```java
package Assign_3;

/** This interface is an implementation of PQ or Priority Queue for this Priority Queue Manipulation program.
 * Micah Rose-Mighty
 * 6498935
 * 2020-11-13
 * Created using IntelliJ
 */


public interface PQ<T extends Comparable<T>> {

    public void transverse();

    public T deleteMin ();

    public void insert(T x);
}

```

```java
package Assign_3;

/** This class is an implementation of the main class
 for this Priority Queue Manipulation program
although the program is run from each of the 3
different priority queue implementation, this class
was created by default by IntelliJ.
 * Micah Rose-Mighty
 * 6498935
 * 2020-11-13
 * Created using IntelliJ
 */

public class Main {

    public static void main(String[] args) {
    }
}

```

```java
package Assign_3;

/** This class is an implementation of the Analysis class for this Priority Queue Manipulation program.
 * Micah Rose-Mighty
 * 6498935
 * 2020-11-13
 * Created using IntelliJ
 */

import java.io.PrintWriter;
import java.time.Duration;
import java.time.Instant;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

public class Analysis {

    public static void printTime(PQ pq, PrintWriter out){ // Method to Print the time analysis of the different types of Priority Queues to the Output File
        Random random = new Random();

        List<Integer> list50 = random.ints(50, 1, 1000).boxed().collect(Collectors.toList());

        Instant start50 = Instant.now();
        list50.stream().forEach(i-> pq.insert(i));
        list50.stream().forEach(i->pq.deleteMin());
        Instant end50 = Instant.now();

        List<Integer> list100 = random.ints(100, 1, 1000).boxed().collect(Collectors.toList());

        Instant start100 = Instant.now();
        list100.stream().forEach(i-> pq.insert(i));
        list100.stream().forEach(i->pq.deleteMin());
        Instant end100 = Instant.now();

        List<Integer> list1000 = random.ints(1000, 1 , 1000).boxed().collect(Collectors.toList());

        Instant start1000 = Instant.now();
```

```
39          list1000.stream().forEach(i-> pq.insert(i));
40          list1000.stream().forEach(i->pq.deleteMin());
41          Instant end1000 = Instant.now();
42
43          List<Integer> list5000 = random.ints(5000, 1
    , 1000).boxed().collect(Collectors.toList());
44
45          Instant start5000 = Instant.now();
46          list5000.stream().forEach(i-> pq.insert(i));
47          list5000.stream().forEach(i->pq.deleteMin());
48          Instant end5000 = Instant.now();
49
50          List<Integer> list10000 = random.ints(10000,
    1, 1000).boxed().collect(Collectors.toList());
51
52          Instant start10000 = Instant.now();
53          list10000.stream().forEach(i-> pq.insert(i));
54          list10000.stream().forEach(i->pq.deleteMin
    ());
55          Instant end10000 = Instant.now();
56
57          out.println("Analysing time of operation");
58          out.printf("%-6s %-15s%n", "n", "Duration");
59          out.printf("----------------------%n");
60          out.printf("%-6s %-15d ns %n", "50", Duration
    .between(start50, end50).toNanos());
61          out.printf("%-6s %-15d ns %n", "100",
    Duration.between(start100, end100).toNanos());
62          out.printf("%-6s %-15d ns %n", "1000",
    Duration.between(start1000, end1000).toNanos());
63          out.printf("%-6s %-15d ns %n", "5000",
    Duration.between(start5000, end5000).toNanos());
64          out.printf("%-6s %-15d ns %n", "10000",
    Duration.between(start10000, end10000).toNanos());
65
66
67      }
68
69      public static void printArrayTime(ArrayHeapPQ pq
    , PrintWriter out){//Method to Print the time
    analysis of the Array Heap type of Priority Queue to
    the Output File
70          Random random = new Random();
71          //50 100 1000 5000 10000
```

```
 72         List<Integer> list50 = random.ints(50, 1,
    1000).boxed().collect(Collectors.toList());
 73
 74         Instant start50 = Instant.now();
 75         list50.stream().forEach(i-> pq.normalInsert(
    i));
 76         pq.buildHeap();
 77         list50.stream().forEach(i->pq.deleteMin());
 78         Instant end50 = Instant.now();
 79
 80         List<Integer> list100 = random.ints(100, 1,
    1000).boxed().collect(Collectors.toList());
 81
 82         Instant start100 = Instant.now();
 83         list100.stream().forEach(i-> pq.normalInsert
    (i));
 84         pq.buildHeap();
 85         list100.stream().forEach(i->pq.deleteMin());
 86         Instant end100 = Instant.now();
 87
 88         List<Integer> list1000 = random.ints(1000, 1
    , 1000).boxed().collect(Collectors.toList());
 89
 90         Instant start1000 = Instant.now();
 91         list1000.stream().forEach(i-> pq.
    normalInsert(i));
 92         pq.buildHeap();
 93         list1000.stream().forEach(i->pq.deleteMin
    ());
 94         Instant end1000 = Instant.now();
 95
 96         List<Integer> list5000 = random.ints(5000, 1
    , 1000).boxed().collect(Collectors.toList());
 97
 98         Instant start5000 = Instant.now();
 99         list5000.stream().forEach(i-> pq.
    normalInsert(i));
100         pq.buildHeap();
101         list5000.stream().forEach(i->pq.deleteMin
    ());
102         Instant end5000 = Instant.now();
103
104         List<Integer> list10000 = random.ints(10000
    , 1, 1000).boxed().collect(Collectors.toList());
```

```
105
106         Instant start10000 = Instant.now();
107         list10000.stream().forEach(i-> pq.
    normalInsert(i));
108         pq.buildHeap();
109         list10000.stream().forEach(i->pq.deleteMin
    ());
110         Instant end10000 = Instant.now();
111
112         out.println("Analysing time of operation");
113         out.printf("%-6s %-15s%n", "n", "Duration");
114         out.printf("----------------------%n");
115         out.printf("%-6s %-15d ns %n", "50",
    Duration.between(start50, end50).toNanos());
116         out.printf("%-6s %-15d ns %n", "100",
    Duration.between(start100, end100).toNanos());
117         out.printf("%-6s %-15d ns %n", "1000",
    Duration.between(start1000, end1000).toNanos());
118         out.printf("%-6s %-15d ns %n", "5000",
    Duration.between(start5000, end5000).toNanos());
119         out.printf("%-6s %-15d ns %n", "10000",
    Duration.between(start10000, end10000).toNanos());
120
121
122     }
123 }
124
125
```

```java
package Assign_3;


/** This class is an implementation of the Array Heap
 Priority Queue class for this Priority Queue
Manipulation program.
 * Micah Rose-Mighty
 * 6498935
 * 2020-11-13
 * Created using IntelliJ
 */


import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class ArrayHeapPQ<T extends Comparable<T>>
implements PQ<T>{

    T[] pq;
    int N ;
    int size = 0;
    PrintWriter out;

    ArrayHeapPQ(){
        this(10000);
    }
    ArrayHeapPQ(int capacity){
        pq = (T[]) new Comparable[capacity+1];
        try {
            out = new PrintWriter("ArrayHeap.txt");
        } catch (FileNotFoundException
fileNotFoundException) {
            fileNotFoundException.printStackTrace();
        }
    }

    public boolean isEmpty(){
        return N == 0;
    }

```

```
41
42
43
44      @Override
45      public void transverse() {//Traversal Method for
    the Array Heap implementation of the priority queue
46          StringBuilder builder = new StringBuilder();
47          preorder(builder,1);
48          String s = Arrays.stream(builder.toString().
    trim().split(" ")).collect(Collectors.joining(",",
    "[","]"));
49          out.println(s);
50      }
51
52      private void preorder(StringBuilder builder, int
    root){// Preorder Traversal Method for the Ordered
    Array Heap implementation of the priority queue
53          if(root>=0 && root <= N){
54              builder.append(pq[root]+" ");
55              preorder(builder,2*root);
56              preorder(builder,2*root+1);
57          }
58
59      }
60
61      @Override
62      public T deleteMin() {//deleteMin Method for the
    Array Heap implementation of the priority queue
63          T min = pq[1];
64          T hold = pq[N];
65          pq[N] = pq[1];
66          pq[1] = hold;
67          pq[N--] = null;
68          size--;
69          sink(1);
70          return min;
71      }
72
73      @Override
74      public void insert(T x) {//Insertion Method for
    the Array Heap implementation of the priority queue
75          pq[++N] = x;
76          size++;
77          buildHeap();
```

```
 78     }
 79
 80     public void normalInsert(T x) {//Insertion
    Method for the Array Heap implementation of the
    priority queue that does not build the heap
 81         pq[++N] = x;
 82         size++;
 83     }
 84
 85     private void swim(int k){
 86         while (k > 1 && less(k, k/2)){
 87             T hold = pq[k];
 88             pq[k] = pq[k/2];
 89             pq[k/2] = hold;
 90             k = k/2;
 91         }
 92     }
 93
 94     private void minHeapify(int i){ // Method to
    create the minheap from the given data input using
    array heap implementation
 95         int left = 2*i;
 96         int right = 2*i+1;
 97         int m;
 98         if (left <= N && less(left, i))
 99             m = left;
100         else m = i;
101         if (right <=N && less(right, m))
102             m = right;
103         if (m!=i){
104             T hold = pq[i];
105             pq[i] = pq[m];
106             pq[m] = hold;
107             minHeapify(m);
108         }
109
110     }
111
112     public void buildHeap(){//Method to build the
    heap using the minHeapify method
113         for (int i=N/2; i>=1; i--){
114             minHeapify(i);
115         }
116     }
```

```
117
118     private boolean less(int key1Pos, int key2Pos){
    // method for comparing the keys of different
    elements in the array heap
119         return pq[key1Pos].compareTo(pq[key2Pos]) <
    0;
120     }
121
122     private void sink(int k){ // method for doing
    swaps within the array heap to keep the minheap
    characteristics true
123         while(2*k <= N){
124             int j = 2*k;
125             if (j< N && less(j+1, j)) j++;
126             if (!less(j, k)) break;
127             T hold = pq[k];
128             pq[k] = pq[j];
129             pq[j] = hold;
130             k =j;
131         }
132     }
133
134
135     public void close(){
136         out.close();
137     }
138
139     public static void main(String[] args){// main
    method for running the ArrayHeap Priority Queue
    implementation
140         ArrayHeapPQ<Integer> pq = new ArrayHeapPQ
    <>();
141         List<Integer> priorityList = MyFileReader.
    priorityList();
142         pq.out.println("-------------------- i
     --------------------");
143         for (Integer i: priorityList){
144             pq.insert(i);
145
146         }
147         pq.out.print("Transversing using PreOrder");
148         pq.transverse();
149         for (Integer i: priorityList){
150             pq.out.println(pq.deleteMin());
```

```
151         }
152         pq.out.println("-------------------- ii
     --------------------");
153         for (Integer i: priorityList){
154             pq.normalInsert(i);
155         }
156         pq.out.print("Transversing using PreOrder
    Before Build Heap ");
157         pq.transverse();
158         pq.buildHeap();
159         pq.out.print("Transversing using PreOrder
    After Build Heap ");
160         pq.transverse();
161         for (Integer i: priorityList){
162             pq.out.println(pq.deleteMin());
163         }
164         Analysis.printTime(pq, pq.out);
165         pq.out.println("\n--------Analysing time of
    Operation of Array Heap when all elements are
    inserted before buildHeap is called------");
166         Analysis.printArrayTime(pq, pq.out);
167         pq.close();
168
169
170
171
172
173     }
174
175 }
176
```

```java
package Assign_3;

/** This class is an implementation of the Binary Tree Heap class for this Priority Queue Manipulation program.
 * Micah Rose-Mighty
 * 6498935
 * 2020-11-13
 * Created using IntelliJ
 */


import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class BinaryHeapPQ<T extends Comparable<T>> implements PQ<T>{

    PrintWriter out;

    BinaryHeapPQ(){
        try{
            out = new PrintWriter("BinaryHeapPQ.txt");
        } catch (FileNotFoundException fileNotFoundException){
            fileNotFoundException.printStackTrace();
        }
    }

    @Override
    public void transverse(){//Traversal Method for the Binary Heap implementation of the priority queue
        StringBuilder builder = new StringBuilder();
        preorder(builder,root);
        String s = Arrays.stream(builder.toString().trim().split(" ")).collect(Collectors.joining(",", "[", "]"));
        out.println(s);
    }

```

```
37      @Override
38      public T deleteMin() {//deleteMin Method for the
    Binary Heap implementation of the priority queue
39          T minVal = minValue(root);
40          deleteKey(minVal);
41          return minVal;
42      }
43
44
45         static class Node<T extends  Comparable<T>> {
46          T key;
47          Node left, right;
48
49
50          public Node(T data){// Node class for this
    Binary Tree Implentation of the Priority Queue
51              key = data;
52              left = right = null;
53          }
54         }
55
56         Node root;
57
58
59         void deleteKey(T key) {// method for deleting
    a node of a certain key within the binary heap
60              root = delete_Recursive(root, key);
61         }
62
63         Node delete_Recursive(Node root, T key) {//
    another method for deleting a node of a certain key
    within the binary tree
64              if (root == null) return root;
65
66              if(key.compareTo((T) root.key) < 0)
67                  root.left = delete_Recursive(root.left
    , key);
68              else if (key.compareTo((T) root.key) > 0)
69                  root.right = delete_Recursive(root.
    right, key);
70              else {
71                  if (root.left == null)
72                      return root.right;
73                  else if (root.right == null)
```

```
74                      return root.left;
75
76                  root.key = minValue(root.right);
77                  root.right = delete_Recursive(root.
   right, (T) root.key);
78          }
79          return root;
80      }
81
82      T minValue(Node root) {// method for getting the
    smallest value of the heap
83              T minval = (T) root.key;
84
85              while (root.left != null) {
86                  minval = (T) root.left.key;
87                  root = root.left;
88              }
89              return minval;
90      }
91
92      @Override
93      public void insert(T key) {// method for
   inserting a node into the binary heap based on key
94              root = insert_Recursive(root, key);
95      }
96
97      Node insert_Recursive(Node root, T key) {//
   recursive insertion method
98              if (root == null){
99                  root = new Node(key);
100                 return root;
101             }
102             if (key.compareTo((T) root.key) < 0)
103                 root.left = insert_Recursive(root.
   left, key);
104             else if (key.compareTo((T) root.key) >= 0
   )
105                 root.right = insert_Recursive(root.
   right, key);
106             return root;
107
108     }
109
110     private void preorder(){
```

```
111             StringBuilder builder = new StringBuilder
    ();
112             preorder(builder,root);
113     }
114 
115     private void preorder(StringBuilder builder,
    Node root) { // method for preorder traversal of the
     binary heap implementation
116             if (root != null) {
117                 builder.append(root.key+" ");
118                 preorder(builder,root.left);
119                 preorder(builder, root.right);
120             }
121     }
122 
123     private void close(){
124             out.close();
125     }
126 
127     public static void main(String[] args){// main
    method for running this binary heap priority queue
    implementation.
128     BinaryHeapPQ<Integer> pq = new BinaryHeapPQ<>();
129         List<Integer> priorityList = MyFileReader.
    priorityList();
130         for (Integer i: priorityList){
131             pq.insert(i);
132         }
133         pq.transverse();
134         for (Integer i: priorityList) {
135             pq.out.println(pq.deleteMin());
136         }
137 
138         Analysis.printTime(pq, pq.out);
139         pq.close();
140 
141     }
142 
143 
144 
145 }
146 
147 
```

```java
package Assign_3;


/** This class is an implementation of the File Reader class for this Priority Queue Manipulation program.
 * Micah Rose-Mighty
 * 6498935
 * 2020-11-13
 * Created using IntelliJ
 */

import java.io.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class MyFileReader {

    public static List<Integer> priorityList (){
        List<Integer> list = new ArrayList<Integer>();
        try (BufferedReader reader = new BufferedReader(new FileReader("assn3in.txt"))) {
            int length = Integer.parseInt(reader.readLine());
            list = Arrays.stream(reader.readLine().split("\\s+")).map(Integer::parseInt).collect(Collectors.toList());
        } catch (FileNotFoundException fileNotFoundException) {
            fileNotFoundException.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return list;
    }

    public static void write(String fileName, String value){
        try {
            PrintWriter writer  = new PrintWriter(fileName);
```

```
35              writer.write(value);
36          } catch (FileNotFoundException
   fileNotFoundException) {
37              fileNotFoundException.printStackTrace();
38          }
39      }
40
41
42  }
```

```java
package Assign_3;

/** This class is an implementation of the Ordered Linked List class for this Priority Queue Manipulation program.
 * Micah Rose-Mighty
 * 6498935
 * 2020-11-13
 * Created using IntelliJ
 */

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.List;

public class OrderedArrayPQ<T extends Comparable<T>> implements PQ<T>{
    T[] pq;
    int N;
    int size;

    PrintWriter out;

    public OrderedArrayPQ(){
        this(10000);
    }

    public OrderedArrayPQ(int capacity){
        pq = (T[]) new Comparable[capacity+1];
        try {
            out = new PrintWriter("OrderedArray.txt");
        } catch (FileNotFoundException fileNotFoundException) {
            fileNotFoundException.printStackTrace();
        }
    }


    @Override
    public void transverse() { //Traversal Method for the Ordered Linked List implementation of the priority queue
        out.print("[");
```

```
38        for(int i=0; i<N; i++){
39            if(i<N-1)
40                out.printf("%d,", pq[i]);
41            else
42                out.printf("%d", pq[i]);
43        }
44        out.println("]");
45      }
46
47      @Override
48      public T deleteMin() { //deleteMin Method for the
     Ordered Linked List implementation of the priority
   queue
49       T min = pq[N-1];
50       pq[N--] = null;
51       return min;
52      }
53
54      @Override
55      public void insert(T key) { //Insertion Method
   for the Ordered Linked List implementation of the
   priority queue
56          pq[N] = key;
57            int i = N-1;
58            while (i>=0 && pq[i].compareTo(key)<0){
59                pq[i+1] = pq[i];
60                i = i-1;
61            }
62            pq[i+1]=key;
63            N++;
64      }
65
66      public void close(){ //close method for the
   output txt file
67          out.close();
68      }
69
70      public static void main(String[] args){ //Main
   method for the Ordered Linked List implementation of
   the priority queue
71          OrderedArrayPQ<Integer> pq = new
   OrderedArrayPQ<>();
72          List<Integer> priorityList = MyFileReader.
   priorityList();
```

```
73          for (Integer i: priorityList){
74              pq.insert(i);
75          }
76          pq.transverse();
77          for (Integer i: priorityList){
78              pq.out.println(pq.deleteMin());
79          }
80
81          Analysis.printTime(pq, pq.out);
82          pq.close();
83
84      }
85
86  }
87
```