Recall that a comparison in any sort is any comparison of magnitude of any two entries in a list and which may or may not result in a swap of two values in a list.

8.5a The following is an implementation of merge sort.

```cpp
template <typename Type>
void merge_sort( Type *array, int n ) {
    merge_sort( array, 0, n - 1 );
}

template <typename Type>
void merge_sort( Type *array, int a, int b ) {
    if ( a >= b )
        return;

    int mid = (a + b)/2;
    merge_sort( array, a, mid );
    merge_sort( array, mid + 1, b );
    merge( array, a, mid, b );
}
```

Overloading in C++ is where two functions have the same name but different signatures. What is the purpose of overloading the function `merge_sort`?

8.5b Implement the function `merge` used in the above implementation of merge sort:

```cpp
template <typename Type>
void merge( Type *array, int a, int mid, int b ) {  
```
8.5c Rewrite the above function so that if the size of the interval being sorted is less than or equal to the static constant `USE_INSERTION_SORT`, which is set to a positive integer greater than or equal to 1.

8.5d Show the steps in applying merge sort where `USE_INSERTION_SORT` is set to 5.

<table>
<thead>
<tr>
<th>72</th>
<th>92</th>
<th>79</th>
<th>38</th>
<th>84</th>
<th>76</th>
<th>83</th>
<th>72</th>
<th>15</th>
<th>35</th>
<th>57</th>
<th>29</th>
<th>91</th>
<th>42</th>
<th>48</th>
<th>67</th>
</tr>
</thead>
</table>

Show the entries prior to each of the successive merges. The last entry has been created for you.

|                  |                  |                  |                  |                  |                  |                  |                  |                  |                  |                  |                  |                  |                  |                  |                  |
|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|

<table>
<thead>
<tr>
<th>15</th>
<th>29</th>
<th>35</th>
<th>38</th>
<th>42</th>
<th>48</th>
<th>57</th>
<th>67</th>
<th>72</th>
<th>72</th>
<th>76</th>
<th>79</th>
<th>83</th>
<th>84</th>
<th>91</th>
<th>92</th>
</tr>
</thead>
</table>
8.5e Merge sort requires a temporary array of size $\Theta(n)$. If each time merge is called, a new array is allocated, this could be very expensive. Instead, consider the following implementation:

```cpp
// Merge sort with temporary array

template <typename Type>
void merge_sort( Type *array, int n ) {
    merge_sort( array, 0, n - 1 );
}

template <typename Type>
void merge_sort( Type *array, int a, int b ) {
    if ( a >= b ) {
        return;
    }

    Type *tmp_array = new Type[b - a + 1];
    int mid = (a + b)/2;
    merge_sort_internal( array, tmp_array, a, mid );
    merge_sort_internal( array, tmp_array, mid + 1, b );
    merge( array, tmp_array, a, mid, b );
    delete [] tmp_array;
}

template <typename Type>
void merge_sort_internal( Type *array, Type *tmp_array, int a, int b ) {
    if ( a >= b ) {
        return;
    }
    int mid = (a + b)/2;
    merge_sort_internal( array, tmp_array, a, mid );
    merge_sort_internal( array, tmp_array, mid + 1, b );
    merge( array, tmp_array, a, mid, b );
}
```

Re-implement the `merge` function so that it uses the entries from $0$ to $b - 1 + 1$ in this temporary array to perform the merge and then copy the values back into `array`. 