DOCUMENTATION

REQUIREMENTS

The main requirement of the project is to find the **minimum and the maximum** from a given set of temperature values, involving $\frac{3}{2}$n – 2 comparisons. Other requirements include:

- Ensuring that all the data objects use the same unit for measuring temperature.
- The input data is stored in a random access buffer that does not differentiate between data from the child nodes and that from the sensors (whose count can be anywhere between 1 and ten of thousands!)
- Each datum is of a particular format viz. (Sensor ID, Location, Time Stamp, Temperature Value, Temperature Unit)

ASSUMPTIONS

- The sensors are assumed to be homogenous (i.e.) the format of the data provided by different sensors is assumed to be uniform.
- The random access buffer is not empty at any point of time.
- The standard unit for temperature is assumed to be Celsius. If the input is in Fahrenheit, it is converted to Celsius.

APPROACHING THE DESIGN ASPECT

- A class should be nested within the main class to house the attributes of the data received by the sensors (sensor id, temperature, etc.)
- The class should have different functions that are capable of converting Fahrenheit to Celsius, printing the data, and copying one object to another.
- If two or more elements have the same value for temperature (which happens to be the minimum or the maximum value), then the record that occurred earlier in the list will be returned.

- The given level of optimality can be achieved if divide and conquer strategy is adopted. Thus, the neuron component that returns the minimum and maximum entries from the pool of input objects has to be recursive.
- The recursive algorithm may have 2 base cases corresponding to n = 1 and n = 2. If n is 1, then the same value is considered as the min and max. If n is 2, it takes just 1 comparison to figure out the min and max among those 2 elements.
- Finally, the min and max obtained from the child node are compared at the parent node to find the absolute min and max at that node.
- Recursion takes place when n satisfies neither of the base cases. 'n' is split into two numbers, where at least one of them is a power of two.
- Optimisation is done by reducing the number of comparisons. In other words, the quicker a value is decomposed to one of the base cases, the quicker the program runs.
- The recursion ends when the value of n shrinks to a value equal to or lesser than 0.
- The number of comparisons can be estimated by having a counter that is incremented every time a comparison is made. This is done to verify the optimality of the code.

APPROACH TO VERSION CONTROL

A git version control system was used as prescribed in the question. Addition of new files and the changes made to the code were committed and pushed upstream in a periodic manner.

TESTING METHODS

Although no explicit testing tool was used for this assignment, the program's response to different inputs was observed. It is seen that the program works for odd number of elements, even number of elements, as well as for a single input. Some of the results are tabulated below:

S. no	Value for n	Unit of temperature	Missing data	Non unique temperature values	Result
1	10	Has C as well as F	No	No	Success
2	11	Has C as well as F	No	Yes	Success
3	0	-	-	-	NullPointerException
4	1	Has C or Has F	No	No	Success for C as well as F
5	10	-	Yes (unit is missing)	No	ArrayIndexOutOfBoundsException
6	10	Has C as well as F	Yes (Time is missing)	Yes	NumberFormatException
7	10	Has C as well as F	Yes(Temperature value is missing)	No	NumberFormatException
8	2	Has C as well as F	No	No	Success
9	2	Has C	No	Yes	Success
10	10	Has C as well as F	Has an empty record(all values are missing)	No	NumberFormatException