thing

```
package sorts.hybrid;

import main.ArrayVisualizer;
import sorts.templates.Sort;

/*
 *
MIT License

Copyright (c) 2020 yuji, implemented by aphitorite, edited by dani_dlg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
 *
 */


final public class YujisBufferedMergeSort2 extends Sort {


    public YujisBufferedMergeSort2(ArrayVisualizer arrayVisualizer) {
        super(arrayVisualizer);

        this.setSortListName("Optimized A/Y Sort");
        this.setRunAllSortsName("Optimized Andrey/Yuji Sort");
        this.setRunSortName("Optimized Andrey/Yuji Sort");
        this.setCategory("Hybrid Sorts");
        this.setComparisonBased(true);
        this.setBucketSort(false);
        this.setRadixSort(false);
        this.setUnreasonablySlow(false);
        this.setUnreasonableLimit(0);
        this.setBogoSort(false);
    }

    public static int ceilLog(int n) {
        int i;
        for(i = 0; (1 << i) < n; i++);
        return i;
    }

    private void multiSwap(int[] array, int a, int b, int len) {
        for(int i = 0; i < len; i++)
            Writes.swap(array, a+i, b+i, 1, true, false);
    }

    private void insertTo(int[] array, int a, int b) {
        Highlights.clearMark(2);
        int temp = array[a];
        while(a > b) Writes.write(array, a, array[(a--)-1], 0.5, true, false);
        Writes.write(array, b, temp, 0.5, true, false);
```

```
}

private int binarySearch(int[] array, int start, int end, int value, boolean left) {
    int a = start, b = end;

    while(a < b) {
        int m = a+(b-a)/2;
        boolean comp;

        if(left) comp = Reads.compareValues(value, array[m]) <= 0;
        else     comp = Reads.compareValues(value, array[m]) < 0;

        if(comp) b = m;
        else     a = m+1;
    }

    return a;
}

private void binaryInsertion(int[] array, int a, int b) {
    for(int i = a+1; i < b; i++)
        this.insertTo(array, i, this.binarySearch(array, a, i, array[i], false));
}


private void mergeWithBufStatic(int[] array, int a, int m, int b, int p, boolean useBinarySearch) {
    int i = 0, j = m, k = a;

    if(useBinarySearch) {
        while(i < m-a && j < b) {
            if(Reads.compareValues(array[j], array[p+i]) == -1) {
                int q = this.binarySearch(array, j, b, array[p+i], true);
                while(j < q) Writes.swap(array, k++, j++, 1, true, false);
            }
            Writes.swap(array, k++, p+(i++), 1, true, false);
        }
        while(i < m-a) {
            Writes.swap(array, k++, p+(i++), 1, true, false);
        }
    }
    else {
        while(i < m-a && j < b) {
            if(Reads.compareValues(array[p+i], array[j]) <= 0) {
                Writes.swap(array, k++, p+(i++), 1, true, false);
            }
            else {
                Writes.swap(array, k++, j++, 1, true, false);
            }
        }
        while(i < m-a) {
            Writes.swap(array, k++, p+(i++), 1, true, false);
        }
    }
}

private int merge(int[] array, int a, int m, int b, int p) {
    int i = a, j = m;
    while(i < m && j < b) {
        if(Reads.compareIndices(array, i, j, 0, false) <= 0)
            Writes.swap(array, p++, i++, 1, true, false);
        else
            Writes.swap(array, p++, j++, 1, true, false);
    }
    int leftover = 0;
    while(i < m) {
        Writes.swap(array, p++, i++, 1, true, false);

    }
    while(j < b) {
        Writes.swap(array, p++, j++, 1, true, false);
        leftover++;
    }
    return leftover;
}
```

```
    private void mergeSort(int[] array, int a, int p, int length) {
        int i, j = 16, pos, ceilLog = ceilLog(length);

        if(length > 16 && (ceilLog & 1) == 1)
            pos = p;
        else
            pos = a;

        for(i = pos; i+16 <= pos+length; i+=16)
            this.binaryInsertion(array, i, i+16);
        this.binaryInsertion(array, i, pos+length);

        int next = pos, posNext;
        while(j < length) {
            pos = next;
            next ^= a ^ p;
            posNext = next;

            for(i = pos; i+2*j <= pos+length; i+=2*j, posNext+=2*j)
                this.merge(array, i, i+j, i+2*j, posNext);
            if(i + j < pos+length)
                this.merge(array, i, i+j, pos+length, posNext);
            else
                while(i < pos+length) Writes.swap(array, i++, posNext++, 1, true, false);

            j *= 2;
        }
    }

    private void bufferedMerge(int[] array, int a, int b) {
        if(b-a <= 16) {
            this.binaryInsertion(array, a, b);
            return;
        }

        int m = (a+b+1)/2;
        this.mergeSort(array, m, 2*m-b, b-m);

        int n = (a+m+1)/2;
        //The 16 means that it will recursively sort 1/16 of the array. It's interesting to change that number and see h
        int limit = (b-a)/16;
        while(m-a > limit) {
            this.mergeSort(array, 2*n-m, n, m-n);
            this.mergeWithBufStatic(array, n, m, b, 2*n-m, (b-m)/(m-n) >= ceilLog(n-a));
            m = n;
            n = (a+m+1)/2;
        }

        // the same as Andreysort's buffer redistribution
        this.bufferedMerge(array, a, m);
        this.multiSwap(array, a, b-(m-a), m-a);
        int s = this.merge(array, m, b-(m-a), b, a);
        this.bufferedMerge(array, b-(m-a)-s, b);

    }


    @Override
    public void runSort(int[] array, int length, int bucketCount) {
        this.bufferedMerge(array, 0, length);
    }
}
```