```java
package edu.caltech.cs2.datastructures;

import edu.caltech.cs2.interfaces.IDictionary;
import edu.caltech.cs2.interfaces.IPriorityQueue;

import java.util.Iterator;

public class MinFourHeap<E> implements IPriorityQueue<E> {

    private static final int DEFAULT_CAPACITY = 10;

    private int size;
    private PQElement<E>[] data;
    private IDictionary<E, Integer> keyToIndexMap;

    /**
     * Creates a new empty heap with DEFAULT_CAPACITY.
     */
    public MinFourHeap() {
        this.size = 0;
        this.data = new PQElement[DEFAULT_CAPACITY];
        this.keyToIndexMap = new ChainingHashDictionary<>(MoveToFrontDictionary::new);
    }

    @Override
    public void increaseKey(PQElement<E> key) {
        Integer index = this.keyToIndexMap.get(key.data);
        PQElement<E> spot = new PQElement<>(this.data[index].data, key.priority);
        this.data[index] = spot;
        percolateDown(index);
    }

    @Override
    public void decreaseKey(PQElement<E> key) {
        Integer index = this.keyToIndexMap.get(key.data);
        PQElement<E> spot = new PQElement<>(this.data[index].data, key.priority);
        this.data[index] = spot;
        percolateUp(index);
    }

    @Override
    public boolean enqueue(PQElement<E> epqElement) {
        for (PQElement<E> i : this.data){
            if (i == epqElement){
                throw new IllegalArgumentException();
            }
        }
        if (this.size == 0){
            data[0] = epqElement;
        }
        else {
            data[this.size - 1] = epqElement;
            percolateUp(this.size - 1);
        }
        this.keyToIndexMap.put(epqElement.data, (int) epqElement.priority);
        this.size++;
        return true;
    }

    private void percolateUp(int index){
        if (index <= 0){
            return;
        }
        if (data[index-1] != null){
            if (data[index].priority < data[index-1].priority){
                PQElement<E> current = data[index];
                PQElement<E> parent = data[index-1];
                this.data[index] = parent;
                this.data[index-1] = current;
            }
            percolateUp(index-1);
        }
    }

    @Override
```

```
    public PQElement<E> dequeue() {
        PQElement<E> returnval = data[0];
        data[0] = null;
        percolateDown(0);
        this.keyToIndexMap.remove(returnval.data);
        this.size--;
        return returnval;
    }

    private void percolateDown(int index){
        if (index >= this.size-1){
            return;
        }
        if (this.data[index] != null && this.data[index+1] != null){
            if (data[index].priority > data[index+1].priority){
                PQElement<E> current = data[index];
                PQElement<E> child = data[index+1];
                this.data[index] = child;
                this.data[index+1] = current;
            }
            percolateDown(index+1);
        }
    }

    @Override
    public PQElement<E> peek() {
        if (this.size == 0){
            return null;
        }
        return this.data[0];
    }

    @Override
    public int size() {
        return this.size;
    }

    @Override
    public Iterator<PQElement<E>> iterator() {
        return null;
    }
}
```