File - /Users/moe/Downloads/cs61b/fa20-s1319/proj2ab/bearmaps/ArrayHeapMinPQ.java

```java
package bearmaps;
import org.w3c.dom.Node;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.NoSuchElementException;

public class ArrayHeapMinPQ<T> implements ExtrinsicMinPQ<T> {
    private ArrayList<Node> pQ;
    private HashMap<T, Integer> inputs;

    private class Node {
        private T heapItem;
        private double prioritized;

        Node(T theItem, double thePrioritized) {
            heapItem = theItem;
            prioritized = thePrioritized;
        }

    }

    public ArrayHeapMinPQ() {
        pQ = new ArrayList<>();
        inputs = new HashMap<>();
    }

    private int parent(int i) {
        return (i - 1) / 2;
    }

    private int leftChild(int i) {
        return i * 2 + 1;
    }

    private int rightChild(int i) {
        return i * 2 + 2;
    }

    private void swap(int x, int z) {
        Node original = pQ.get(x);
        pQ.set(x, pQ.get(z));
        pQ.set(z, original);
        inputs.put(pQ.get(x).heapItem, x);
        inputs.put(pQ.get(z).heapItem, z);
    }

    private void swim(int k) {
        while (k > 0) {
            int theParent = parent(k);
            if (!(pQ.get(k).prioritized < pQ.get(theParent).prioritized)) {
                return;
            }
```

```
54              swap(k, theParent);
55              k = theParent;
56          }
57          /*if (parent(k) > k) {
58              swap(k, parent(k));
59              swim(parent(k));
60          }*/
61          /*while (k > 0)
62          int parentNum = parent(k);
63          swap(k, );
64          k = parentNum;*/
65      }
66
67      private boolean trueorFalse(int x, int y) {
68          return pQ.get(x).prioritized < pQ.get(y).prioritized;
69      }
70      private void sink(int k) {
71          while (leftChild(k) < size()) {
72              int i = leftChild(k);
73              if (rightChild(k) < size() && trueorFalse(rightChild(k), i)) {
74                  i = rightChild(k);
75              }
76              if (pQ.get(k).prioritized < pQ.get(i).prioritized) {
77                  return;
78              }
79              swap(k, i);
80              k = i;
81          }
82      }
83
84      /* Adds an item with the given priority value. Throws an
85       * IllegalArgumentExceptionb if item is already present.
86       * You may assume that item is never null. */
87      public void add(T item, double priority) {
88          if (contains(item)) {
89              throw new IllegalArgumentException();
90          }
91          pQ.add(new Node(item, priority));
92          inputs.put(item, size() - 1);
93          swim(size() - 1);
94
95      }
96
97      /* Returns true if the PQ contains the given item. */
98      public boolean contains(T item) {
99          return inputs.containsKey(item);
100     }
101
102     /* Returns the minimum item. Throws NoSuchElementException if the PQ is empty. */
103     public T getSmallest() {
104         if (pQ.isEmpty()) {
105             throw new NoSuchElementException();
106         }
```

```
107         return pQ.get(0).heapItem;
108     }
109
110     /* Removes and returns the minimum item. Throws NoSuchElementException if the PQ is empty. */
111     public T removeSmallest() {
112         if (pQ.isEmpty()) {
113             throw new NoSuchElementException();
114         }
115         int last = size() - 1;
116         T topNode = getSmallest();
117         swap(0, last);
118         pQ.remove(last);
119         sink(0);
120         inputs.remove(topNode);
121         return topNode;
122     }
123
124     /* Returns the number of items in the PQ. */
125     public int size() {
126         return pQ.size();
127     }
128
129     /* Changes the priority of the given item. Throws NoSuchElementException if the item
130      * doesn't exist. */
131     public void changePriority(T item, double priority) {
132         if (!contains(item)) {
133             throw new NoSuchElementException();
134         }
135         int theItem = inputs.get(item);
136         double prevPrioritized = pQ.get(theItem).prioritized;
137         pQ.get(theItem).prioritized = priority;
138         if (prevPrioritized < priority) {
139             sink(theItem);
140         }
141         if (prevPrioritized >= priority) {
142             swim(theItem);
143         }
144     }
145 }
146
```