Implement a doubly linked list class. The class must use a template to allow for this container to be created for any data type. The implementation must be done with linked lists. A head and tail pointer should be maintained for efficiency. Creating a variable should look like `LinkedList<int> x;`

```cpp
template<class DATATYPE>
struct LLNode
{
   DATATYPE data;
   LLNode<DATATYPE> *next;
   LLNode<DATATYPE> *prev;
};

template<class DATATYPE>
class LinkedList
{
private:
    //use a doubly linked-list based implementation
    //keep a head and tail pointer for efficiency
    LLNode<DATATYPE> *Head;
    LLNode<DATATYPE> *Tail;
    int Count;
public:
    //setup initial conditions
    LinkedList();
    //delete all dynamic memory, etc.
    ~LinkedList();
    //Equality operator to check if two lists are equal
    bool operator==(const LinkedList<DATATYPE>&);
    //Inequality operator to check if two lists are equal
    bool operator!=(const LinkedList<DATATYPE>&);
    //add x to front of list
    void addFront(DATATYPE);
    //add x to back of list
    void addBack(DATATYPE);
    //add x as the i-th thing in the list
    //if there are less than i things, add it to the back
    void add(DATATYPE, int);
    //remove and return front item from list
    DATATYPE removeFront();
    //remove and return back item from list
    DATATYPE removeBack();
    //return value of back item (but don't remove it)
    DATATYPE getBack();
    //return value of front item (but don't remove it)
    DATATYPE getFront();
    //return how many items are in the list
    int length();
    //print all elements in the linked list
    void print();
};
```
**Bonus(2):** Overload the stream insertion operators `<<` to call the print function.

**Bonus(3):** Overload the operators `<`, `<=`, `>`, `>=` for comparison. This can be done based off of assumed ordering of the data.

**Bonus(3):** Create a 'void sort()' function that sorts the list. Assume the DATATYPEs can be compared with standard comparison operators.