COLLECTION BENCHMARK

TESTING THE PERFORMANCE OF SEVERAL JAVA COLLECTION IMPLEMENTATIONS

By ByteConstruction

- 220392X – Mendis A. R. S.
- 220407C – Muthumala V. D. W.
- 220694B – Wickramage W. K. T. V.
- 220303E – Kariyawasam U. G. R. T.

PROGRAM DESIGN DESCRIPTION

The provided code conducts performance tests on the below listed Java Collection implementations.

- HashSet
- TreeSet
- LinkedHashSet
- ArrayList
- LinkedList
- ArrayDeque
- PriorityQueue
- HashMap
- TreeMap
- LinkedHashMap

It systematically measures the time taken for common operations like **“add”** a given element to the collection, check if a collection **“contains”** a given element , **“remove”** a given element from the collection, **“clear”** all the elements in the collection.

The program first initiates the collections with their default initialCapacity values and loadFactor values. Then the program loads the above initiated collections with 100,000 random integer objects (from 0 to 99,999) and performs the above operations 100 times to calculate average execution times.

JAVA CODE

```
import java.util.*;
import java.util.concurrent.ThreadLocalRandom;

public class CollectionPerformanceTest {
    public static void main(String[] args) {
        int numElements = 100000;
        int numTests = 100;
        int[][] times = new int[][]{{}};
        //creating a 2-D list of lists structure with
arrayList
        ArrayList<ArrayList<Float>> timeList = new
ArrayList<>();
        ArrayList<Float> innerList = new
ArrayList<>();

        // Instantiate collections
        HashSet<Integer> hashSet = new HashSet<>();
        TreeSet<Integer> treeSet = new TreeSet<>();
        LinkedHashSet<Integer> linkedHashSet = new
LinkedHashSet<>();
        ArrayList<Integer> arrayList = new
ArrayList<>();
        LinkedList<Integer> linkedList = new
LinkedList<>();
        ArrayDeque<Integer> arrayDeque = new
ArrayDeque<>();
        PriorityQueue<Integer> priorityQueue = new
PriorityQueue<>();
        HashMap<Integer, Integer> hashMap = new
HashMap<>();
        TreeMap<Integer, Integer> treeMap = new
TreeMap<>();
        LinkedHashMap<Integer, Integer> linkedHashMap
= new LinkedHashMap<>();

        // Fill collections with random integers
        Random rand = new Random();
        while (hashSet.size() < numElements) {
            int randInt =
ThreadLocalRandom.current().nextInt(0, 100000);
            if (hashSet.contains(randInt)) {
                continue;
            }
```

```
            hashSet.add(randInt);
            treeSet.add(randInt);
            linkedHashSet.add(randInt);
            arrayList.add(randInt);
            linkedList.add(randInt);
            arrayDeque.add(randInt);
            priorityQueue.add(randInt);
            hashMap.put(randInt, randInt);
            treeMap.put(randInt, randInt);
            linkedHashMap.put(randInt, randInt);
        }

        //Testing the performance of HashSet
        HashSet<Integer> hashSet_bkp = hashSet;
        long startTime, endTime, totalTime;
        float cummulativeTime = 0;

        // Test "add" method
        for (int i = 0; i < numTests; i++) {
            hashSet = hashSet_bkp;
            startTime = System.nanoTime();
            hashSet.add(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        float averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("HashSet add time: " +
averageTime + " ns");

        // Test "contains" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            hashSet = hashSet_bkp;
            startTime = System.nanoTime();
            boolean found =
hashSet.contains(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("HashSet contains time: "
+ averageTime + " ns");
```

```
        // Test "remove" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            hashSet = hashSet_bkp;
            startTime = System.nanoTime();
            hashSet.remove(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("HashSet remove time: " +
averageTime + " ns");

        // Test "clear" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            hashSet = hashSet_bkp;
            startTime = System.nanoTime();
            hashSet.clear();
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("HashSet clear time: " +
averageTime + " ns");
        timeList.add(innerList);
        System.out.println(innerList);
        innerList.clear();
        hashSet_bkp.clear();


        //Testing the performance of TreeSet
        TreeSet<Integer> treeSet_bkp = treeSet;
        cummulativeTime = 0;

        for (int i = 0; i < numTests; i++) {
            treeSet = treeSet_bkp;
            startTime = System.nanoTime();
            treeSet.add(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
```

```
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("treeSet add time: " +
averageTime + " ns");

        // Test "contains" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            treeSet = treeSet_bkp;
            startTime = System.nanoTime();
            boolean found =
treeSet.contains(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("treeSet contains time: "
+ averageTime + " ns");

        // Test "remove" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            treeSet = treeSet_bkp;
            startTime = System.nanoTime();
            treeSet.remove(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("treeSet remove time: " +
averageTime + " ns");

        // Test "clear" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            treeSet = treeSet_bkp;
            startTime = System.nanoTime();
            treeSet.clear();
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
```

```
        innerList.add(averageTime);
        System.out.println("treeSet clear time: " +
averageTime + " ns");

        timeList.add(innerList);
        System.out.println(innerList);
        innerList.clear();
        treeSet_bkp.clear();


        //Testing the performance of linkedHashSet
        LinkedHashSet<Integer> linkedHashSet_bkp =
linkedHashSet;
        cummulativeTime = 0;

        for (int i = 0; i < numTests; i++) {
            linkedHashSet = linkedHashSet_bkp;
            startTime = System.nanoTime();
            linkedHashSet.add(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("linkedHashSet add time: "
+ averageTime + " ns");

        // Test "contains" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            linkedHashSet = linkedHashSet_bkp;
            startTime = System.nanoTime();
            boolean found =
linkedHashSet.contains(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("linkedHashSet contains
time: " + averageTime + " ns");

        // Test "remove" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            linkedHashSet = linkedHashSet_bkp;
```

```
            startTime = System.nanoTime();

linkedHashSet.remove(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("linkedHashSet remove
time: " + averageTime + " ns");

        // Test "clear" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            linkedHashSet = linkedHashSet_bkp;
            startTime = System.nanoTime();
            linkedHashSet.clear();
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("linkedHashSet clear time:
" + averageTime + " ns");
        timeList.add(innerList);
        System.out.println(innerList);
        innerList.clear();
        linkedHashSet_bkp.clear();


        //Testing the performance of arrayList
        ArrayList<Integer> arrayList_bkp = arrayList;
        cummulativeTime = 0;

        for (int i = 0; i < numTests; i++) {
            arrayList = arrayList_bkp;
            startTime = System.nanoTime();
            arrayList.add(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
```

```
        System.out.println("arrayList add time: " +
averageTime + " ns");

        // Test "contains" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            arrayList = arrayList_bkp;
            startTime = System.nanoTime();
            boolean found =
arrayList.contains(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("arrayList contains time:
" + averageTime + " ns");

        // Test "remove" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            arrayList = arrayList_bkp;
            startTime = System.nanoTime();
            arrayList.remove(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("arrayList remove time: "
+ averageTime + " ns");

        // Test "clear" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            arrayList = arrayList_bkp;
            startTime = System.nanoTime();
            arrayList.clear();
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
```

```
        System.out.println("arrayList clear time: " +
averageTime + " ns");

        timeList.add(innerList);
        System.out.println(innerList);
        System.out.println();
        innerList.clear();
        arrayList_bkp.clear();


        //Testing the performance of linkedList
        LinkedList<Integer> linkedList_bkp =
linkedList;
        cummulativeTime = 0;

        for (int i = 0; i < numTests; i++) {
            linkedList = linkedList_bkp;
            startTime = System.nanoTime();
            linkedList.add(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("linkedList add time: " +
averageTime + " ns");

        // Test "contains" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            linkedList = linkedList_bkp;
            startTime = System.nanoTime();
            boolean found =
linkedList.contains(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("linkedList contains time:
" + averageTime + " ns");

        // Test "remove" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            linkedList = linkedList_bkp;
```

```
            startTime = System.nanoTime();
            linkedList.remove(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("linkedList remove time: "
+ averageTime + " ns");

        // Test "clear" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            linkedList = linkedList_bkp;
            startTime = System.nanoTime();
            linkedList.clear();
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("linkedList clear time: "
+ averageTime + " ns");
        timeList.add(innerList);
        System.out.println(innerList);
        System.out.println();
        innerList.clear();
        linkedList_bkp.clear();


        //Testing the performance of arrayDeque
        ArrayDeque<Integer> arrayDeque_bkp =
arrayDeque;
        cummulativeTime = 0;

        for (int i = 0; i < numTests; i++) {
            arrayDeque = arrayDeque_bkp;
            startTime = System.nanoTime();
            arrayDeque.add(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
```

```
        System.out.println("arrayDeque add time: " +
averageTime + " ns");

        // Test "contains" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            arrayDeque = arrayDeque_bkp;
            startTime = System.nanoTime();
            boolean found =
arrayDeque.contains(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("arrayDeque contains time:
" + averageTime + " ns");

        // Test "remove" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            arrayDeque = arrayDeque_bkp;
            startTime = System.nanoTime();
            arrayDeque.remove(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("arrayDeque remove time: "
+ averageTime + " ns");

        // Test "clear" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            arrayDeque = arrayDeque_bkp;
            startTime = System.nanoTime();
            arrayDeque.clear();
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
```

```
        System.out.println("arrayDeque clear time: "
+ averageTime + " ns");

        timeList.add(innerList);
        System.out.println(innerList);
        System.out.println();
        innerList.clear();
        arrayDeque_bkp.clear();


        //Testing the performance of priorityQueue
        PriorityQueue<Integer> priorityQueue_bkp =
priorityQueue;
        cummulativeTime = 0;

        for (int i = 0; i < numTests; i++) {
            priorityQueue = priorityQueue_bkp;
            startTime = System.nanoTime();
            priorityQueue.add(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("priorityQueue add time: "
+ averageTime + " ns");

        // Test "contains" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            priorityQueue = priorityQueue_bkp;
            startTime = System.nanoTime();
            boolean found =
priorityQueue.contains(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("priorityQueue contains
time: " + averageTime + " ns");

        // Test "remove" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            priorityQueue = priorityQueue_bkp;
```

```
            startTime = System.nanoTime();

priorityQueue.remove(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("priorityQueue remove
time: " + averageTime + " ns");

        // Test "clear" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            priorityQueue = priorityQueue_bkp;
            startTime = System.nanoTime();
            priorityQueue.clear();
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("priorityQueue clear time:
" + averageTime + " ns");

        timeList.add(innerList);
        System.out.println(innerList);
        System.out.println();
        innerList.clear();
        priorityQueue_bkp.clear();


        //Testing the performance of hashMap
        HashMap<Integer, Integer> hashMap_bkp =
hashMap;
        cummulativeTime = 0;

        for (int i = 0; i < numTests; i++) {
            hashMap = hashMap_bkp;
            startTime = System.nanoTime();
            hashMap.put(rand.nextInt(100000),
rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
```

```
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("hashMap add time: " +
averageTime + " ns");

        // Test "contains" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            hashMap = hashMap_bkp;
            startTime = System.nanoTime();
            boolean found =
hashMap.containsKey(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("hashMap contains time: "
+ averageTime + " ns");

        // Test "remove" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            hashMap = hashMap_bkp;
            startTime = System.nanoTime();
            hashMap.remove(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("hashMap remove time: " +
averageTime + " ns");

        // Test "clear" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            hashMap = hashMap_bkp;
            startTime = System.nanoTime();
            hashMap.clear();
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
```

```
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("hashMap clear time: " +
averageTime + " ns");

        timeList.add(innerList);
        System.out.println(innerList);
        System.out.println();
        innerList.clear();
        hashMap_bkp.clear();


        //Testing the performance of treeMap
        TreeMap<Integer, Integer> treeMap_bkp =
treeMap;
        cummulativeTime = 0;

        for (int i = 0; i < numTests; i++) {
            treeMap = treeMap_bkp;
            startTime = System.nanoTime();
            treeMap.put(rand.nextInt(100000),
rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("treeMap add time: " +
averageTime + " ns");

        // Test "contains" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            treeMap = treeMap_bkp;
            startTime = System.nanoTime();
            boolean found =
treeMap.containsKey(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("treeMap contains time: "
+ averageTime + " ns");

        // Test "remove" method
```

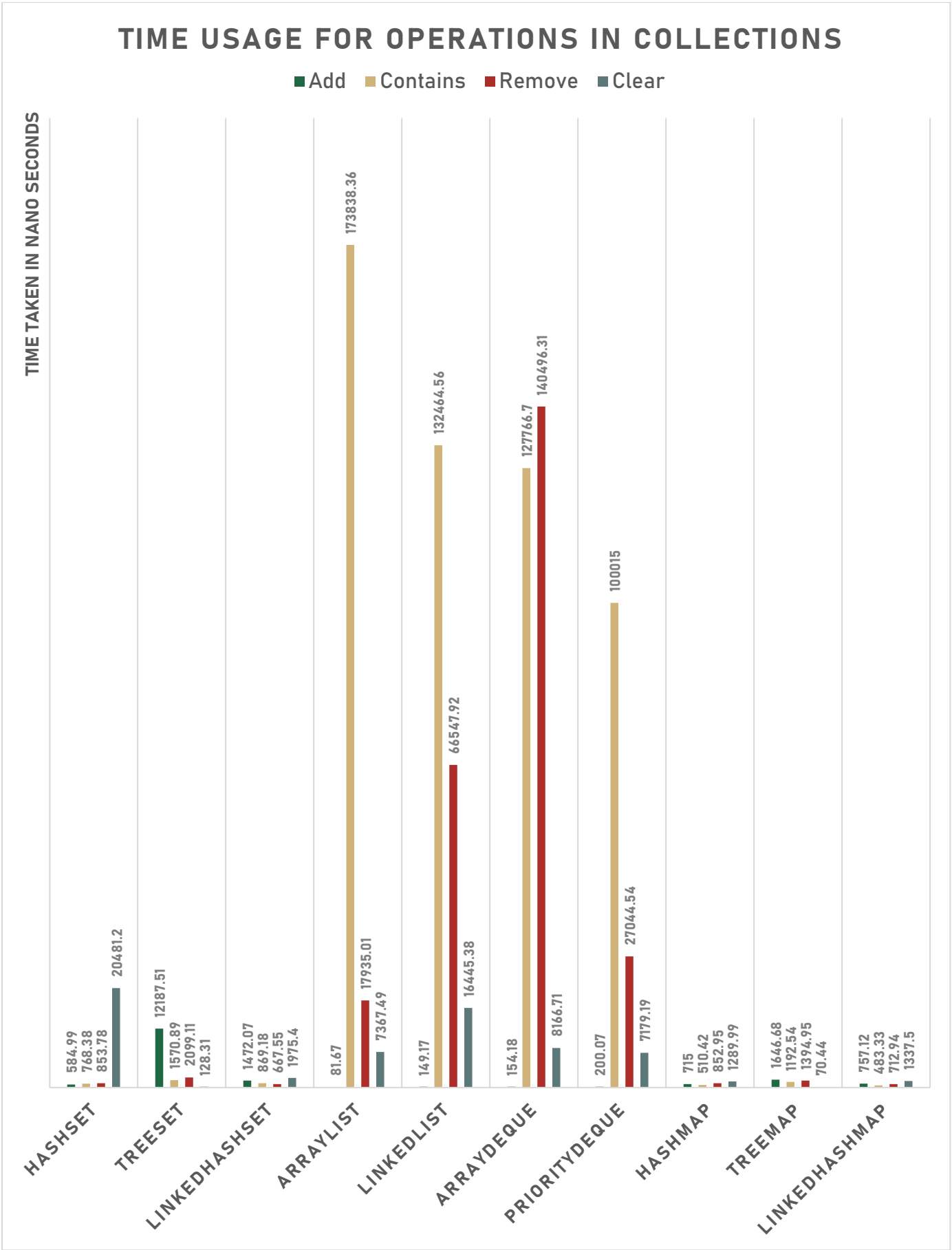
```
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            treeMap = treeMap_bkp;
            startTime = System.nanoTime();
            treeMap.remove(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("treeMap remove time: " +
averageTime + " ns");

        // Test "clear" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            treeMap = treeMap_bkp;
            startTime = System.nanoTime();
            treeMap.clear();
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("treeMap clear time: " +
averageTime + " ns");

        timeList.add(innerList);
        System.out.println(innerList);
        System.out.println();
        innerList.clear();
        treeMap_bkp.clear();


        //Testing the performance of linkedHashMap
        LinkedHashMap<Integer, Integer>
linkedHashMap_bkp = linkedHashMap;
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            linkedHashMap = linkedHashMap_bkp;
            startTime = System.nanoTime();
            linkedHashMap.put(rand.nextInt(100000),
rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
```

```
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("linkedHashMap add time: "
+ averageTime + " ns");

        // Test "contains" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            linkedHashMap = linkedHashMap_bkp;
            startTime = System.nanoTime();
            boolean found =
linkedHashMap.containsKey(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("linkedHashMap contains
time: " + averageTime + " ns");

        // Test "remove" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            linkedHashMap = linkedHashMap_bkp;
            startTime = System.nanoTime();

linkedHashMap.remove(rand.nextInt(100000));
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("linkedHashMap remove
time: " + averageTime + " ns");

        // Test "clear" method
        cummulativeTime = 0;
        for (int i = 0; i < numTests; i++) {
            linkedHashMap = linkedHashMap_bkp;
            startTime = System.nanoTime();
            linkedHashMap.clear();
            endTime = System.nanoTime();
            totalTime = endTime - startTime;
```

```
            cummulativeTime = cummulativeTime +
totalTime;
        }
        averageTime = cummulativeTime/100;
        innerList.add(averageTime);
        System.out.println("linkedHashMap clear time:
" + averageTime + " ns");

        timeList.add(innerList);
        System.out.println(innerList);
        System.out.println();
    }
}
```

COMPARISON

	Add (nanoseconds)	Contains (nanoseconds)	Remove (nanoseconds)	Clear (nanoseconds)
HashSet	584.99	768.38	853.78	20481.2
TreeSet	12187.51	1570.89	2099.11	128.31
LinkedHashSet	1472.07	869.18	667.55	1975.4
ArrayList	81.67	173838.36	17935.01	7367.49
LinkedList	149.17	132464.56	66547.92	16445.38
ArrayDeque	154.18	127766.7	140496.31	8166.71
PriorityDeque	200.07	100015.0	27044.54	7179.19
HashMap	715.0	510.42	852.95	1289.99
TreeMap	1646.68	1192.54	1394.95	70.44
LinkedHashMap	757.12	483.33	712.94	1337.5


TIME USAGE FOR OPERATIONS IN COLLECTIONS
Add
Contains
Remove
Clear
TIME TAKEN IN NANO SECONDS
584.99
768.38
853.78
20481.2
12187.51
1570.89
2099.11
128.31
1472.07
869.18
667.55
1975.4
81.67
173838.36
17935.01
7367.49
149.17
132464.56
66547.92
16445.38
154.18
127766.7
140496.31
8166.71
200.07
100015
27044.54
7179.19
715
510.42
852.95
1289.99
1646.68
1192.54
1394.95
70.44
757.12
483.33
712.94
1337.5
HASHSET
TREESET
LINKEDHASHSET
ARRAYLIST
LINKEDLIST
ARRAYDEQUE
PRIORITYDEQUE
HASHMAP
TREEMAP
LINKEDHASHMAP

DISCUSSION ON THE REASONS OF PERFORMANCE VARIATIONS

Implementation Differences for Collections	Different Java Collection implementations have different underlying data structures which can lead to varying of performance.
Algorithmic Complexity	The efficiency of Algorithms used in the implementation of collection operations can significantly impact the performance.
Hardware Considerations	Performance can also vary depending on the hardware factors like CPU speed and cache behavior.
Effect of Input Data	If the Integer Objects created by rand.nextInt() happen to result in collisions with elements in the Set-based collections, it may lead to degraded performances.