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.

8.5*a* The following is an implementation of merge sort.

```
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`?

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

```
template <typename Type>
void merge( Type *array, int a, int mid, int b ) {


















}
```

8.5*c* 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.

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

72	92	79	38	84	76	83	72	15	35	57	29	91	42	48	67

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

15	29	35	38	42	48	57	67	72	72	76	79	83	84	91	92

8.5*e* Merge sort requires a temporary array of size Θ(*n*). If each time merge is called, a new array is allocated, this could be very expensive. Instead, consider the following implementation:

```
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`.