# Question 2

The macros in the original Header file call the functions **qsort**, **bsearch** and **\_bsearch\_insert\_location** for different data types. For each data type there is also a dedicated compare function. To reduce the amount of code and to make it much easier to maintain, I have eliminated the macros and used C++ templates instead. I have written a class template with static methods: **compare\_fn**, **quick\_sort**, **binary\_search**, \_**bsearch\_insert\_location** (definition was not provided in the original C code) and **bsearch\_insert\_location** and **create\_unique\_list**. Thanks to the use of templates, these functions are valid for all types of data and there is no need to write different definitions for each data type. The functions are declared **static** so that an instance of the class is not necessary.

Because the compare function is used for all types of data, particular cases need to be taken into consideration. Provided that the following is the prototype for the compare function:

int compare\_fn(const void \*v, const void \*w);

If the return value is positive, then v > w, if it is negative then v < w, if it is 0 then v = w. This is determined simply by taking the difference between v and w. This works for all integer types, but for float and double it will not work, because if the modulus of the difference is less then 1 then the return value will be stored as 0, because of the implicit conversion from double/float to int. For example, if we have **a** = 0.4, **b** = 0.3, then **a**-**b** = 0.1 should signal the **qsort** function that **a** is greater than **b**, but instead it would return 0 and wrongly signal to **qsort** that a=b. This inconvenience is avoided if we store **a**-**b** in a temporary double variable and then check if the difference is between 0 and 1 or between -1 and 0. If it is between 0 and 1 then the compare function should return 1, if it is between -1 and 0, then it should return -1.

In the original C code there is a compare function for arrays of char\* (or strings). Essentially, the function uses the strcmp function to evaluate which string is “greater” than the other (which one comes after the other during the sorting process). In the modified compare\_fn, I have used an if statement to check whether the template argument is a char\*. If this is the case, then the function returns the return value of strcmp.

The original file uses a user defined data (probably a struct) called string\_t. I have provided a class called String (see String.h and String.cpp) which essentially is a modified version of std::string. The reason I did not use std::string is because in my own version of String I have overloaded the “-” operator like this:

int String::operator - (class String &other\_string) {

return strcmp(this->s, other\_string.s);

}

In this way we do not need to write an if statement in the compare function to check whether the template argument is a class of type String. Instead, when the statement

return (int )(\*a - \*b);

Is encountered in the compare function, the previously overloaded operator will be called. Note that the String class contains the function

static int string\_case\_compare(class String &a, class String &b);

which has no definition in the original source code, hence, it has been left undefined in the modified version as well.