisons per iteration is k m,(k is high –low, mm is the middle) and there are n / k iterations, so the total complexity is (k m) * (n / k) = n (nm) / k, and (nlogn), is the original merge sorting complexity (nlogn), when k = 1 is n ^ 2.

```
private static void merge(int[] a, int low, int mid, int high)
{
    int i = low;
    int j = mid+1;

    for(; i <= mid && j <= high; )
    {
        if(a[i] < a[j])
        {
            i++;
        }
        else
        {
            int temp = a[j];
            for(int l = j - 1; l >= i ; l--)
            {
               a[l+1] = a[l];
            }
            a[i] = temp;
            i++;
            j++;
            mid++;
        }
    }

}
```

It’s stable, even in an efficient implementation. When merging two halves, just make sure to use “L <= R” so that you favor left-half values over right-half values, if they are equal.
Stability — and the fact that it doesn’t have “potentially pathological performance” in the worst case

```
int mid = (low + high)/2;
if(low < high)
{
    MergeSort(a, low, mid);
    MergeSort(a, mid+1, high);
    merge(a, low, mid, high);
}
```