```
package nachos.threads;  // don't change this. Gradescope needs it.

public class DLList
{
    private DLLElement first;  // pointer to first node
    private DLLElement last;   // pointer to last node
    private int size;          // number of nodes in list

    /**
     * Creates an empty sorted doubly-linked list.
     */
    public DLList() {
        first = null;
        last = null;
        size = 0;
    }

    /**
     * Add item to the head of the list, setting the key for the new
     * head element to min_key - 1, where min_key is the smallest key
     * in the list (which should be located in the first node).
     * If no nodes exist yet, the key will be 0.
     */
    public void prepend(Object item) {
        DLLElement newNode;
        if (isEmpty()) {
            newNode = new DLLElement(item, 0);
            last = newNode;
        } else {
            newNode = new DLLElement(item, first.key - 1);
            newNode.next = first;
            first.prev = newNode;
        }

        first = newNode;
        size += 1;
    }

    /**
     * Removes the head of the list and returns the data item stored
in
     * it.  Returns null if no nodes exist.
     *
     * @return the data stored at the head of the list or null if list
empty
     */
    public Object removeHead() {
        if (isEmpty()) {
            return null;
        } else {
            Object toReturn = first.data;
            first = first.next;
            size -= 1;

            if (!isEmpty()) {
                first.prev = null;
            } else {
                last = null;
            }
            return toReturn;
        }


    }

    /**
     * Tests whether the list is empty.
     *
     * @return true iff the list is empty.
     */
    public boolean isEmpty() {
        return first == null;
    }

    /**
     * returns number of items in list
     * @return
     */
    public int size() {
        return size;
    }


    /**
     * Inserts item into the list in sorted order according to
sortKey.
     */
    public void insert(Object item, Integer sortKey) {
        DLLElement newNode = new DLLElement(item, sortKey);
        if (isEmpty()) {
            last = newNode;
            first = newNode;

        } else if (first.key > sortKey) {
            first.prev = newNode;
            newNode.next = first;
            first = newNode;

        } else {
            if (sortKey >= last.key) {
                last.next = newNode;
                newNode.prev = last;
                last = newNode;
            } else {

                DLLElement currNode = first;
                DLLElement prevNode = first.prev;
                while(!(currNode == null) && currNode.key < sortKey) {
                    prevNode = currNode;
                    currNode = currNode.next;
                }

                prevNode.next = newNode;
                newNode.next = currNode;
                newNode.prev = prevNode;
                currNode.prev = newNode;
            }
        }

        size += 1;

    }


    /**
     * returns list as a printable string. A single space should
separate each list item,
     * and the entire list should be enclosed in parentheses. Empty
list should return "()"
     * @return list elements in order
     */
    public String toString() {
        if (isEmpty()) {
            return "()";
        } else {
            String toReturn = "(" + first.toString();
            DLLElement currNode = first.next;
            while(currNode != null) {
                toReturn += " " + currNode.toString();
                currNode = currNode.next;
            }
            toReturn += ")";
            return toReturn;
        }

    }

    /**
     * returns list as a printable string, from the last node to the
first.
```

```
     * String should be formatted just like in toString.
     * @return list elements in backwards order
     */
    public String reverseToString(){
        if (isEmpty()) {
            return "()";
        } else {
            String toReturn = "(" + last.toString();
            DLLElement currNode = last.prev;
            while(currNode != null) {
                toReturn += " " + currNode.toString();
                currNode = currNode.prev;
            }
            toReturn += ")";
            return toReturn;
        }
    }

    /**
     *  inner class for the node
     */
    private class DLLElement
    {
        private DLLElement next;
        private DLLElement prev;
        private int key;
        private Object data;

        /**
         * Node constructor
         * @param item data item to store
         * @param sortKey unique integer ID
         */
        public DLLElement(Object item, int sortKey)
        {
                key = sortKey;
                data = item;
                next = null;
                prev = null;
        }

        /**
         * returns node contents as a printable string
         * @return string of form [<key>,<data>] such as [3,"ham"]
         */
        public String toString(){
            return "[" + key + "," + data + "]";
        }
    }
}
```