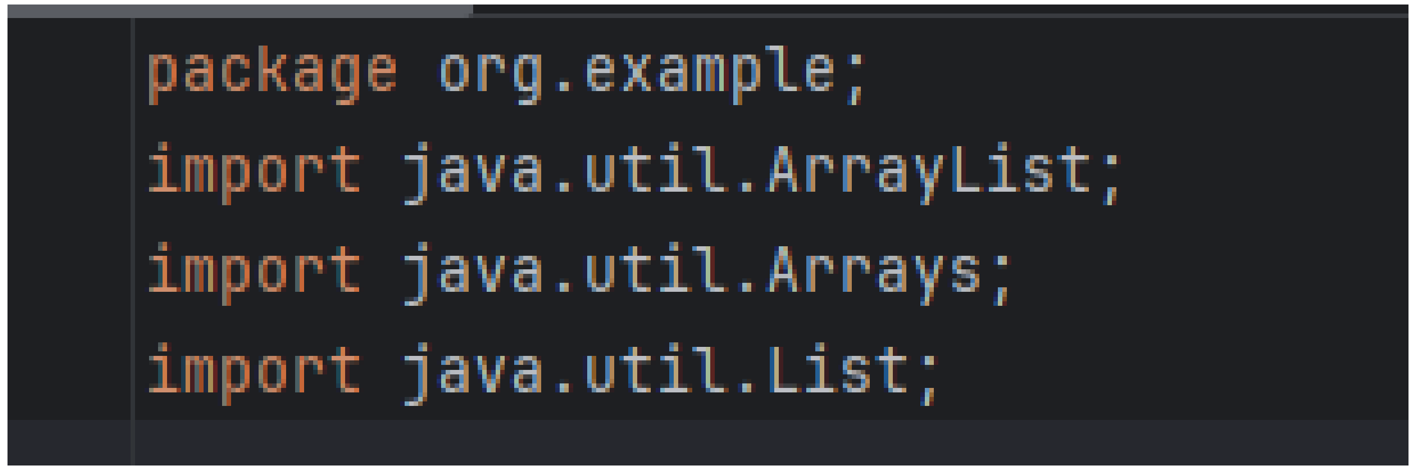
Pair 2

Student B: Kendebayev Nurassyl

Student A: Aktilek

this is Nurassyl’s anasys of Aktilek’s code

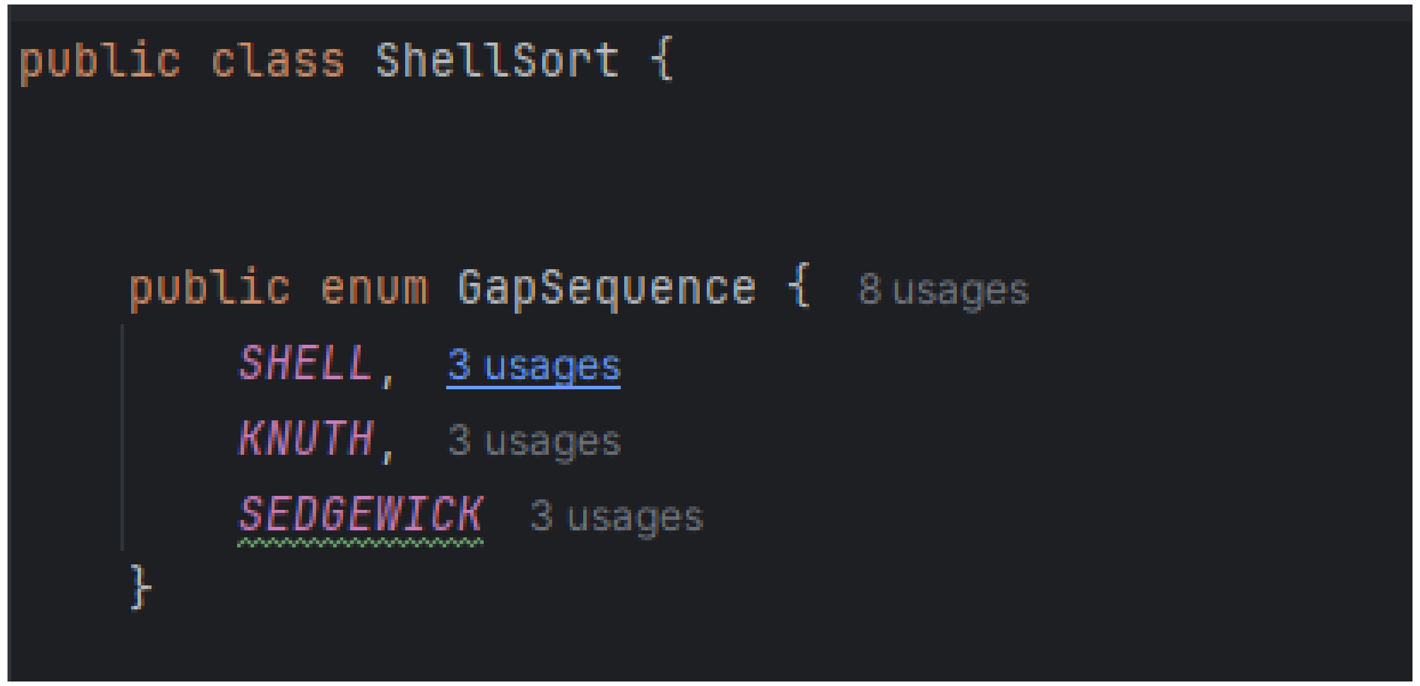
Shell sort code review



```java
package org.example;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
```

Import	Why it’s needed
package org.example	Declares where your class belongs (organization).
import java.util.ArrayList	To create resizable lists for gap sequences.
import java.util.Arrays	To copy and print arrays easily.
import java.util.List	To use the general List interface for collections.

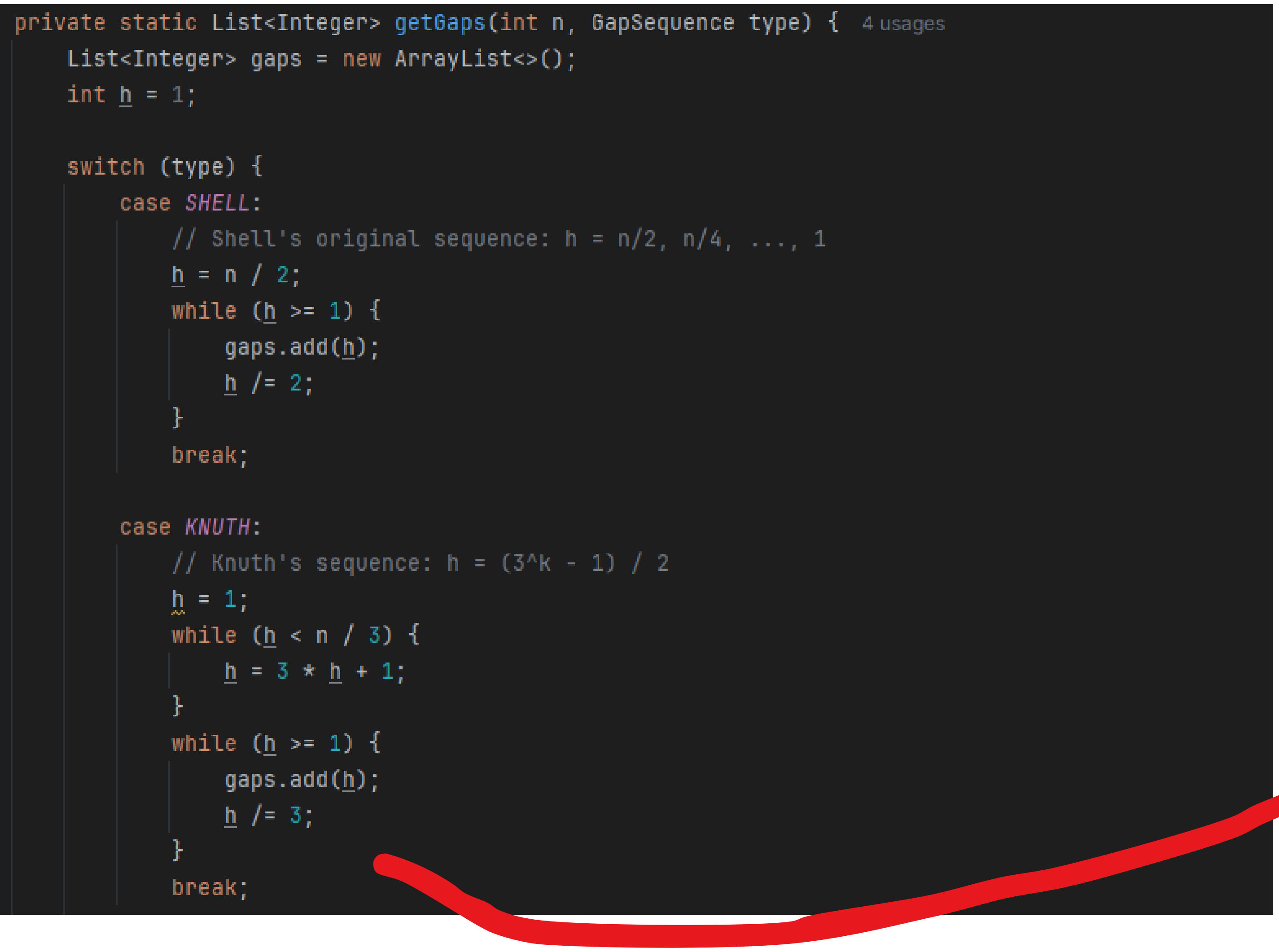
There are 4 methods in code



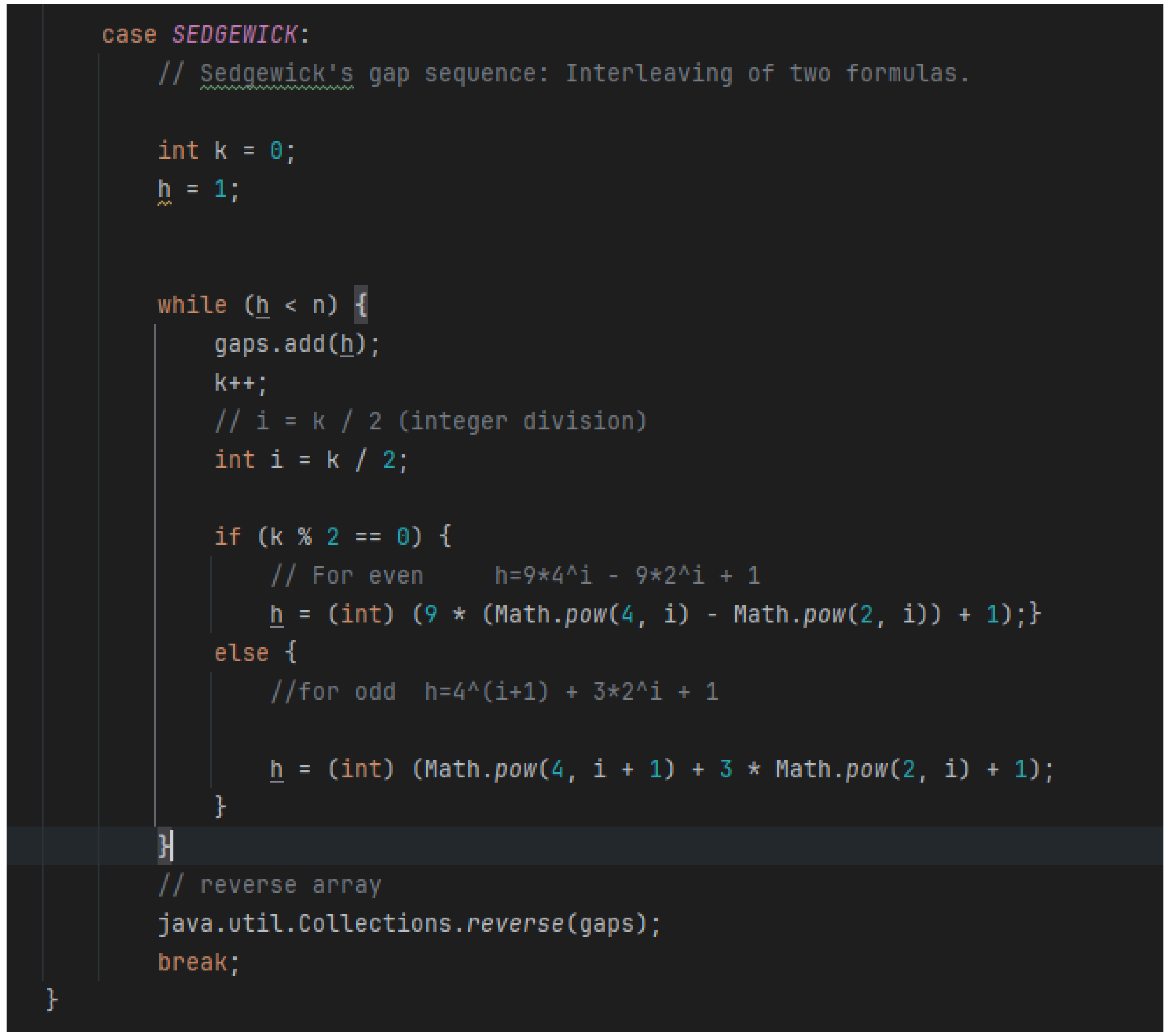
```
public class ShellSort {


    public enum GapSequence {  8 usages
        SHELL,  3 usages
        KNUTH,  3 usages
        SEDGEWICK  3 usages
    }
```

1)defines three types of gap sequences — SHELL, KNUTH, SEDGEWICK

2)This method generates an array of gap values for Shell Sort based on the chosen gap sequence type (Shell, Knuth, or Sedgewick).

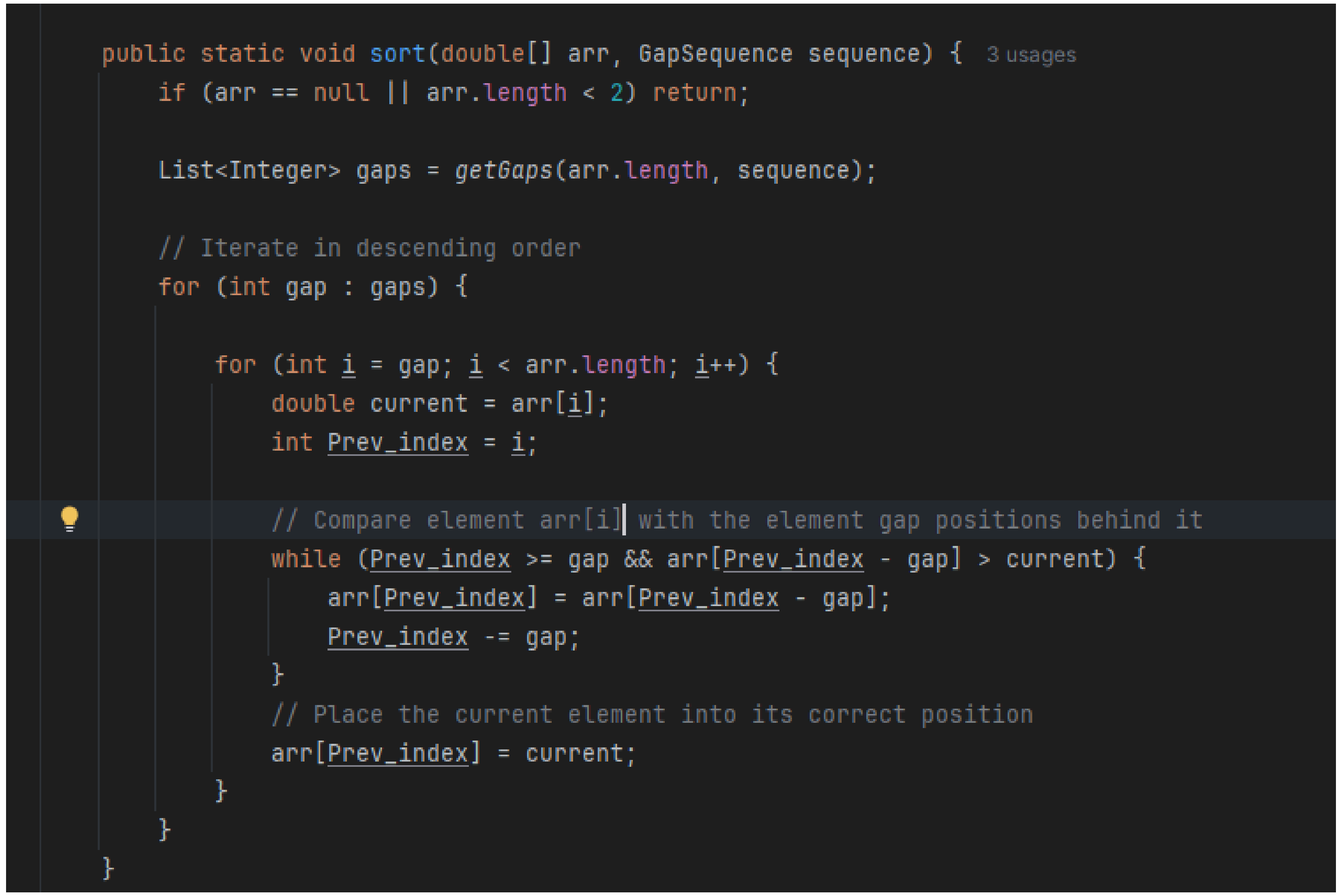


```java
private static List<Integer> getGaps(int n, GapSequence type) {  4 usages
    List<Integer> gaps = new ArrayList<>();
    int h = 1;

    switch (type) {
        case SHELL:
            // Shell's original sequence: h = n/2, n/4, ..., 1
            h = n / 2;
            while (h >= 1) {
                gaps.add(h);
                h /= 2;
            }
            break;

        case KNUTH:
            // Knuth's sequence: h = (3^k - 1) / 2
            h = 1;
            while (h < n / 3) {
                h = 3 * h + 1;
            }
            while (h >= 1) {
                gaps.add(h);
                h /= 3;
            }
            break;
```

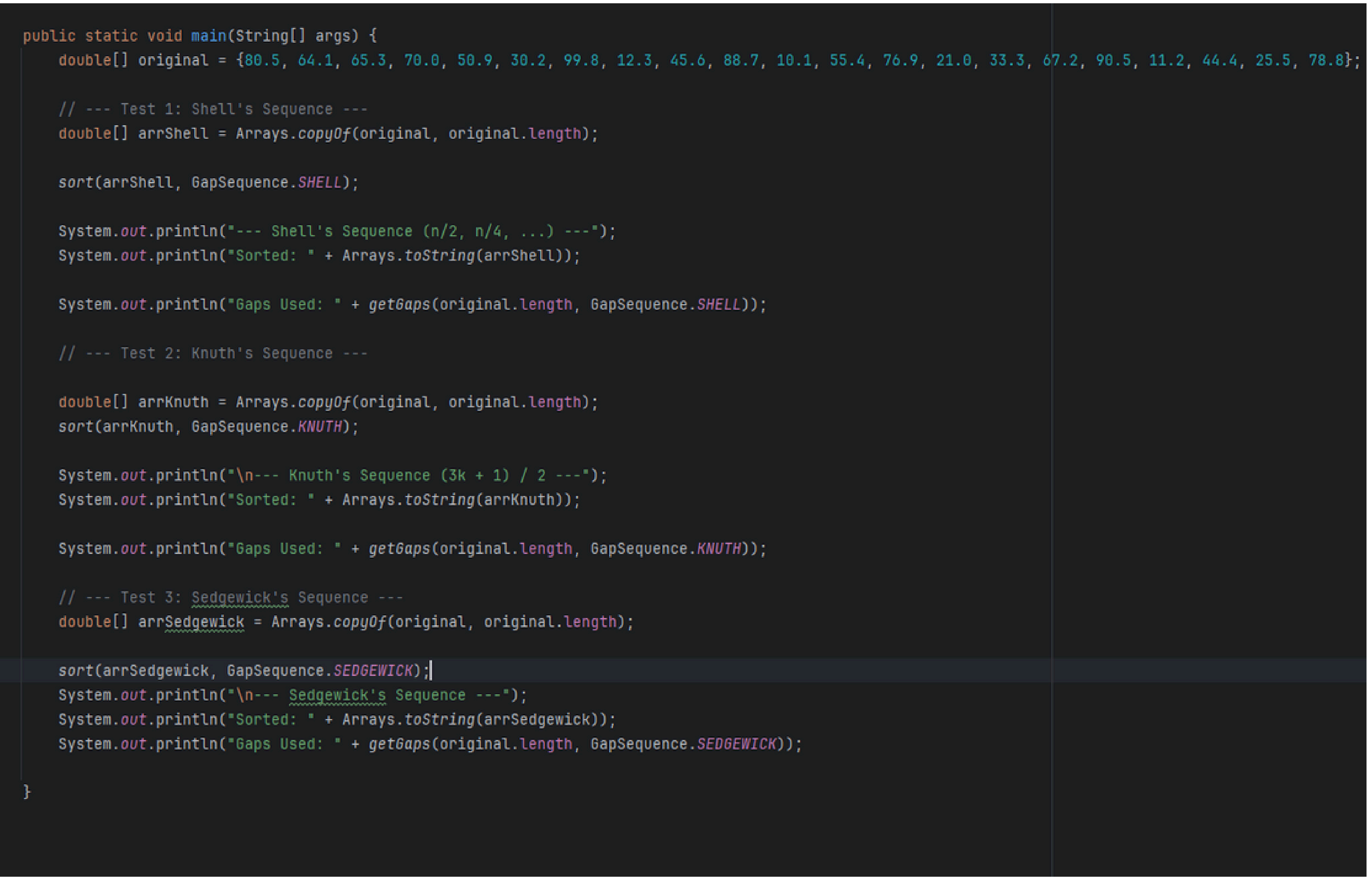


```java
        case SEDGEWICK:
            // Sedgewick's gap sequence: Interleaving of two formulas.

            int k = 0;
            h = 1;


            while (h < n) {
                gaps.add(h);
                k++;
                // i = k / 2 (integer division)
                int i = k / 2;

                if (k % 2 == 0) {
                    // For even     h=9*4^i - 9*2^i + 1
                    h = (int) (9 * (Math.pow(4, i) - Math.pow(2, i)) + 1);}
                else {
                    //for odd  h=4^(i+1) + 3*2^i + 1

                    h = (int) (Math.pow(4, i + 1) + 3 * Math.pow(2, i) + 1);
                }
            }
            // reverse array
            java.util.Collections.reverse(gaps);
            break;
    }
```

3)This method performs the Shell Sort algorithm, sorting the given array using the specified gap sequence strategy.



```
public static void sort(double[] arr, GapSequence sequence) {  3 usages
    if (arr == null || arr.length < 2) return;

    List<Integer> gaps = getGaps(arr.length, sequence);

    // Iterate in descending order
    for (int gap : gaps) {

        for (int i = gap; i < arr.length; i++) {
            double current = arr[i];
            int Prev_index = i;

            // Compare element arr[i] with the element gap positions behind it
            while (Prev_index >= gap && arr[Prev_index - gap] > current) {
                arr[Prev_index] = arr[Prev_index - gap];
                Prev_index -= gap;
            }
            // Place the current element into its correct position
            arr[Prev_index] = current;
        }
    }
}
```

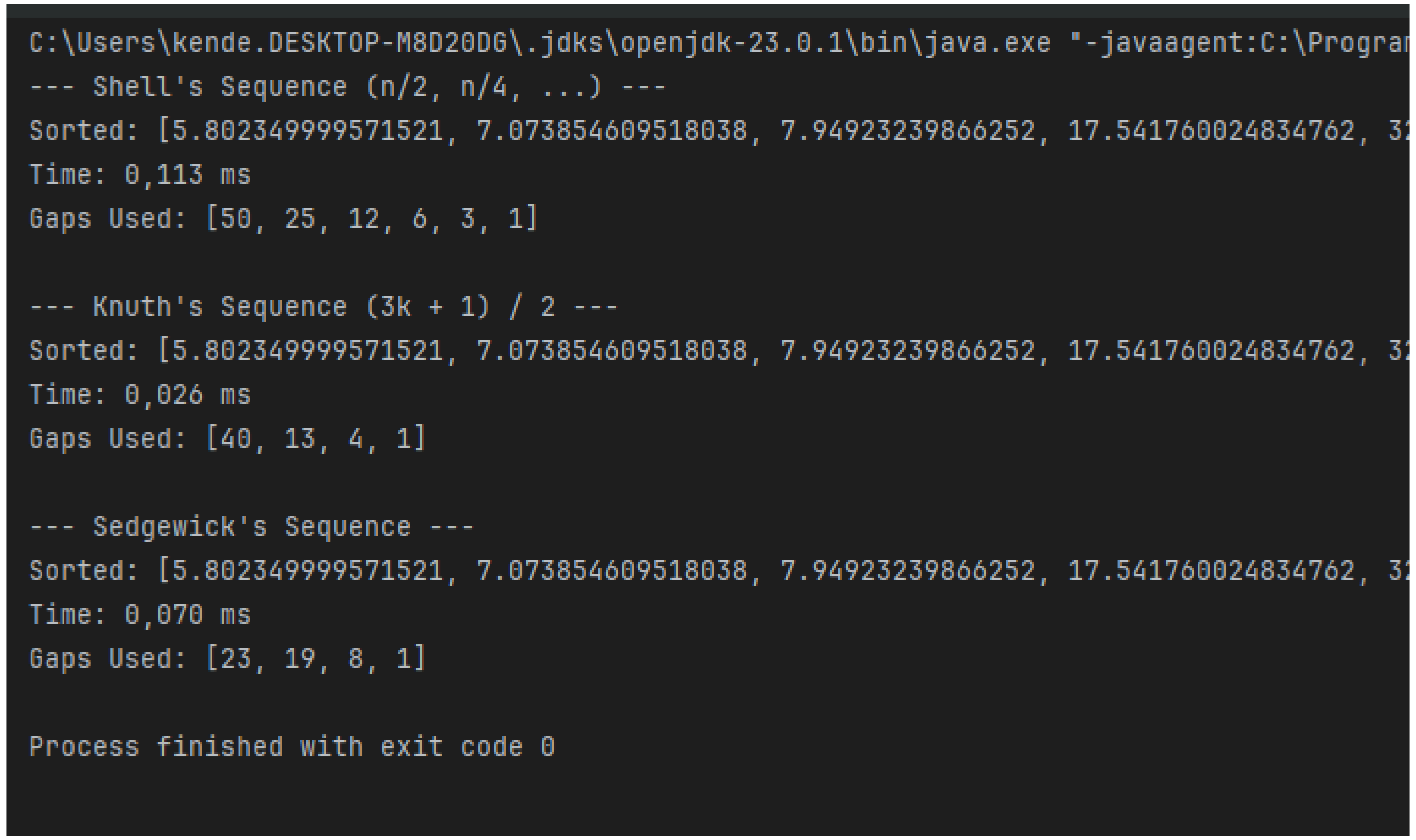


```java
public static void main(String[] args) {
    double[] original = {80.5, 64.1, 65.3, 70.0, 50.9, 30.2, 99.8, 12.3, 45.6, 88.7, 10.1, 55.4, 76.9, 21.0, 33.3, 67.2, 90.5, 11.2, 44.4, 25.5, 78.8};

    // --- Test 1: Shell's Sequence ---
    double[] arrShell = Arrays.copyOf(original, original.length);

    sort(arrShell, GapSequence.SHELL);

    System.out.println("--- Shell's Sequence (n/2, n/4, ...) ---");
    System.out.println("Sorted: " + Arrays.toString(arrShell));

    System.out.println("Gaps Used: " + getGaps(original.length, GapSequence.SHELL));

    // --- Test 2: Knuth's Sequence ---

    double[] arrKnuth = Arrays.copyOf(original, original.length);
    sort(arrKnuth, GapSequence.KNUTH);

    System.out.println("\n--- Knuth's Sequence (3k + 1) / 2 ---");
    System.out.println("Sorted: " + Arrays.toString(arrKnuth));

    System.out.println("Gaps Used: " + getGaps(original.length, GapSequence.KNUTH));

    // --- Test 3: Sedgewick's Sequence ---
    double[] arrSedgewick = Arrays.copyOf(original, original.length);

    sort(arrSedgewick, GapSequence.SEDGEWICK);
    System.out.println("\n--- Sedgewick's Sequence ---");
    System.out.println("Sorted: " + Arrays.toString(arrSedgewick));
    System.out.println("Gaps Used: " + getGaps(original.length, GapSequence.SEDGEWICK));

}
```

4) this is main method. It calls the other methods to do their part of job(GapSequence, getGaps, sort). And then prints the results

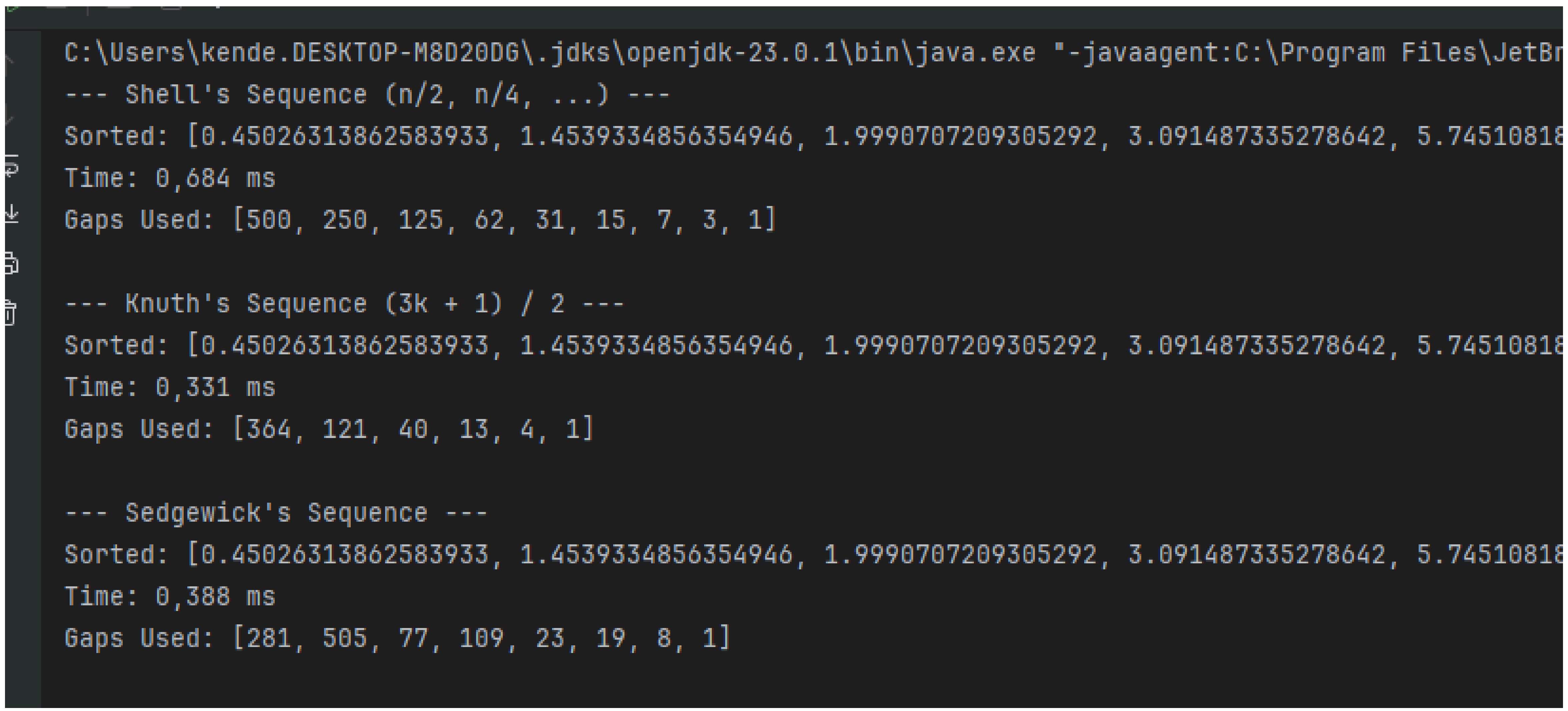
This is the result of Aktilek’s code

perfomanse 100



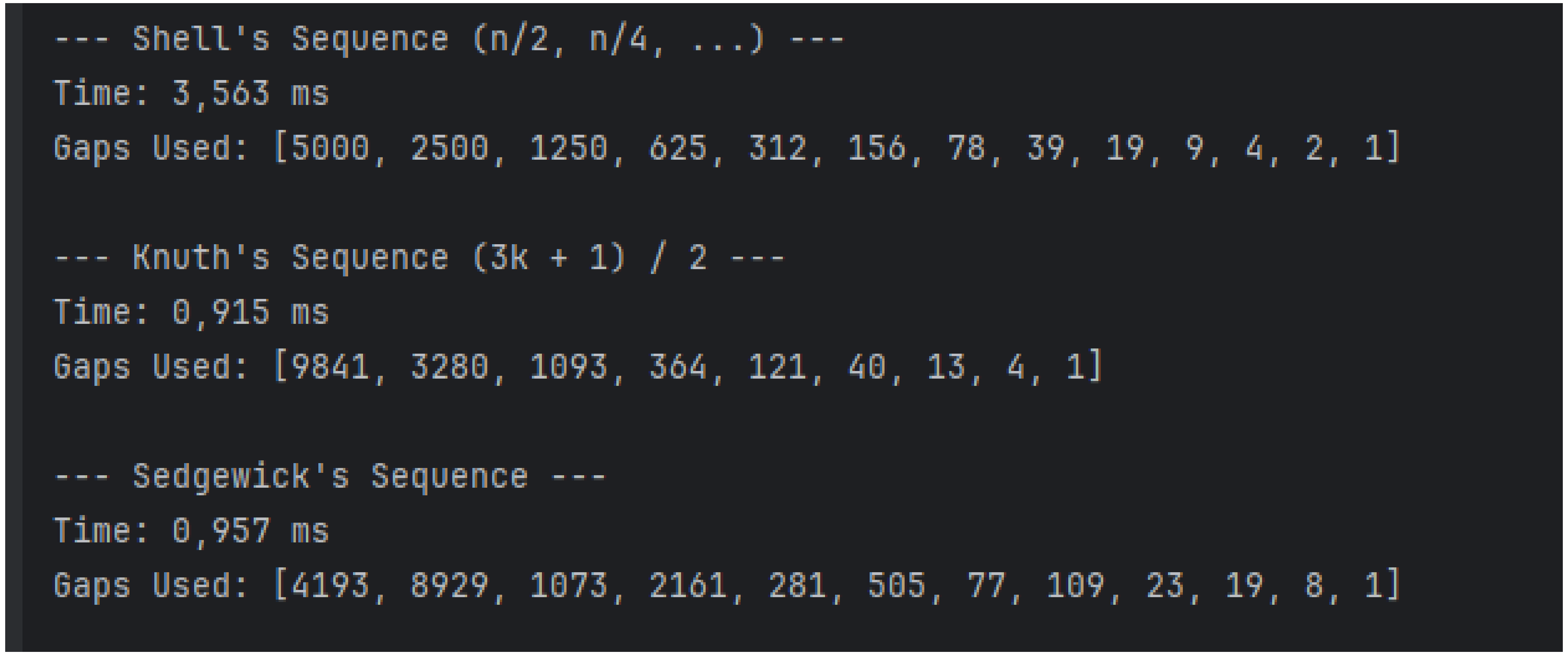
```
C:\Users\kende.DESKTOP-M8D20DG\.jdks\openjdk-23.0.1\bin\java.exe "-javaagent:C:\Progra
--- Shell's Sequence (n/2, n/4, ...) ---
Sorted: [5.802349999571521, 7.073854609518038, 7.94923239866252, 17.541760024834762, 3
Time: 0,113 ms
Gaps Used: [50, 25, 12, 6, 3, 1]

--- Knuth's Sequence (3k + 1) / 2 ---
Sorted: [5.802349999571521, 7.073854609518038, 7.94923239866252, 17.541760024834762, 3
Time: 0,026 ms
Gaps Used: [40, 13, 4, 1]

--- Sedgewick's Sequence ---
Sorted: [5.802349999571521, 7.073854609518038, 7.94923239866252, 17.541760024834762, 3
Time: 0,070 ms
Gaps Used: [23, 19, 8, 1]

Process finished with exit code 0
```

perfomanse 1000



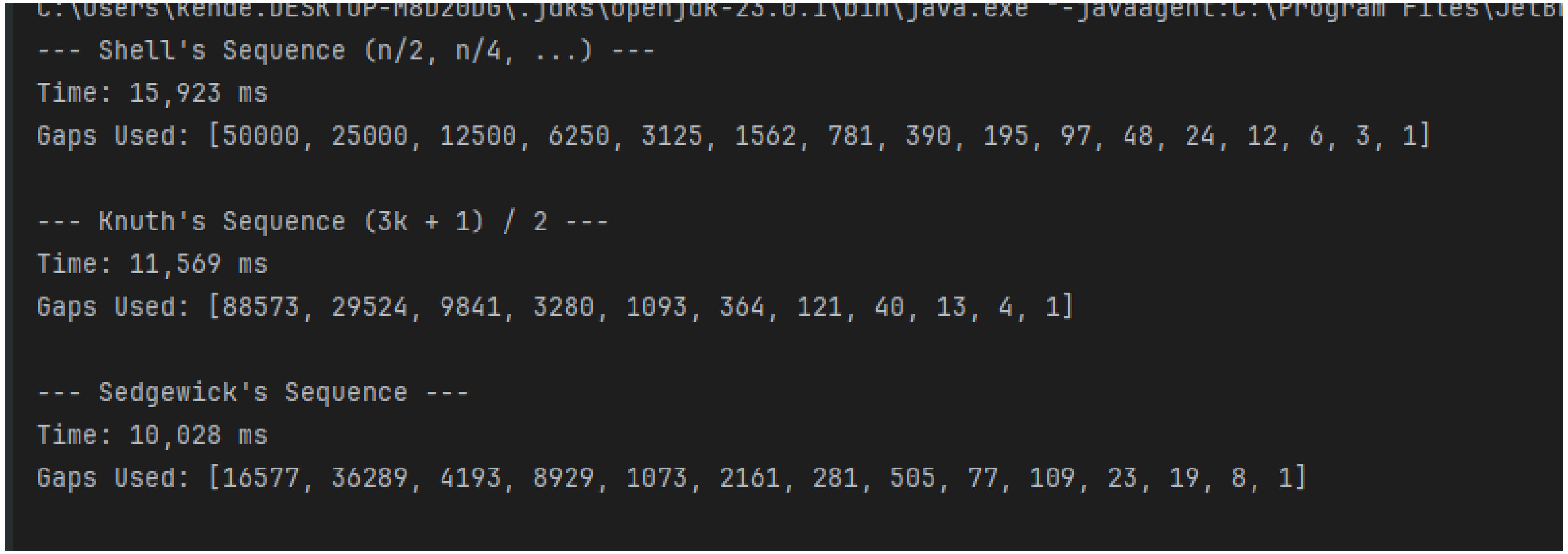
```
C:\Users\kende.DESKTOP-M8D20DG\.jdks\openjdk-23.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBr
--- Shell's Sequence (n/2, n/4, ...) ---
Sorted: [0.45026313862583933, 1.4539334856354946, 1.9990707209305292, 3.091487335278642, 5.74510818
Time: 0,684 ms
Gaps Used: [500, 250, 125, 62, 31, 15, 7, 3, 1]

--- Knuth's Sequence (3k + 1) / 2 ---
Sorted: [0.45026313862583933, 1.4539334856354946, 1.9990707209305292, 3.091487335278642, 5.74510818
Time: 0,331 ms
Gaps Used: [364, 121, 40, 13, 4, 1]

--- Sedgewick's Sequence ---
Sorted: [0.45026313862583933, 1.4539334856354946, 1.9990707209305292, 3.091487335278642, 5.74510818
Time: 0,388 ms
Gaps Used: [281, 505, 77, 109, 23, 19, 8, 1]
```

perfomanse 10000



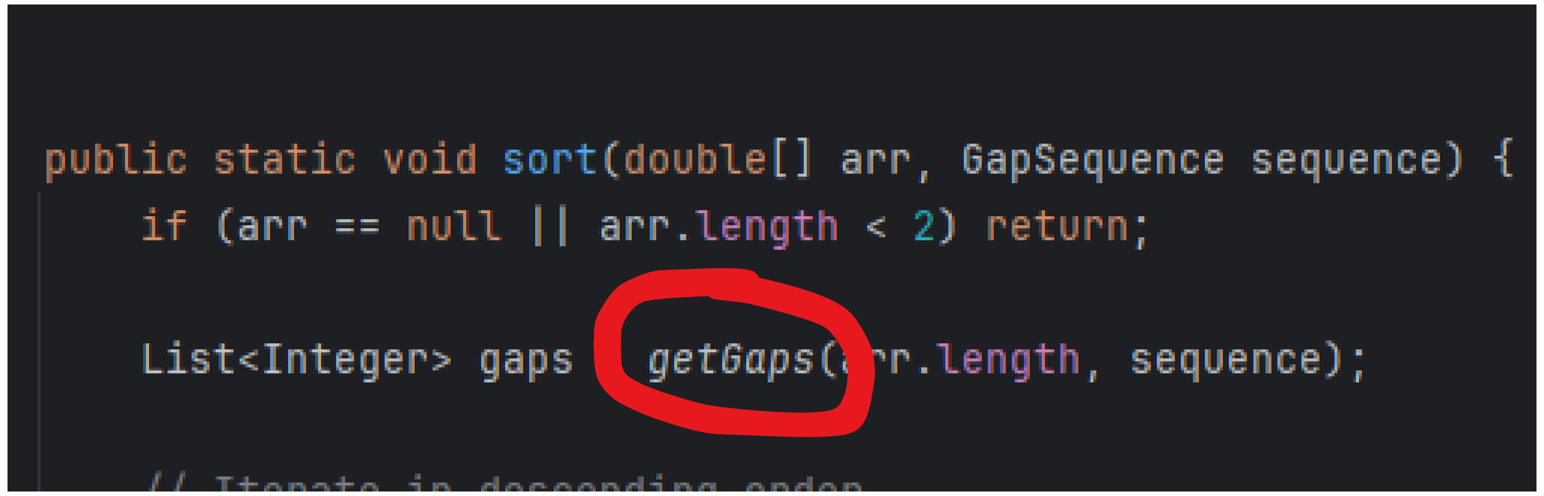
```
--- Shell's Sequence (n/2, n/4, ...) ---
Time: 3,563 ms
Gaps Used: [5000, 2500, 1250, 625, 312, 156, 78, 39, 19, 9, 4, 2, 1]

--- Knuth's Sequence (3k + 1) / 2 ---
Time: 0,915 ms
Gaps Used: [9841, 3280, 1093, 364, 121, 40, 13, 4, 1]

--- Sedgewick's Sequence ---
Time: 0,957 ms
Gaps Used: [4193, 8929, 1073, 2161, 281, 505, 77, 109, 23, 19, 8, 1]
```

perfomanse 100000

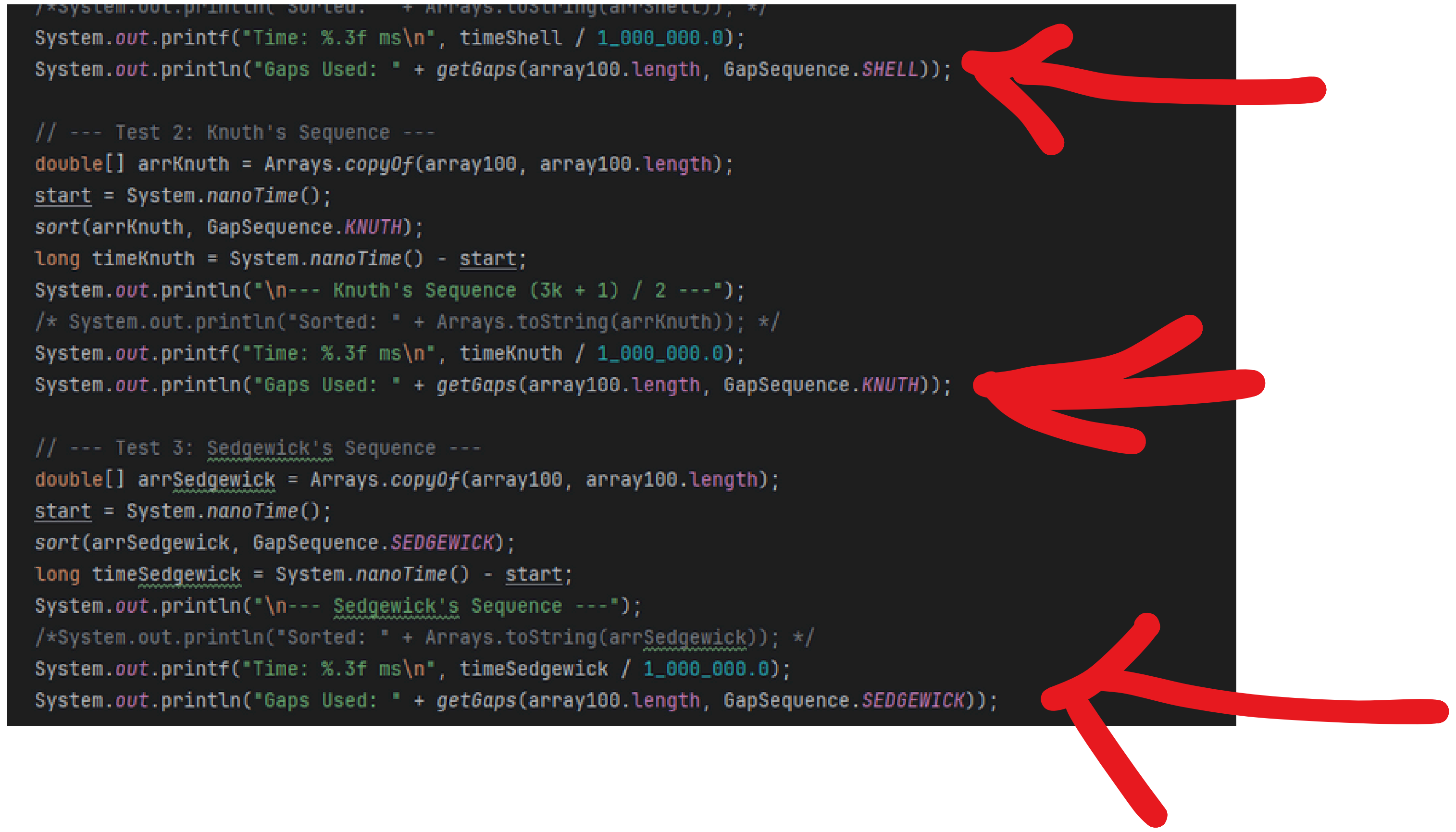


```
C:\Users\Rende.DESKTOP-M8D20DG\.jdks\openjdk-23.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBr
--- Shell's Sequence (n/2, n/4, ...) ---
Time: 15,923 ms
Gaps Used: [50000, 25000, 12500, 6250, 3125, 1562, 781, 390, 195, 97, 48, 24, 12, 6, 3, 1]

--- Knuth's Sequence (3k + 1) / 2 ---
Time: 11,569 ms
Gaps Used: [88573, 29524, 9841, 3280, 1093, 364, 121, 40, 13, 4, 1]

--- Sedgewick's Sequence ---
Time: 10,028 ms
Gaps Used: [16577, 36289, 4193, 8929, 1073, 2161, 281, 505, 77, 109, 23, 19, 8, 1]
```

I found out that there is 2 times called the getGaps

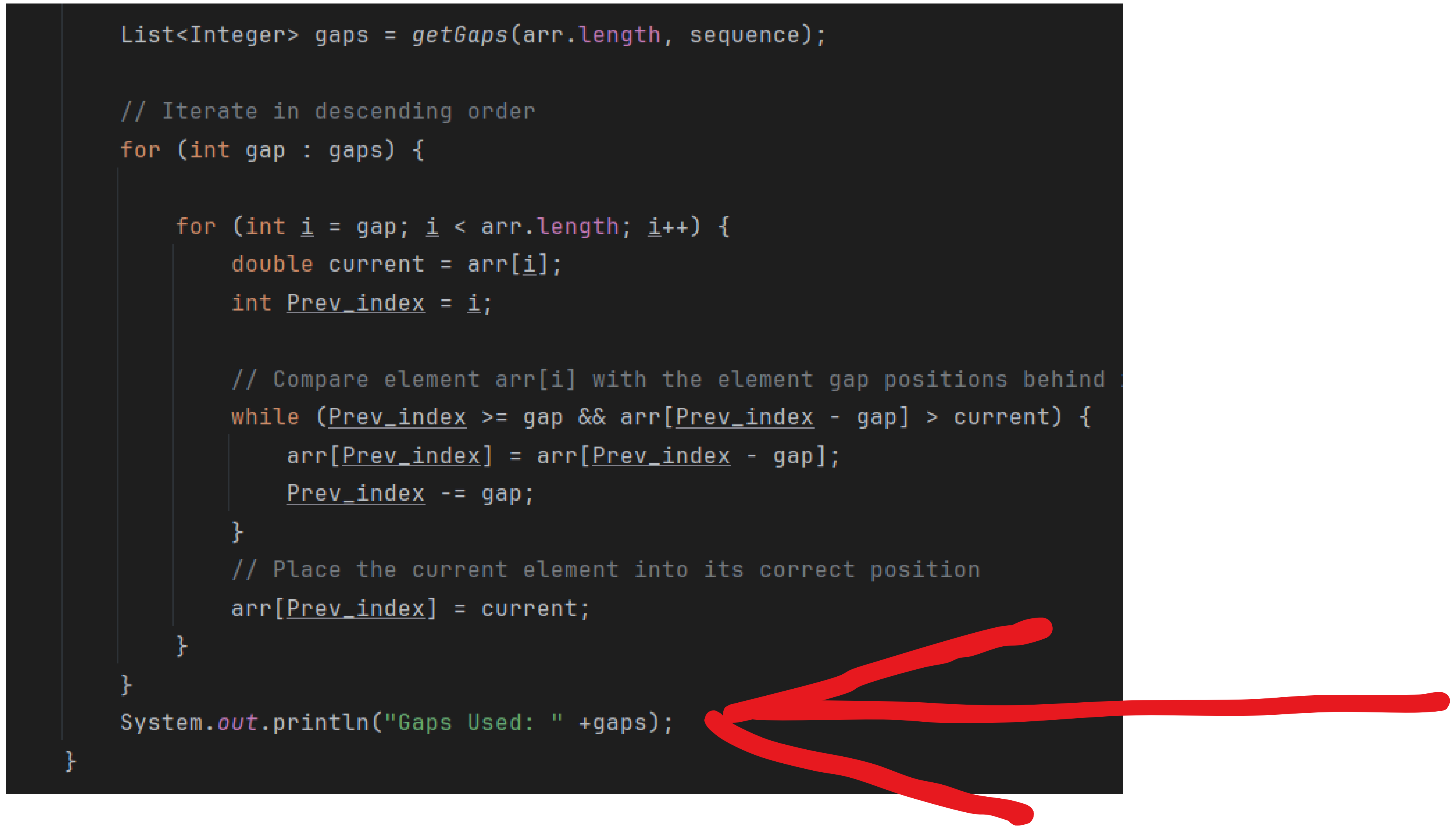


```
public static void sort(double[] arr, GapSequence sequence) {
    if (arr == null || arr.length < 2) return;

    List<Integer> gaps = getGaps(arr.length, sequence);

    // Iterate in descending order
```



```
/*System.out.println("Sorted: " + Arrays.toString(arrShell)); */
System.out.printf("Time: %.3f ms\n", timeShell / 1_000_000.0);
System.out.println("Gaps Used: " + getGaps(array100.length, GapSequence.SHELL));

// --- Test 2: Knuth's Sequence ---
double[] arrKnuth = Arrays.copyOf(array100, array100.length);
start = System.nanoTime();
sort(arrKnuth, GapSequence.KNUTH);
long timeKnuth = System.nanoTime() - start;
System.out.println("\n--- Knuth's Sequence (3k + 1) / 2 ---");
/* System.out.println("Sorted: " + Arrays.toString(arrKnuth)); */
System.out.printf("Time: %.3f ms\n", timeKnuth / 1_000_000.0);
System.out.println("Gaps Used: " + getGaps(array100.length, GapSequence.KNUTH));

// --- Test 3: Sedgewick's Sequence ---
double[] arrSedgewick = Arrays.copyOf(array100, array100.length);
start = System.nanoTime();
sort(arrSedgewick, GapSequence.SEDGEWICK);
long timeSedgewick = System.nanoTime() - start;
System.out.println("\n--- Sedgewick's Sequence ---");
/*System.out.println("Sorted: " + Arrays.toString(arrSedgewick)); */
System.out.printf("Time: %.3f ms\n", timeSedgewick / 1_000_000.0);
System.out.println("Gaps Used: " + getGaps(array100.length, GapSequence.SEDGEWICK));
```

I have removed them and put “print “inside sort method to be efficient



```
    List<Integer> gaps = getGaps(arr.length, sequence);

    // Iterate in descending order
    for (int gap : gaps) {

        for (int i = gap; i < arr.length; i++) {
            double current = arr[i];
            int Prev_index = i;

            // Compare element arr[i] with the element gap positions behind
            while (Prev_index >= gap && arr[Prev_index - gap] > current) {
                arr[Prev_index] = arr[Prev_index - gap];
                Prev_index -= gap;
            }
            // Place the current element into its correct position
            arr[Prev_index] = current;
        }
    }
    System.out.println("Gaps Used: " +gaps);
}
```

The worst-case **time complexities** for the gap sequences of the code are:

SHELL: O(n2)

KNUTH: O(n3/2)

SEDGEWICK: O(n4/3)

REPORT

The S**hell Sort** algorithm is an extension of the simple insertion sort that allows the exchange of far-apart elements. It was invented by Donald Shell in 1959 to improve sorting performance by introducing a sequence of gaps. Instead of comparing adjacent elements (as in insertion sort), Shell Sort starts by comparing elements far apart, progressively reducing the gap until it becomes one. When the gap equals 1, the algorithm performs a final insertion sort pass, ensuring that the array is fully sorted.

Key Idea

Shell Sort sorts subarrays formed by taking every gapth element

Larger gaps move elements long distances, reducing disorder quickly.

Smaller gaps finalize local ordering efficiently.

Gap Sequences Used

Sequence Formula Example **Complexity** (Worst Case)

Shell n/2, n/4, ..., 1 50, 25, 12, 6, 3, 1 **O(n^2)**

Knuth h = 3h + 1 1, 4, 13, 40, ... **O(n^(3/2))**

Sedgewick h = 9*4^i - 9*2^i + 1 or h = 4^(i+1) - 3*2^(i+1) + 1 1, 5, 19, 41, 109, ... **O(n^(4/3)**)

Aktilek’s code correctly implements Shell Sort using three different gap sequences — Shell, Knuth, and Sedgewick — and includes benchmarking to measure performance. The implementation is clear, functional, and modular, but it could be slightly optimized by reducing redundant operations such as multiple calls to getGaps() in main(). Overall, it’s a solid and well-structured version that effectively demonstrates the effect of different gap sequences on sorting efficiency.

Thank YOU for your attention