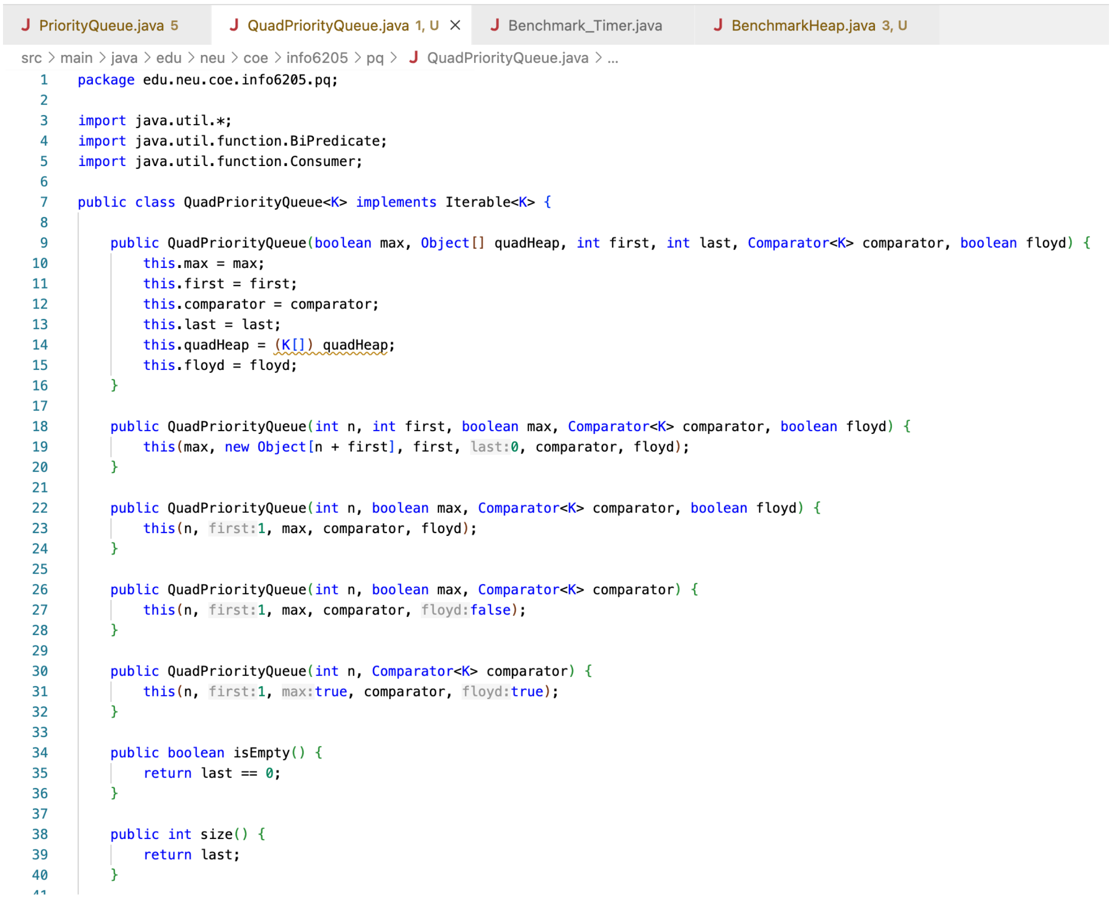
Program Structures and Algorithms

Fall 2024

NAME: Xinyi Xu

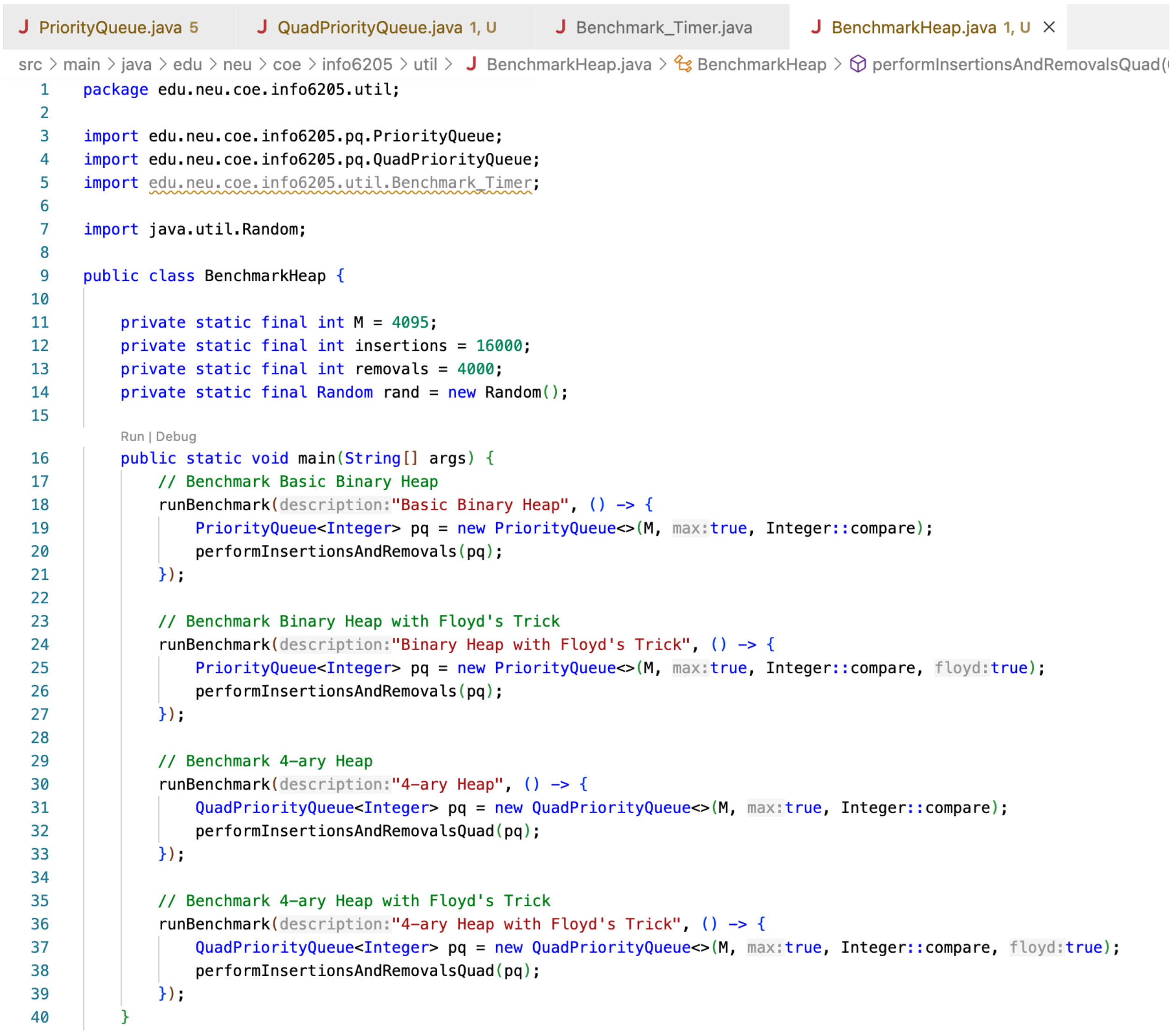
NUID: 002856992

Code Screenshots:



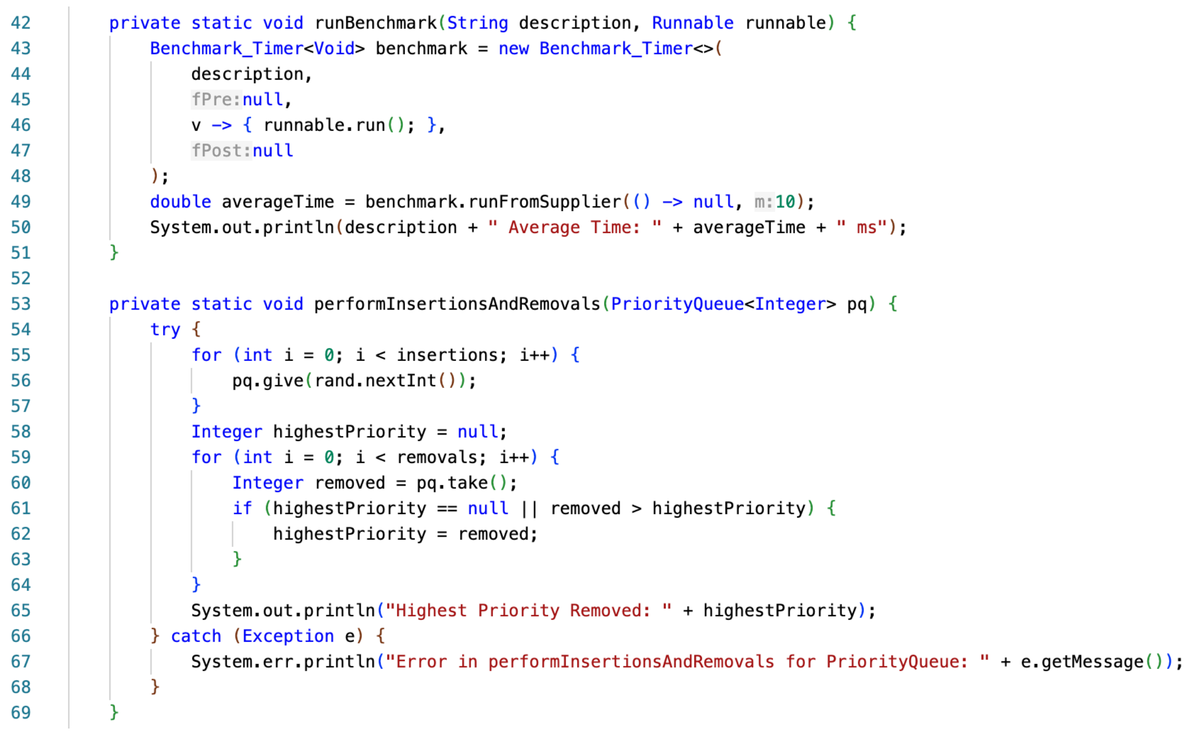
PriorityQueue.java 5
QuadPriorityQueue.java 1, U
Benchmark_Timer.java
BenchmarkHeap.java 3, U
src > main > java > edu > neu > coe > info6205 > pq > QuadPriorityQueue.java > ...

```java
package edu.neu.coe.info6205.pq;

import java.util.*;
import java.util.function.BiPredicate;
import java.util.function.Consumer;

public class QuadPriorityQueue<K> implements Iterable<K> {

    public QuadPriorityQueue(boolean max, Object[] quadHeap, int first, int last, Comparator<K> comparator, boolean floyd) {
        this.max = max;
        this.first = first;
        this.comparator = comparator;
        this.last = last;
        this.quadHeap = (K[]) quadHeap;
        this.floyd = floyd;
    }

    public QuadPriorityQueue(int n, int first, boolean max, Comparator<K> comparator, boolean floyd) {
        this(max, new Object[n + first], first, last:0, comparator, floyd);
    }

    public QuadPriorityQueue(int n, boolean max, Comparator<K> comparator, boolean floyd) {
        this(n, first:1, max, comparator, floyd);
    }

    public QuadPriorityQueue(int n, boolean max, Comparator<K> comparator) {
        this(n, first:1, max, comparator, floyd:false);
    }

    public QuadPriorityQueue(int n, Comparator<K> comparator) {
        this(n, first:1, max:true, comparator, floyd:true);
    }

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

    public int size() {
        return last;
    }
    public void give(K key) {
        if (last == quadHeap.length - first)
            last--;
        quadHeap[++last + first - 1] = key;
        swimUp(last + first - 1);
    }

    public K take() throws PQException {
        if (isEmpty()) throw new PQException(msg:"4-ary Heap is empty");
        if (floyd) return doTake(this::snake);
        else return doTake(this::sink);
    }

    K doTake(Consumer<Integer> f) {
        K result = quadHeap[first];
        swap(first, last-- + first - 1);
        f.accept(first);
        quadHeap[last + first] = null;
        return result;
    }

    void sink(int k) {
        doHeapify(k, (a, b) ->!unordered(a, b));
    }

    private int doHeapify(int k, BiPredicate<Integer, Integer> p) {
        int i = k;
        while (firstChild(i) <= last + first - 1) {
            int j = firstChild(i);
            for (int c = 1; c < 4; c++) {
                if (j + c < last + first - 1 && unordered(j, j + c)) j++;
            }
            if (!unordered(k, j)) break;
            swap(i, j);
            i = j;
        }
        return i;
    }
    void snake(int k) {
        swimUp(doHeapify(k, (a, b) ->!unordered(a, b)));
    }

    void swimUp(int k) {
        int i = k;
        while (i > first && unordered(parent(i), i)) {
            swap(i, parent(i));
            i = parent(i);
        }
    }

    private void swap(int i, int j) {
        K tmp = quadHeap[i];
        quadHeap[i] = quadHeap[j];
        quadHeap[j] = tmp;
    }

    boolean unordered(int i, int j) {
        if (quadHeap[i] == null || quadHeap[j] == null) {
            return false;
        }
        return (comparator.compare(quadHeap[i], quadHeap[j]) > 0) ^ max;
    }

    private int parent(int k) {
        return (k + 1 - first) / 4 + first - 1;
    }

    private int firstChild(int k) {
        return (k + 1 - first) * 4 + first - 1;
    }

    private final boolean max;
    private final int first;
    private final Comparator<K> comparator;
    private final K[] quadHeap;
    private int last;
    private final boolean floyd;

    public Iterator<K> iterator() {
        Collection<K> copy = new ArrayList<>(Arrays.asList(Arrays.copyOf(quadHeap, last + first)));
        Iterator<K> result = copy.iterator();
        if (first > 0) result.next();
        return result;
    }
}
```

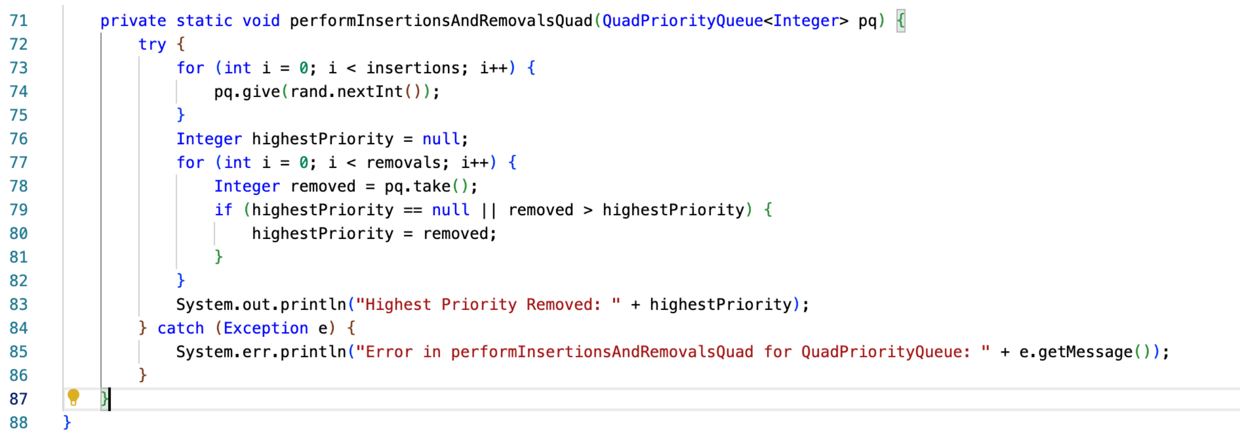


PriorityQueue.java 5
QuadPriorityQueue.java 1, U
Benchmark_Timer.java
BenchmarkHeap.java 1, U
src > main > java > edu > neu > coe > info6205 > util > BenchmarkHeap.java > BenchmarkHeap > performInsertionsAndRemovalsQuad(
Run | Debug

```java
package edu.neu.coe.info6205.util;

import edu.neu.coe.info6205.pq.PriorityQueue;
import edu.neu.coe.info6205.pq.QuadPriorityQueue;
import edu.neu.coe.info6205.util.Benchmark_Timer;

import java.util.Random;

public class BenchmarkHeap {

    private static final int M = 4095;
    private static final int insertions = 16000;
    private static final int removals = 4000;
    private static final Random rand = new Random();

    public static void main(String[] args) {
        // Benchmark Basic Binary Heap
        runBenchmark(description:"Basic Binary Heap", () -> {
            PriorityQueue<Integer> pq = new PriorityQueue<>(M, max:true, Integer::compare);
            performInsertionsAndRemovals(pq);
        });

        // Benchmark Binary Heap with Floyd's Trick
        runBenchmark(description:"Binary Heap with Floyd's Trick", () -> {
            PriorityQueue<Integer> pq = new PriorityQueue<>(M, max:true, Integer::compare, floyd:true);
            performInsertionsAndRemovals(pq);
        });

        // Benchmark 4-ary Heap
        runBenchmark(description:"4-ary Heap", () -> {
            QuadPriorityQueue<Integer> pq = new QuadPriorityQueue<>(M, max:true, Integer::compare);
            performInsertionsAndRemovalsQuad(pq);
        });

        // Benchmark 4-ary Heap with Floyd's Trick
        runBenchmark(description:"4-ary Heap with Floyd's Trick", () -> {
            QuadPriorityQueue<Integer> pq = new QuadPriorityQueue<>(M, max:true, Integer::compare, floyd:true);
            performInsertionsAndRemovalsQuad(pq);
        });
    }
```

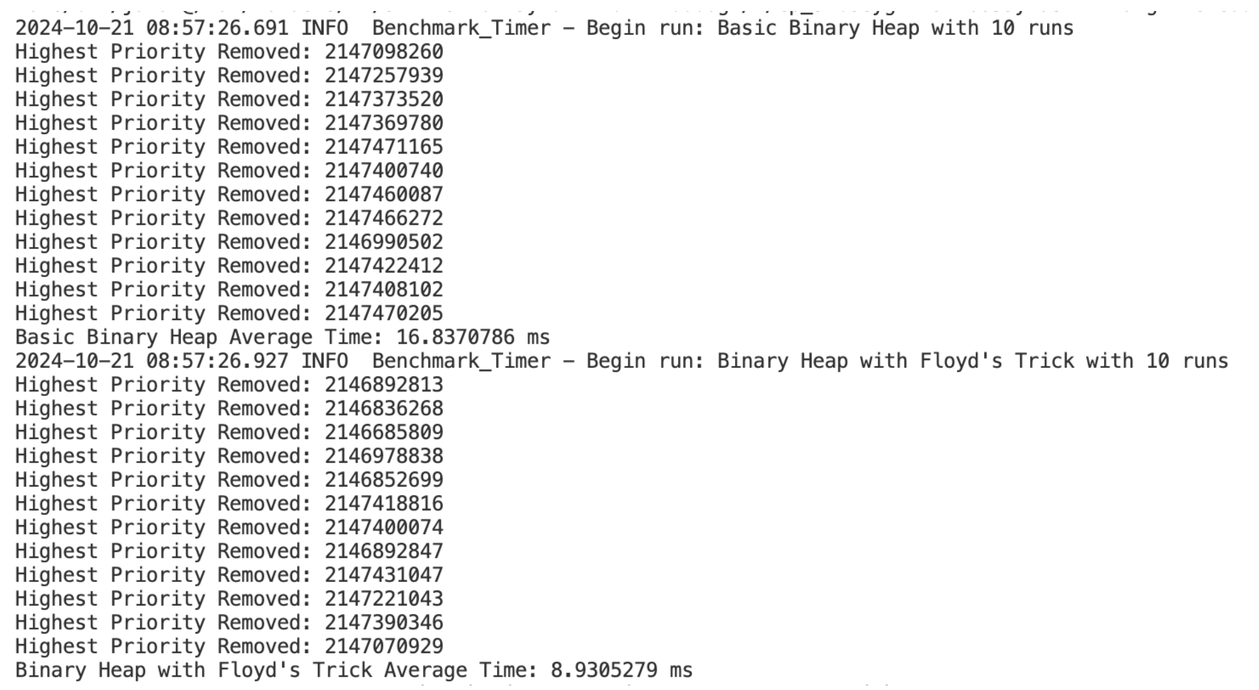


```java
    private static void runBenchmark(String description, Runnable runnable) {
        Benchmark_Timer<Void> benchmark = new Benchmark_Timer<>(
            description,
            fPre:null,
            v -> { runnable.run(); },
            fPost:null
        );
        double averageTime = benchmark.runFromSupplier(() -> null, m:10);
        System.out.println(description + " Average Time: " + averageTime + " ms");
    }

    private static void performInsertionsAndRemovals(PriorityQueue<Integer> pq) {
        try {
            for (int i = 0; i < insertions; i++) {
                pq.give(rand.nextInt());
            }
            Integer highestPriority = null;
            for (int i = 0; i < removals; i++) {
                Integer removed = pq.take();
                if (highestPriority == null || removed > highestPriority) {
                    highestPriority = removed;
                }
            }
            System.out.println("Highest Priority Removed: " + highestPriority);
        } catch (Exception e) {
            System.err.println("Error in performInsertionsAndRemovals for PriorityQueue: " + e.getMessage());
        }
    }
```

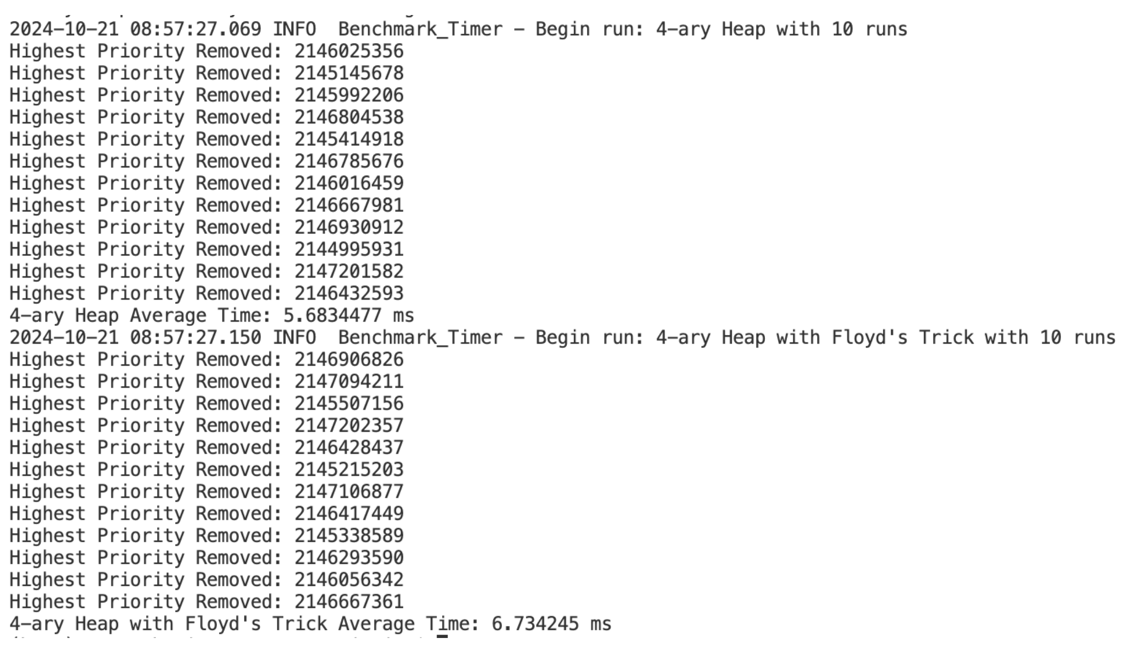


```java
    private static void performInsertionsAndRemovalsQuad(QuadPriorityQueue<Integer> pq) {
        try {
            for (int i = 0; i < insertions; i++) {
                pq.give(rand.nextInt());
            }
            Integer highestPriority = null;
            for (int i = 0; i < removals; i++) {
                Integer removed = pq.take();
                if (highestPriority == null || removed > highestPriority) {
                    highestPriority = removed;
                }
            }
            System.out.println("Highest Priority Removed: " + highestPriority);
        } catch (Exception e) {
            System.err.println("Error in performInsertionsAndRemovalsQuad for QuadPriorityQueue: " + e.getMessage());
        }
    }
}
```

Unit Test Screenshots:

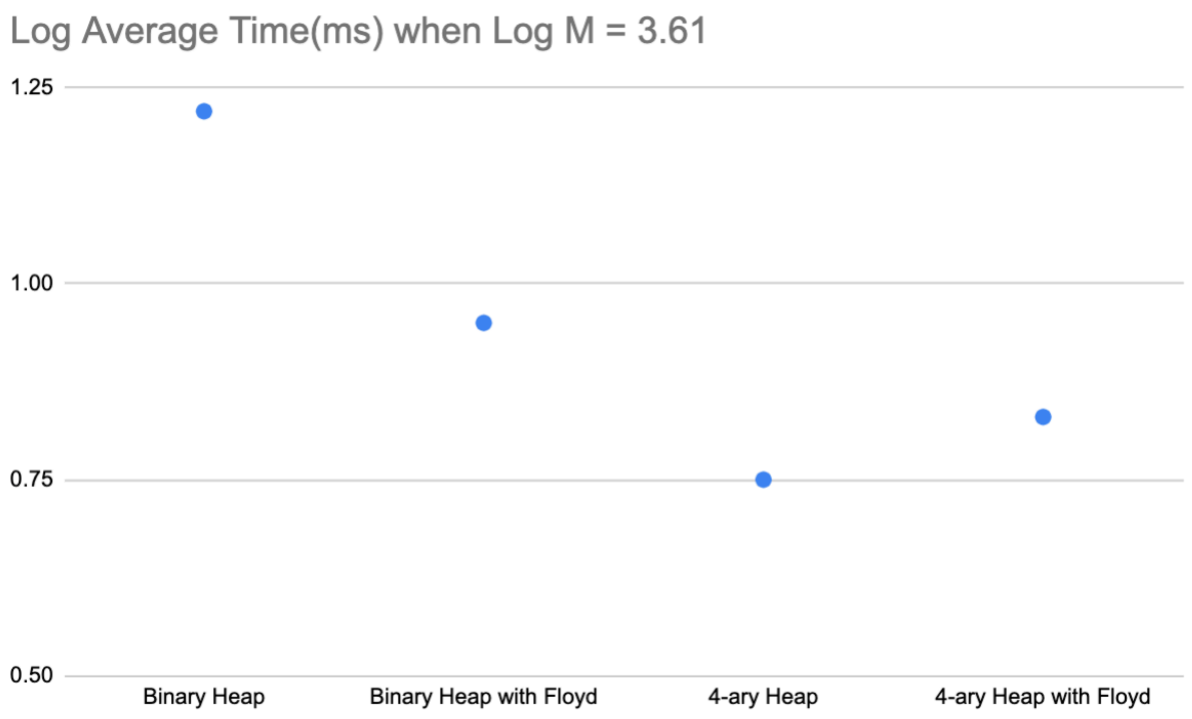


```
2024-10-21 08:57:26.691 INFO  Benchmark_Timer - Begin run: Basic Binary Heap with 10 runs
Highest Priority Removed: 2147098260
Highest Priority Removed: 2147257939
Highest Priority Removed: 2147373520
Highest Priority Removed: 2147369780
Highest Priority Removed: 2147471165
Highest Priority Removed: 2147400740
Highest Priority Removed: 2147460087
Highest Priority Removed: 2147466272
Highest Priority Removed: 2146990502
Highest Priority Removed: 2147422412
Highest Priority Removed: 2147408102
Highest Priority Removed: 2147470205
Basic Binary Heap Average Time: 16.8370786 ms
2024-10-21 08:57:26.927 INFO  Benchmark_Timer - Begin run: Binary Heap with Floyd's Trick with 10 runs
Highest Priority Removed: 2146892813
Highest Priority Removed: 2146836268
Highest Priority Removed: 2146685809
Highest Priority Removed: 2146978838
Highest Priority Removed: 2146852699
Highest Priority Removed: 2147418816
Highest Priority Removed: 2147400074
Highest Priority Removed: 2146892847
Highest Priority Removed: 2147431047
Highest Priority Removed: 2147221043
Highest Priority Removed: 2147390346
Highest Priority Removed: 2147070929
Binary Heap with Floyd's Trick Average Time: 8.9305279 ms
```



```
2024-10-21 08:57:27.069 INFO  Benchmark_Timer - Begin run: 4-ary Heap with 10 runs
Highest Priority Removed: 2146025356
Highest Priority Removed: 2145145678
Highest Priority Removed: 2145992206
Highest Priority Removed: 2146804538
Highest Priority Removed: 2145414918
Highest Priority Removed: 2146785676
Highest Priority Removed: 2146016459
Highest Priority Removed: 2146667981
Highest Priority Removed: 2146930912
Highest Priority Removed: 2144995931
Highest Priority Removed: 2147201582
Highest Priority Removed: 2146432593
4-ary Heap Average Time: 5.6834477 ms
2024-10-21 08:57:27.150 INFO  Benchmark_Timer - Begin run: 4-ary Heap with Floyd's Trick with 10 runs
Highest Priority Removed: 2146906826
Highest Priority Removed: 2147094211
Highest Priority Removed: 2145507156
Highest Priority Removed: 2147202357
Highest Priority Removed: 2146428437
Highest Priority Removed: 2145215203
Highest Priority Removed: 2147106877
Highest Priority Removed: 2146417449
Highest Priority Removed: 2145338589
Highest Priority Removed: 2146293590
Highest Priority Removed: 2146056342
Highest Priority Removed: 2146667361
4-ary Heap with Floyd's Trick Average Time: 6.734245 ms
```

Observations:



4-ary heap performs better than the binary heap, and the Floyd trick can help improve the performance.