This program uses 3 functions in order to perform the merge sort algorithm, a main function, a merge function which does the sorting, and a mergeSort helper function which makes function calls and splits the vector up. This program uses a vector instead of dynamically allocated arrays. I primarily chose a vector so we would not have to declare any set sizes, and vectors are also extremely versatile. Finally, this program has the option of reading through a file for numbers, or getting numbers through iostream's cin from the user.

The function starts in main, where the user is asked if they would like to enter a filename or test numbers entering them from the terminal. If neither option is selected, the program ends. If the user chooses 0, or to enter a filename, the user is asked to input the filename. The file is then opened using fstream, and all contents of the file are pushed into the vector. If the file can't be found, then the program outputs "Unable to open file". We then close the file and call the mergeSort function. If the user inputs 1, then they are asked to enter how many numbers they would like to input. The user can then enter the numbers they would like to be sorted. If the user continues to enter more numbers than what they put for the size, they aren't pushed to the vector. Then, we call mergeSort.

When first calling mergeSort, we take our filled vector, the leftmost point (always 0) and the rightmost point (always the size of the array - 1). The purpose of this function is to split the array into multiple smaller arrays, before calling the merge function to sort them from least to greatest. The first thing the function does is check if the sub-array has zero or one element, which means it is already sorted, and there is no need to perform further sorting. This is the base case of the recursive algorithm. If the base case is true, the function returns without doing anything. Then, if the sub-array has more than one element, the function calculates the middle index of the said array. This is done to avoid any overflow issues when finding the middle index.The first mergeSort call is to sort the left half of the sub-array to the middle. The second one is to sort the right half of the sub-array, from the middle.Once the two halves are sorted, the merge function is called to merge the two sorted halves back together to create the final sorted sub-array.

The first thing that the merge function does is calculate the size of the left and right sub-arrays that will be merged. Then, we create two vectors for each of the sides of the array using the sizes we just calculated. These vectors are just to temporarily hold the numbers in the two arrays. Next, we fill those two vectors with the left and right sides of the numArray. Three indexes are then created in order to keep track of the positions of the leftArray, rightArray, and numArray respectively. A while loop is used to merge the two sub-arrays into the original numArray. The loop runs until either leftArray or rightArray is completely processed. Inside the loop, the elements of leftArray and

rightArray are compared, and the smaller element is placed in the original numArray. Their index is incremented to move to the next element in the corresponding sub-array. The index k is also incremented to move to the next position in the numArray where the next element will be placed. In case there are any remaining elements, I implemented 2 while loops for both arrays in order to put the forgotten elements in the numArray. Finally, after the vector is completely sorted, it is printed for the user to see.