CLASS DESIGN

In Project 1 I implemented a class called Time\_Series. In this class I made the member functions Load, PRINT, ADD, UPDATE, mean, is\_monotonic, and best\_fit. These are the functions that are required to complete project 1. I also developed three other helper functions that were useful in some of the other functions.

The first helper function that I implemented is arr\_shift, this function takes in three parameters, int temp\_val, int year, and double data. I use this helper function in my ADD function to shift all the elements in the array over one array index to maintain chronological order of the years.

The two other functions were used to help in my LOAD, ADD, and UPDATE. The first function is called does\_arr\_need\_resized. Anytime that I am adding or removing anything from the data and year array I call this function. This function determines if I need to double or quarter the capacity of the data and year array. If resizing is needed I update the m\_arr\_size variable and call the second function arr\_resize.

In arr\_resize I first allocate two temporary dynamically allocated arrays of capacity m\_arr\_size, then I loop through my data and year array in order to populate the temporary arrays with the information stored in the data and year array’s. I then deallocated the original arrays by using delete, after calling delete I assign the data and year array to the memory address of the temporary arrays.

ALTERNATIVES AND JUSTIFICATION

In this project we had the choice of either storing invalid data in our array’s. I decided to discard any invalid data, to me this made more sense, and I feel that this way would’ve been easier to implement rather than storing the invalid data.

In LOAD when populating the data and year array I would check if the data equals -1, if it did, I would increment the year because you need to keep track of the years for the rest of the data. Then if the data wasn’t equal to -1, I would use my private member variable m\_count to add the data and year into the arrays then increment m\_count and the year. In the add function if the user attempts to add invalid data I print failure to the console and return.

The UPDATE function is a little bit more complex since you can update a years data with invalid data. I am not storing invalid so I have to check if the “new” data that being updated is -1 and if so remove that year and data from the array. To do so I have to shift all elements back one array index then decrement m\_count since there is one less element in the array.

If I decided to store the invalid data I would’ve had to check in the PRINT function to skip any invalid data. Also in the , is\_monotonic function I would’ve had to make sure that the invalid data doesn’t affect if the data is monotonic.

RUNTIME ANALYSIS

My UPDATE function has a worst case runtime of O(N) when the year being updated is not present in the year array. For example, if we attempt to update year xxxx, but xxxx is not in the array, which contains N items. We need to loop through the array and check each entry to see if it matches xxxx. Since xxxx is not in the array, the function must traverse all N items to confirm that xxxx isn’t stored in the year array. This complete traversal of N items results in a worst case runtime of O(N).

My is\_monotonic function has a best case runtime of O(1) when the data array is empty. This happens because the first operation of the function is a check to determine if any data is stored in the array. Specifically, it checks whether the member variable m\_count equals zero. If m\_count is zero, there are no elements in the array, so the function returns false. Since no loops or additional operations are executed this case has best case runtime of O(1).