**RATIONALE**

**For this project,I decided to use a BST<int>years to stores all the year that is read from the file and a map<long long,set<Records>>RecordMap.I also used STL vector as a temporary container to get data from the structure.**

BST is a sequential data structure which stores elements in a node and maintains elements in their order. If elements greater than node insert in right node or if less than node insert in left .We can achieve a logarithmic time complexity for insertion ,deletion, searching etc. We can traverse the tree in various order pre order, post, in order to order the keys. BST is more flexible than an array. However ,the disadvantage of BST is that we should always implement a balanced BST tree otherwise the cost of operations will degenerate into a linear search. The shape of the tree depends on the order of insertions, and it can be degenerated and may end up into a linked list. When inserting or searching for an element, the key of each visited node has to be compared with the key of the element to be inserted/found. Keys in the tree may be long and the run time may increase. **That is why I decided to store the years only instead of the sensor data for a year. The files are also not read in in order ,therefore the BST will not be balanced**.– Average depth of a node is O(log n); – Maximum depth of a node is O(n).**I will use BST year for searching** .**If user enters a year ,I will use the BST searching function to check if this year exist in the structures.**

Map is an associative container where we have a key and a value pair to insert. Each key is unique and cannot be changed. It can be inserted or deleted but not altered. This is perfect to avoid duplication of data. It allows fast access to the value using the key. A map is also implemented as red black tree (a self-balancing balanced BST),thus key are stored in order. Searching an element in std::map is 0(log N) and still O(log N) in worst case. This is better than unordered map which is O(1) or O(n) in worst cases. std::map also use less memory when storing big chunks of data. **Since most operations rely on finding data for a given month of a year, I decided to create a key using the month and year to uniquely identify the set of values for this given month of this year. If month is 3 and year is 2010 the key for the map will be( (3\*10000)+year)=32010.It is easy to implement and code using the std::map as there are STL algorithms already available.**

Set is similar to std::map as they are both use Red black tree with time complexities of O(log N) for searching, insertion and deletion. Set store keys only. You can access elements via find() with the STL algorithms. Set also prevent duplications of data .**As there are duplicates of data in some file for a given time of day of a month of a given years std::set is the perfect structure to handle this issue. I use the std::set to store the Records(wind speed, ambient temperature and solar radiation ,date, time) for a given month of year. For ordering the Records and avoid duplication I decided to attach a key to each record. I use the day, time and min to create the key for each record of a given day and time. I have inserted a new variable string key in my Record.h.Example for day=1 min=00 and hours=9,the key will be 9001(string-(“9”+”00”+”1”).String key might not be the most efficient way of ordering the records. I was having difficulties to find a proper formula using numeric data types to generate a key for each Record. I will need to overload the < operator of Record class.**

**Map<long long, set<>> is much easier to code for loading data into the structure with just one line compared to a map<long long, vector<>> where you will need to create an empty vector first load and deallocate memory which is painful. Therefore, I decided to use the vector store only get data from the structure so that the user can perform the require calculations. It is easier to load specific data for a given month of a year using pushback and use clear() to de allocate.**

**I have decided to create a Data Control class (Datacontrol.h).This will provide encapsulation/information hiding. The user will have controlled access of the structures using a set of public methods available. This class will have functions that will allow user to windspeed, solar radiation and temperature from Record map. I have also used a Menu. H to keep all the functions of the main program functions. This will make my main.cpp looks cleaner. As I will only create a pointer of type menu and point to the loop function that contains all the 5 required menu.**

**Moreover, I wanted to come up with an efficient way of storing my data for rapid searching and inserting data. I also wanted avoid structures that are too complex to implement or require lots of coding and given the limited time frame it will be difficult for me.**