import java.util.ArrayList;

public class Heap<T extends Comparable<T>> {

    private final int childCount;

    private final ArrayList<T> data;

    public Heap(int childCount) {

        this.validateChildCount(childCount);

        this.childCount = childCount;

        this.data = new ArrayList<T>();

    }

    private void validateChildCount(int childCount) {

        // ensure childCount is valid

        // ensure childCount is greater than zero

        if (childCount <= 0) {

            throw new IllegalArgumentException("childCount must be greater than zero");

        }

        // ensure childCount is a power of 2

        double logChildCount = Math.log(childCount) / Math.log(2);

        if (Math.ceil(logChildCount) != Math.floor(logChildCount)) {

            throw new IllegalArgumentException("childCount must be a power of 2");

        }

    }

    public void insert(T item) {

        // insert an item into the heap

        data.add(item);

        int itemIndex = data.size() - 1;

        while (itemIndex > 0) {

            itemIndex = this.swapUp(itemIndex);

        }

    }

    private int swapUp(int childIndex) {

        // check a child against its parent, and swap it if necessary to satisfy heap property

        T childValue = data.get(childIndex);

        int parentIndex = (int) Math.floor((float) (childIndex - 1) / childCount);

        if (parentIndex >= 0) {

            T parentValue = data.get(parentIndex);

            if (childValue.compareTo(parentValue) > 0) {

                data.set(parentIndex, childValue);

                data.set(childIndex, parentValue);

                return parentIndex;

            }

        }

        return -1;

    }

    public T popMax() {

        // pop the max value off the heap, return null if none remain

        if (data.size() > 0) {

            T maxItem = data.get(0);

            if (data.size() > 1) {

                T lastItem = data.remove(data.size() - 1);

                data.set(0, lastItem);

                int itemIndex = 0;

                while (itemIndex >= 0) {

                    itemIndex = this.swapDown(itemIndex);

                }

            }

            return maxItem;

        } else {

            return null;

        }

    }

    private int swapDown(int parentIndex) {

        // check a parent against all children and swap it with the highest child if necessary to satisfy heap property

        T parentValue = data.get(parentIndex);

        // determine largest child

        int largestChildIndex = 0;

        T largestChildValue = null;

        for (int i = 0; i < childCount; i++) {

            int childIndex = childCount \* parentIndex + i + 1;

            if (childIndex < data.size() - 1) {

                T childValue = data.get(childIndex);

                if (largestChildValue == null || childValue.compareTo(largestChildValue) > 0) {

                    largestChildIndex = childIndex;

                    largestChildValue = childValue;

                }

            }

        }

        // perform swap if necessary

        if (largestChildValue != null && parentValue.compareTo(largestChildValue) < 0) {

            data.set(parentIndex, largestChildValue);

            data.set(largestChildIndex, parentValue);

            return largestChildIndex;

        }

        return -1;

    }

}