Author

Kang Shentu

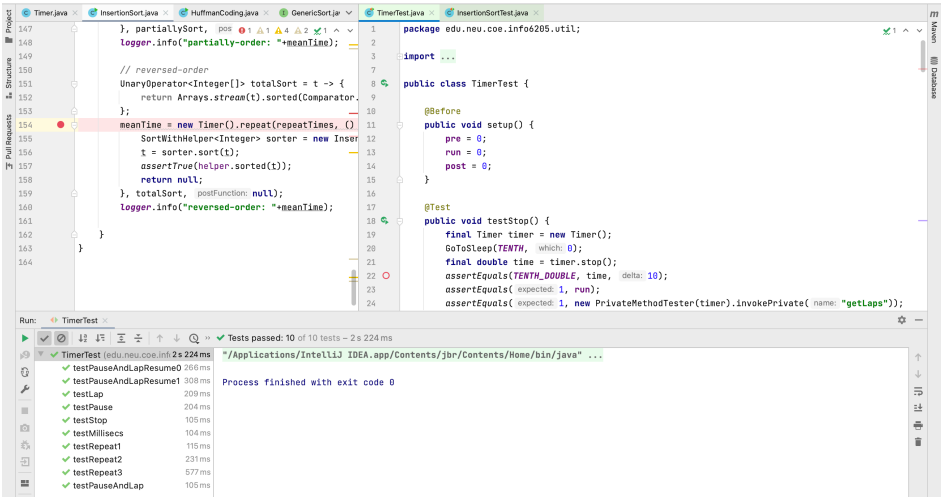
001569432

Part1

Implement Code

```
    // 1.
    private static long getClock() {
        // TO BE IMPLEMENTED
        return System.nanoTime();
    }
    // 2.
    private static double toMillisecs(long ticks) {
        // TO BE IMPLEMENTED
        return ticks * 1E-6;
    }
    // 3.
    public <T, U> double repeat(int n, Supplier<T>
supplier, Function<T, U> function, UnaryOperator<T>
preFunction, Consumer<U> postFunction) {
        logger.trace("repeat: with " + n + " runs");
        this.running = false;
        this.ticks = 0;
        for (int i = 0; i < n; i++) {
            T value = supplier.get();
            if(Objects.nonNull(preFunction))
                value = preFunction.apply(value);
            resume();
            U result = function.apply(value);
            pauseAndLap();
            if(Objects.nonNull(postFunction))
                postFunction.accept(result);
        }
        return meanLapTime();
    }
```

Passed Test



Part2

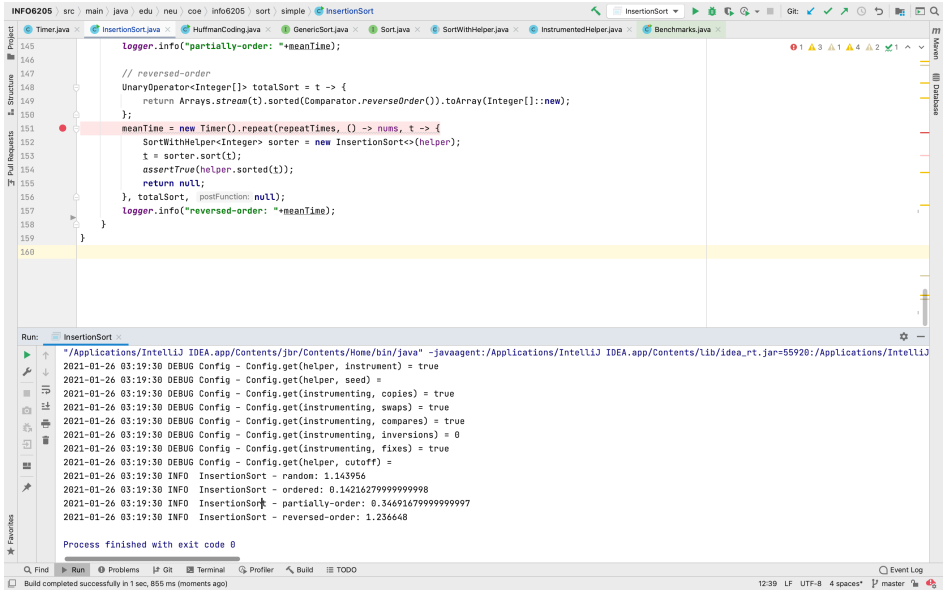
Implement Code

```
    // 1.
    public void sort(X[] xs, int from, int to) {
        rangeCheck(xs.length, from, to);
        final Helper<X> helper = getHelper();
        // TO BE IMPLEMENTED
        for (int i = from+1; i < to; i++) {
            for (int j = i - 1; j >= 0; j--) {
                if(helper.compare(xs, j, j+1) > 0) {
                // if (xs[j].compareTo(xs[j + 1]) > 0) {
                    helper.swap(xs, j, j + 1);
                } else {
                    break;
                }
            }
        }
    }
    // 2.
    static void rangeCheck(int arrayLength, int
fromIndex, int toIndex) {
        if (fromIndex > toIndex) {
            throw new
IllegalArgumentException("fromIndex(" + fromIndex + ") >
toIndex(" + toIndex + ")");
        } else if (fromIndex < 0) {
```

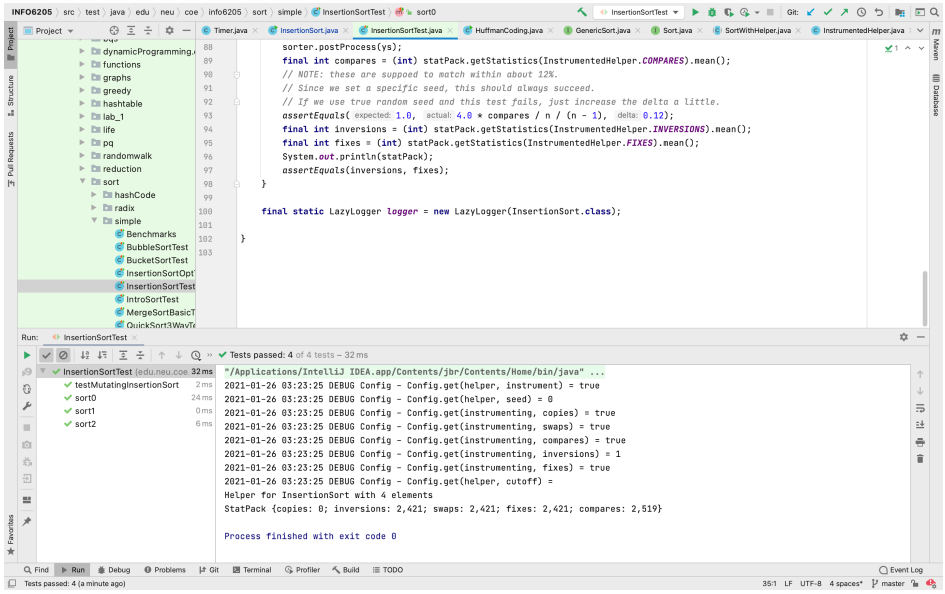
```
            throw new
ArrayIndexOutOfBoundsException(fromIndex);
        } else if (toIndex > arrayLength) {
            throw new
ArrayIndexOutOfBoundsException(toIndex);
        }
    }
    // 3. main method
public static void main(String[] args) throws IOException
{
        int repeatTimes = 5;
        int n = 100;

        Config config = Config.load();
        Helper<Integer> helper =
HelperFactory.create("InsertionSort", n, config);
        helper.init(n);
        Integer[] nums = helper.random(Integer.class, r -
> r.nextInt(1000));
        // random
        double meanTime = new Timer().repeat(repeatTimes,
() -> nums, t -> {
            SortWithHelper<Integer> sorter = new
InsertionSort<>(helper);
            t = sorter.sort(t);
            assertTrue(helper.sorted(t));
            return null;
        });
        logger.info("random: "+meanTime);

        // ordered
        meanTime = new Timer().repeat(repeatTimes, () ->
nums, t -> {
            SortWithHelper<Integer> sorter = new
InsertionSort<>(helper);
            t = sorter.sort(t);
            assertTrue(helper.sorted(t));
            return null;
        });
        logger.info("ordered: "+meanTime);

        // partially-order
        UnaryOperator<Integer[]> partiallySort = t->{
```

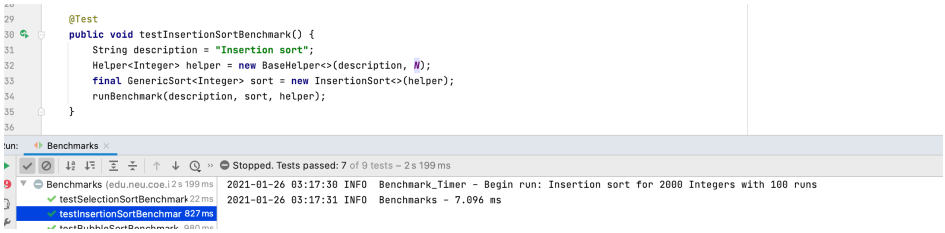
```
            SortWithHelper<Integer> sorter = new
InsertionSort<>(helper);
            Random random = new Random();
            int max = random.nextInt(t.length);
            int min = random.nextInt(max);
            sorter.sort(t, min, max);
            return t;
        };
        meanTime = new Timer().repeat(repeatTimes, () ->
nums, t -> {
            SortWithHelper<Integer> sorter = new
InsertionSort<>(helper);
            t = sorter.sort(t);
            assertTrue(helper.sorted(t));
            return null;
        }, partiallySort, null);
        logger.info("partially-order: "+meanTime);

        // reversed-order
        UnaryOperator<Integer[]> totalSort = t -> {
            return
Arrays.stream(t).sorted(Comparator.reverseOrder()).toArra
y(Integer[]::new);
        };
        meanTime = new Timer().repeat(repeatTimes, () ->
nums, t -> {
            SortWithHelper<Integer> sorter = new
InsertionSort<>(helper);
            t = sorter.sort(t);
            assertTrue(helper.sorted(t));
            return null;
        }, totalSort, null);
        logger.info("reversed-order: "+meanTime);
    }
```

Screenshot


logger.info("partially-order: "+meanTime);
// reversed-order
UnaryOperator<Integer[]> totalSort = t -> {
return Arrays.stream(t).sorted(Comparator.reverseOrder()).toArray(Integer[]::new);
};
meanTime = new Timer().repeat(repeatTimes, () -> nums, t -> {
SortWithHelper<Integer> sorter = new InsertionSort<>(helper);
t = sorter.sort(t);
assertTrue(helper.sorted(t));
return null;
}, totalSort, postFunction: null);
logger.info("reversed-order: "+meanTime);
}
}
Run: InsertionSort
"/Applications/IntelliJ IDEA.app/Contents/jbr/Contents/Home/bin/java" -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=55920:/Applications/IntelliJ
2021-01-26 03:19:30 DEBUG Config - Config.get(helper, instrument) = true
2021-01-26 03:19:30 DEBUG Config - Config.get(helper, seed) =
2021-01-26 03:19:30 DEBUG Config - Config.get(instrumenting, copies) = true
2021-01-26 03:19:30 DEBUG Config - Config.get(instrumenting, swaps) = true
2021-01-26 03:19:30 DEBUG Config - Config.get(instrumenting, compares) = true
2021-01-26 03:19:30 DEBUG Config - Config.get(instrumenting, inversions) = 0
2021-01-26 03:19:30 DEBUG Config - Config.get(instrumenting, fixes) = true
2021-01-26 03:19:30 DEBUG Config - Config.get(helper, cutoff) =
2021-01-26 03:19:30 INFO InsertionSort - random: 1.143956
2021-01-26 03:19:30 INFO InsertionSort - ordered: 0.14216279999999998
2021-01-26 03:19:30 INFO InsertionSort - partially-order: 0.34691679999999997
2021-01-26 03:19:30 INFO InsertionSort - reversed-order: 1.236648
Process finished with exit code 0

Passed Test


sorter.postProcess(ys);
final int compares = (int) statPack.getStatistics(InstrumentedHelper.COMPARES).mean();
// NOTE: these are supposed to match within about 12%.
// Since we set a specific seed, this should always succeed.
// If we use true random seed and this test fails, just increase the delta a little.
assertEquals(expected: 1.0, actual: 4.0 * compares / n / (n - 1), delta: 0.12);
final int inversions = (int) statPack.getStatistics(InstrumentedHelper.INVERSIONS).mean();
final int fixes = (int) statPack.getStatistics(InstrumentedHelper.FIXES).mean();
System.out.println(statPack);
assertEquals(inversions, fixes);
}
final static LazyLogger logger = new LazyLogger(InsertionSort.class);
}
Run: InsertionSortTest
Tests passed: 4 of 4 tests – 32 ms
InsertionSortTest (edu.neu.coe 32 ms
testMutatingInsertionSort 2 ms
sort0 24 ms
sort1 0 ms
sort2 6 ms
"/Applications/IntelliJ IDEA.app/Contents/jbr/Contents/Home/bin/java" ...
2021-01-26 03:23:25 DEBUG Config - Config.get(helper, instrument) = true
2021-01-26 03:23:25 DEBUG Config - Config.get(helper, seed) = 0
2021-01-26 03:23:25 DEBUG Config - Config.get(instrumenting, copies) = true
2021-01-26 03:23:25 DEBUG Config - Config.get(instrumenting, swaps) = true
2021-01-26 03:23:25 DEBUG Config - Config.get(instrumenting, compares) = true
2021-01-26 03:23:25 DEBUG Config - Config.get(instrumenting, inversions) = 1
2021-01-26 03:23:25 DEBUG Config - Config.get(instrumenting, fixes) = true
2021-01-26 03:23:25 DEBUG Config - Config.get(helper, cutoff) =
Helper for InsertionSort with 4 elements
StatPack {copies: 0; inversions: 2,421; swaps: 2,421; fixes: 2,421; compares: 2,519}
Process finished with exit code 0


@Test
public void testInsertionSortBenchmark() {
String description = "Insertion sort";
Helper<Integer> helper = new BaseHelper<>(description, N);
final GenericSort<Integer> sort = new InsertionSort<>(helper);
runBenchmark(description, sort, helper);
}
Benchmarks
Stopped. Tests passed: 7 of 9 tests – 2 s 199 ms
Benchmarks (edu.neu.coe.i 2 s 199 ms
testSelectionSortBenchmark 22 ms
testInsertionSortBenchmar 827 ms
2021-01-26 03:17:30 INFO Benchmark_Timer - Begin run: Insertion sort for 2000 Integers with 100 runs
2021-01-26 03:17:31 INFO Benchmarks - 7.096 ms

Conclusion

As the screenshot shown, the reversed-order array is the most time-consuming.

The number of comparisons is not certain. The less the number of comparisons, the more the data movement after the insertion point, especially when the total amount of data is huge.

The best time complexity is O(n) when the array is in order, and the worst time complexity is O(n^2) when the array is in reverse order. It is known that the average time complexity of inserting an element into an ordered array is O(n), then n operations are performed, so the average time complexity is O(n^2).