INFO6205 Final Project - Fall 2021

Team members

Yu An NUID: 001523003

Lan Gao NUID: 001568670

Requirements

This task is to implement **MSD radix sort** for a natural language which uses Unicode characters.
You may choose your own language or (Simplified) Chinese. Additionally, you will complete a literature survey of relevant papers and you will compare your method with **Timsort**, **Dual-pivot Quicksort**, **Huskysort**, and **LSD radix sort**.

Task

In the implemention of this task, we chose to compare all the methods in sorting (Simplified) Chinese.

MSD radix sort

In the process of implementing msd along the lines of LSD, some problems arose. The algorithm of msd is implemented recursively, splitting the array into smaller arrays after each sorting, and sorting smaller arrays next time. According to the above idea of sorting only one character of Hanzi Pinyin each time, each layer of Hanzi characters will add up to 7 more layers(mix to 6 characters and 1 number to represent tone) of sorting, and the time cost will directly rise exponentially.

In accomplishing, we converted the Chinese character array into a two-dimensional array before sorting, and stored the Chinese characters and the corresponding pinyin string. Each sort directly operates on the pinyin string, so that the whole sort is basically converted to English character sort, radix can also use the 256 ASCII code table. Time is reduced to linearithmic.

```java
import net.sourceforge.pinyin4j.PinyinHelper;
import sort.elementary.InsertionSortMSD;

/**
 * Class to implement Most significant digit string sort (a radix sort).
 */
public class MSDStringSort {

    /**
     * Sort an array of Strings using MSDStringSort.
     *
     * @param a the array to be sorted.
     */
    public static void sort(String[] a) {
        int n = a.length;
        aux = new String[n][2];
        String[][] arywithPinyin = new String[n][2];
        for (int i =0 ;i<a.length;i++){
            StringBuilder sb = new StringBuilder();
            for (char c:a[i].toCharArray()){
                String[] pinyin = PinyinHelper.toHanyuPinyinStringArray(c);
                sb.append(pinyin[0]);
            }
            arywithPinyin[i][0] = a[i];
            arywithPinyin[i][1] = sb.toString();
        }
        sort(arywithPinyin, lo: 0, hi: n-1, d: 0);
        for (int i=0;i<arywithPinyin.length;i++){
            a[i]=arywithPinyin[i][0];
        }
    }
```

```java
/**
 * Sort from a[lo] to a[hi] (exclusive), ignoring the first d characters of each String.
 * This method is recursive.
 *
 * @param a the array to be sorted.
 * @param lo the low index.
 * @param hi the high index (one above the highest actually processed).
 * @param d the number of characters in each String to be skipped.
 */
private static void sort(String[][] a, int lo, int hi, int d) {
    if (hi <= lo + cutoff) InsertionSortMSD.sort(a, lo, hi, d);
    else {
        int[] count = new int[radix + 2];        // Compute frequency counts.
        for (int i = lo; i <= hi; i++)
            count[charAt(a[i][1], d) + 2]++;
        for (int r = 0; r < radix + 1; r++)      // Transform counts to indices.
            count[r + 1] += count[r];
        for (int i = lo; i <= hi; i++)     // Distribute.
            aux[count[charAt(a[i][1], d) + 1]++] = a[i];
        // Copy back.
        if (hi - lo >= 0) System.arraycopy(aux, srcPos: 0, a, lo, length: hi - lo + 1);

        // Recursively sort for each character value.
        // TO BE IMPLEMENTED
        for(int r =0;r<radix;r++){
            sort(a, lo: lo+count[r], hi: lo+count[r+1]-1, d: d+1);
        }
        //Collator c = Collator.getInstance(Locale.CHINA);
    }
}
```

```java
    private static int charAt(String s, int charPosition) {
        if (charPosition < s.length()){

            return s.charAt(charPosition);
        }else{
            return -1;
        }
    }

    private static final int radix = 256;
    private static final int cutoff = 0;
    private static String[][] aux;       // auxiliary array for distribution
}
```

LSD radix sort

LSD radix sort sorts characters in order from the rightmost character to the left through an ordered alphabet. The first idea for sorting Chinese characters is to find a coding table that exists in pinyin order, and to obtain the coding values of Chinese characters that can be sorted normally according to radix.
After searching, we found that GBK basically meets the condition - it is encoded by two bytes, ranging from 8140 to FEFE, and contains 21003 Chinese characters in total. However, during the experiment, we found that the 3755 common Chinese characters in the GBK code are in pinyin order, but the relationship between the second-level Chinese characters and the first-level Chinese characters is not in pinyin order (the first-level Chinese characters and the second-level Chinese characters are in different coding areas).
To solve the problem, the first method is to sort the GBK table by collator, then store the GBK characters into a searchable table, and create a radix array of the same size as the table. However, this solution will take an exaggerated amount of time, and when raidx becomes 65536 (unicode) encoding, it will take thousands of times longer, so if we use the GBK encoding table, it will also take thousands of times longer.
Therefore, we need to change the way of thinking, directly through pinyin4j to get the pinyin and tone of Chinese characters, in the counting stage of each sorting, the characters will be converted into pinyin (the format of pinyin is lowercase pinyin plus tone), and then the original LSD algorithm will have one more layer of loops to deal with the sorting of individual pinyin.

```
/**
 * charAsciiVal method returns ASCII value of particular character in a String.
 *
 * @param str          String input for which ASCII Value need to be found
 * @param charPosition Character position of which ASCII value needs to be found. If character
 *                     doesn't exist then ASCII value of null i.e. 0 is returned
 * @return int Returns ASCII value
 */
private int charAsciiVal(String str, int charPosition,int charPinyinPosition) {
    if (charPosition >= str.length()) {
        return -1;
    }
    String[] pinyinStringArray = PinyinHelper.toHanyuPinyinStringArray(str.charAt(charPosition));
    if (charPinyinPosition>=pinyinStringArray[0].length()){
        return -1;
    }

    return pinyinStringArray[0].charAt(charPinyinPosition);
}
```

```java
/**
 * sort method is implementation of LSD String sort algorithm.
 *
 * @param strArr It contains an array of String on which LSD sort needs to be performed
 * @param from   This is the starting index from which sorting operation will begin
 * @param to     This is the ending index up until which sorting operation will be continued
 */
public void sort(String[] strArr, int from, int to) {
    int maxLength = findMaxLength(strArr);
    for (int i = maxLength - 1; i >= 0; i--){
        int maxPinyinLength = findMaxPinyinLength(strArr,i);
        for (int j = maxPinyinLength-1;j>=0;j--)
            charSort(strArr, i, from, to,j);
    }

}

private int findMaxPinyinLength(String[] strArr, int charPosition) {
    int maxPinyinLength = Integer.MIN_VALUE;
    for (String str : strArr){
        if(charPosition>=str.length()){
            continue;
        }
        String[] pinyinStringArray = PinyinHelper.toHanyuPinyinStringArray(str.charAt(charPosition));
        maxPinyinLength= Math.max(pinyinStringArray[0].length(),maxPinyinLength);
    }
    return maxPinyinLength;
}
```

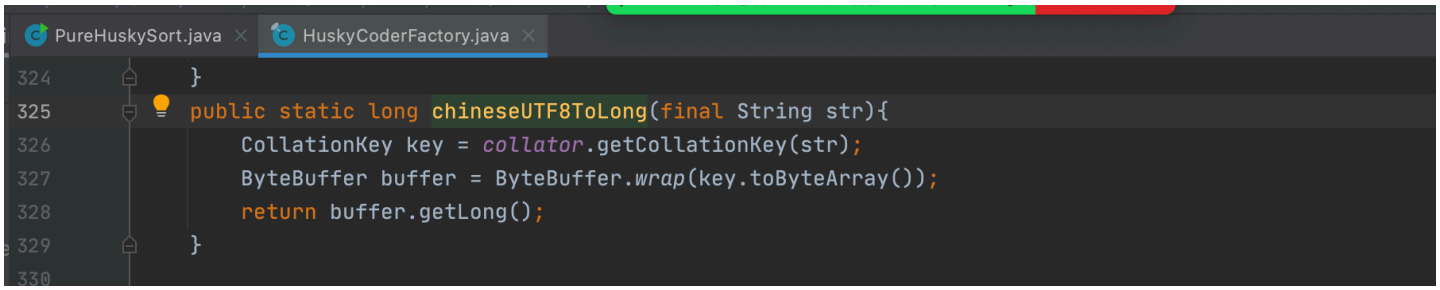
Huskysort

In Huskysort, we found it could convert the strings into long which is comparable directly from the original codes and paper. Therefore, we only need to find a tool that can encode Chinese, and the coding is practiced in the order of pinyin, and we can obtain comparable values.

So, through the collator class, we got comparionKey of each Chinese characters. Then obtained the Byte arrays corresponding to the key value through toByteArray(), and finally converted the long value to complete the work of the encoder.

But, during the experiment, "陈旻玥" and ""陈旻旻" have the same value after encoded, so this is not perfect. We had to set the value of perfect of coder to false in the coderfactiory class. Then used insertion sort or arrays.sort to do the second sort.

In addition, the byte values of the same encoding value are actually different. There is something left to study --- if it's the hidden bug of byte conversion to long type that leads to imperfect encoding.



Timsort

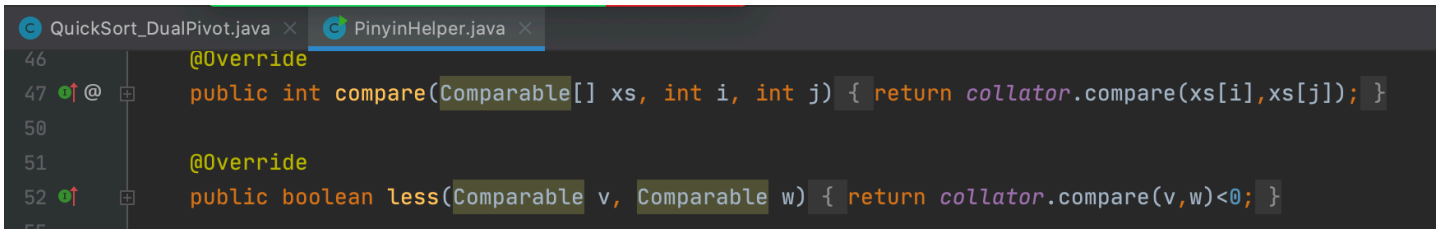
Since Timsort is already allowed to use collator to sort Chinese words, so we only need to provide collator.getInstance(Local.China) in the function.

```
public void sort(X[] xs, int from, int to) { Arrays.sort(xs, from, to, Collator.getInstance(Locale.CHINA)); }
```

Dual-pivot Quicksort

For quicksort, the only thing needed to know is the size relationship of the strings to sort them. With the collator class provided by java we can compare Chinese strings, in this case, for Quicksort, there is no difference between comparing Chinese and comparing numbers.

So we built the PinyinHelper class which extended from the helper class to accomplish the comparison of pinyin.



QuickSort_DualPivot.java | PinyinHelper.java

```
103         }
104
105         @Override
106 @       public boolean swapStableConditional(Comparable[] xs, int i) {
107             Comparable x = xs[i];
108             Comparable y = xs[i-1];
109             boolean result = collator.compare(x,y)< 0;
110             if (result) {
111                 xs[i] = y;
112                 xs[i - 1] = x;
113             }
114             return result;
115         }
```

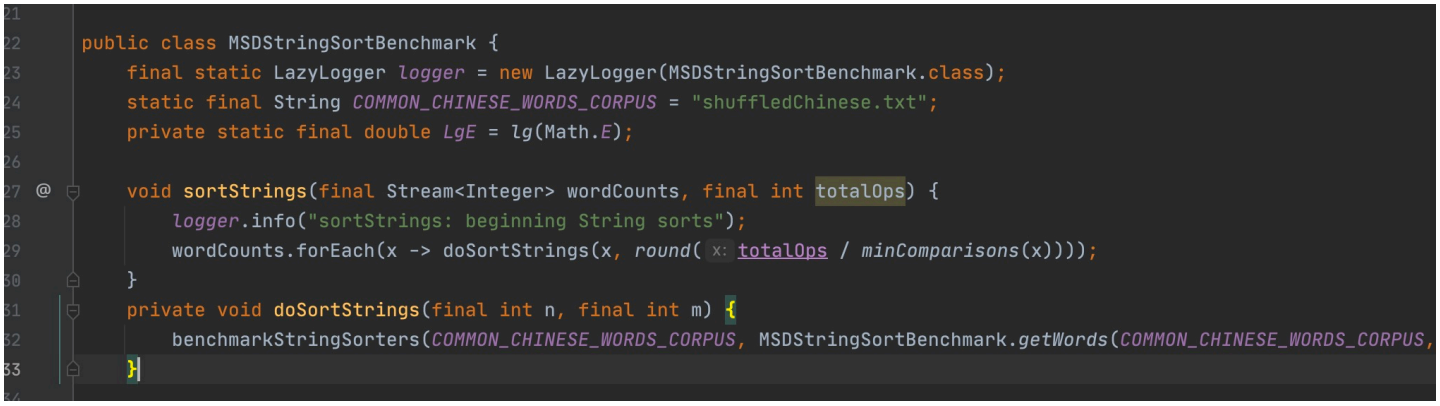
QuickSort_DualPivot.java | PinyinHelper.java

```
198     }                                                                    ⚠ 28
199 @   public static boolean isSorted(String[] xs){
200         for (int i=1;i<xs.length;i++) {
201             if (collator.compare(xs[i], xs[i-1]) < 0){
202                 return false;
203
204             }
205         }
206         return true;
207     }
208 @   public static boolean isSorted(String[] xs,String type){
209         int n = xs.length;
210         String[][] arywithPinyin = new String[n][2];
211         for (int i =0 ;i<xs.length;i++){
212             StringBuilder sb = new StringBuilder();
213             for (char c:xs[i].toCharArray()){
214                 String[] pinyin = net.sourceforge.pinyin4j.PinyinHelper.toHanyuPinyinStringArray(c);
215                 sb.append(pinyin[0]);
216             }
217             arywithPinyin[i][0] = xs[i];
218             arywithPinyin[i][1] = sb.toString();
219         }
220         for (int i=1;i<arywithPinyin.length;i++){
221             if (arywithPinyin[i][1].compareTo(arywithPinyin[i-1][1])<0){
222                 return false;
223             }
224         }
225         return true;
226     }
227
```

Benchmark

Basde on Assignment2 and the Benchmark class in Huskysort we built the benchmark class and benchmark test class for each sort methods.

Use MSD radix sort as an example:



```
public class MSDStringSortBenchmark {
    final static LazyLogger logger = new LazyLogger(MSDStringSortBenchmark.class);
    static final String COMMON_CHINESE_WORDS_CORPUS = "shuffledChinese.txt";
    private static final double LgE = lg(Math.E);

    void sortStrings(final Stream<Integer> wordCounts, final int totalOps) {
        logger.info("sortStrings: beginning String sorts");
        wordCounts.forEach(x -> doSortStrings(x, round( x: totalOps / minComparisons(x))));
    }
    private void doSortStrings(final int n, final int m) {
        benchmarkStringSorters(COMMON_CHINESE_WORDS_CORPUS, MSDStringSortBenchmark.getWords(COMMON_CHINESE_WORDS_CORPUS,
    }
```

```
package sort.counting;

import ...

public class MSDStringSortBenchmarkTest {
    MSDStringSortBenchmark benchmark;
    final static String[] args = new String[]{"23251","46502","95237","190475","380951","999999"};
    @Before
    public void setUp() throws Exception {
        final Config config = Config.load(LSDStringSortBenchmark.class);
        benchmark = new MSDStringSortBenchmark();
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void sortStrings() { benchmark.sortStrings(Arrays.stream(args).map(Integer::parseInt),  totalOps: 60000000); }
}
```

Unit Test

For all sort methods we created a GetUnsortedArray class to generate the array used to test.

And there is something need to be discussed of MSD and LSD radix sort.

To determine whether the sorting is successful or not, we directly used the compareTo method of String to determine the value of the pinyin strings. We didn't use Collator to compare, because in the case of different characters with the same sound, MSD and LSD radix sort will keep the order before sorting, but Collator determines that these two characters are not equal, and we think our order is correct even if we only consider pinyin sorting.

For example: "滨" and "彬" have same tone and pinyin, which is " bin". Our algorithm think "滨,彬" and "彬, 滨" are the same things. but when we use Collator to compare，"滨" is larger than "彬". but "滨,彬" and "彬, 滨" are all correct order, if we only consider pinyin order.

```java
@Test
public void sort() {
    String[] strary = GetUnsortedArray.get();
    MSDStringSort.sort(strary);
    assertTrue(PinyinHelper.isSorted(strary, type: "MSD"));
}
```

PinyinHelper.java TimSort.java GetUnsortedArray.java

```java
import ...

public class GetUnsortedArray {
    public static LazyLogger logger = new LazyLogger(GetUnsortedArray.class);
    public static String[] get(){
        String path = "src/main/resources/shuffledChinese.txt";
        try {
            ArrayList<String> arrayList = new ArrayList<~>();
            String encoding="UTF-8";
            File file=new File(path);
            if(file.isFile() && file.exists()){ //判断文件是否存在
                InputStreamReader read = new InputStreamReader(
                        new FileInputStream(file),encoding);//考虑到编码格式
                BufferedReader bufferedReader = new BufferedReader(read);
                String lineTxt = null;
                while((lineTxt = bufferedReader.readLine()) != null){
                    arrayList.add(lineTxt);
                }
                read.close();
            }else{
                System.out.println("找不到指定的文件");
            }
            String[] strary = arrayList.toArray(new String[0]);
            return  strary;
        }catch (Exception e){
            logger.info("读取"+path+"文件失败");
            return null;
        }

    }
```

```java
public static String[] get(String size){
```

```
    public static String[] get(String size){
        String substring = size.substring(0, size.length());
        double parseDouble = Double.parseDouble(substring);
        double v=0d;
        if (size.charAt(size.length())=='K'||size.charAt(size.length())=='k'){
            v = parseDouble / 10.5 / 1024*999998;
        }else {
            v = parseDouble / 10.5 *999998;
        }

        String[] results = new String[(int) v];
        System.arraycopy(get(), srcPos: 0,results, destPos: 0,(int)v);
        return results;
    }
}
```

Relationship Conclusion

From the result of sorting different sizes of Chinese names in 250k, 500k, 1M, 2M, 4M, 10M, we can see:

a. When the size of names is under 4M, the efficiency relationship among these five methods is Huskysort >MSD radix sort >Timsort >LSD radix sort >= Dual-pivot Quicksort;

b. When the size of names is over 4M, Dual-pivot Quicksort is faster than LSD radix sort；

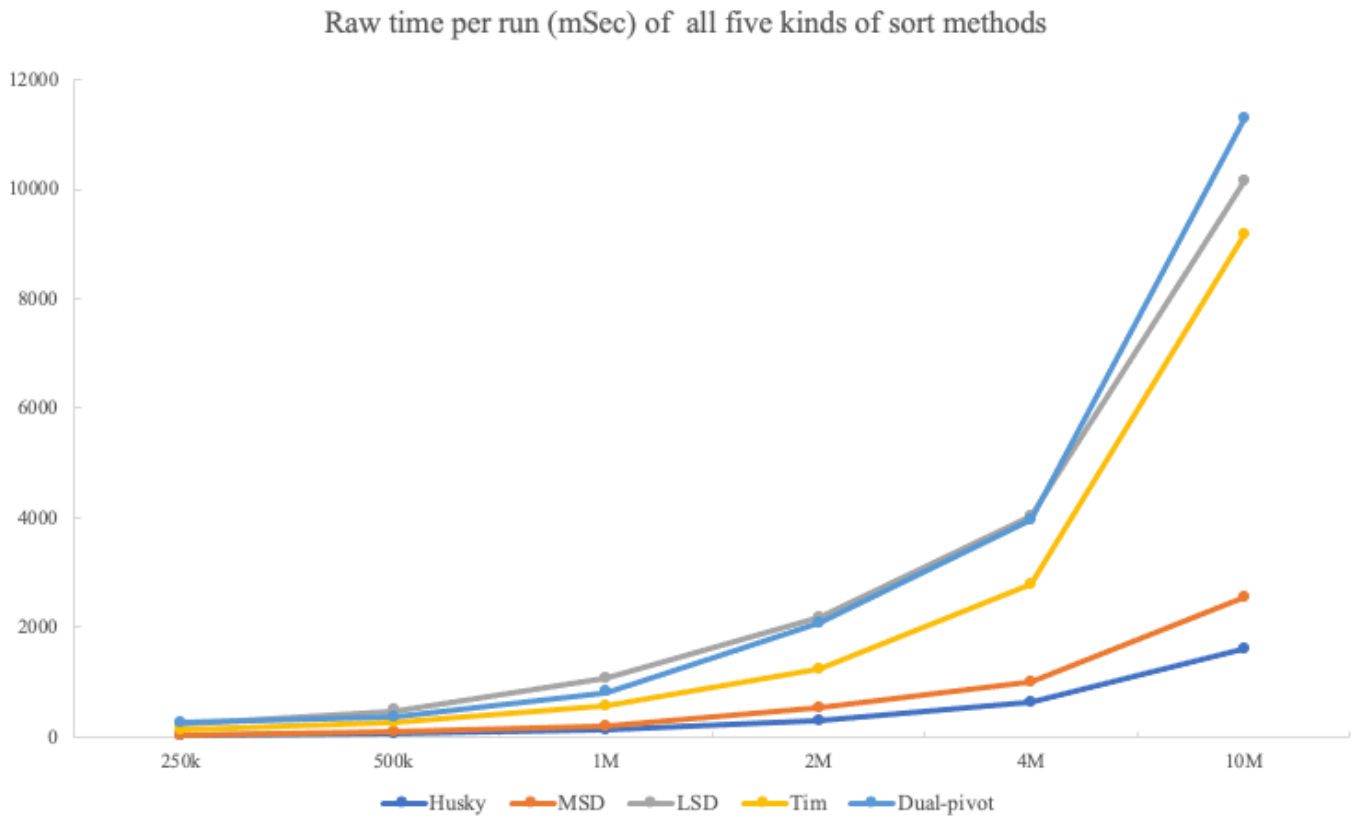
c. As the size of names goes up, the gap of efficiency difference becomes more obvious. Huskysort and MSD radix sort have better performance.

If LSD radix sort uses the same way as MSD radix sort to convert the Chinese character array into a two-dimensional array before sorting and store the Chinese characters and the corresponding pinyin strings. We can assume that the efficiency of LSD will go up.

Graphical Representation

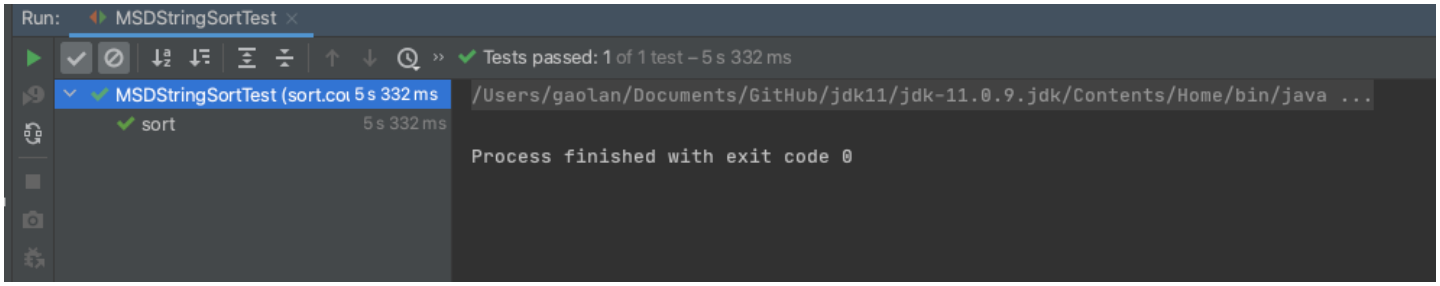
Raw time per run (mSec) in five kinds of sort methods

Array size	23251	46502	95237	190475	380951	999999
File size	250k	500k	1M	2M	4M	10M
Husky	31.26	66.95	142.74	304.62	638.89	1601.97
MSD	39.86	87.34	195.71	520.4	992.92	2536.55
LSD	213.06	480.53	1053.91	2188.75	4025.94	10141.85
Tim	114.22	265.63	563.21	1253.73	2788.93	9175.25
Dual-pivot	253.12	367.02	817.32	2075.18	3957.85	11299.43

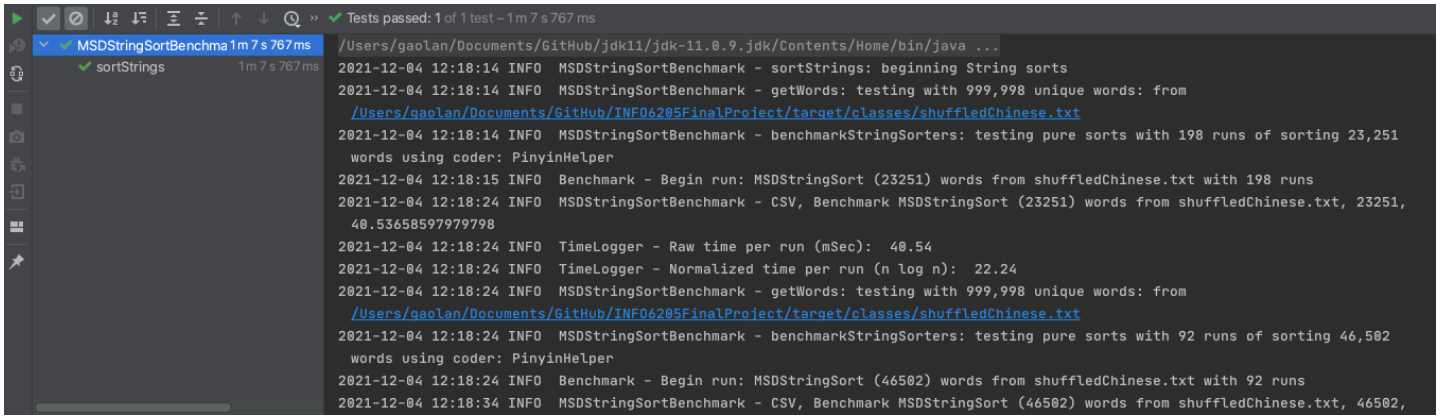


Unit test result

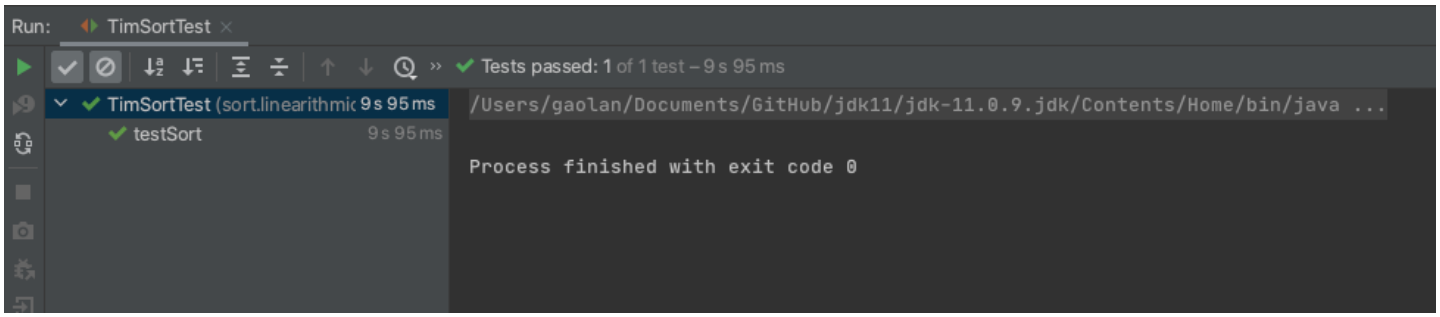
MSD radix sort



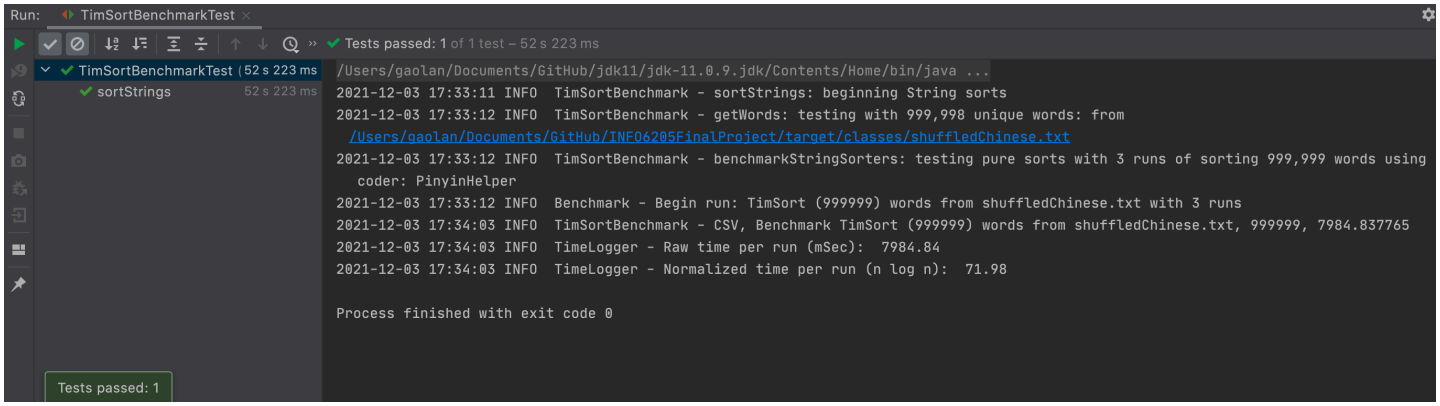
Benchmark



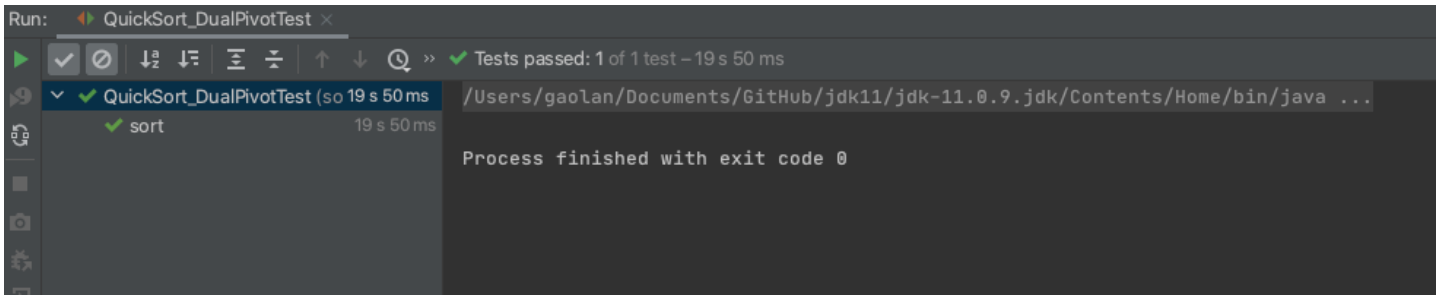
Timsort



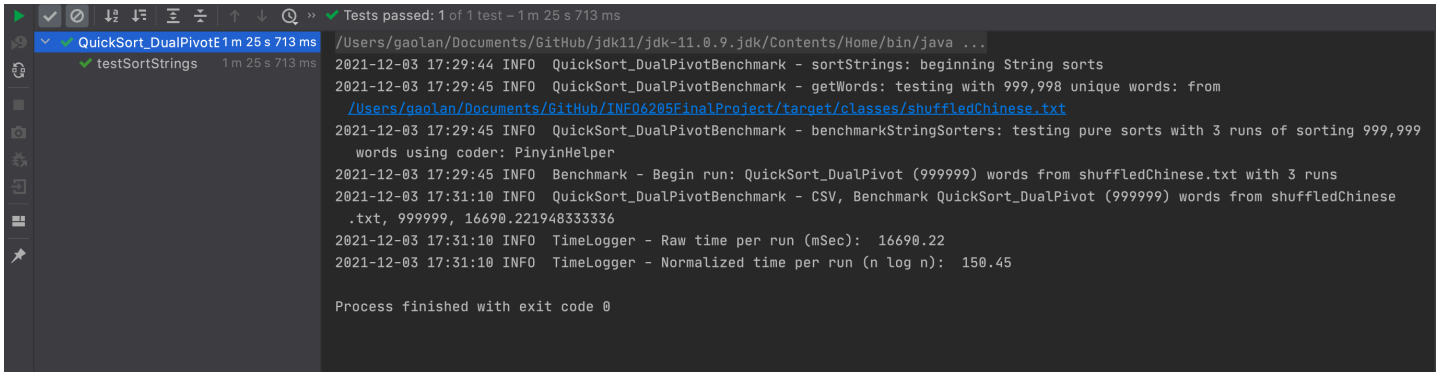
Benchmark



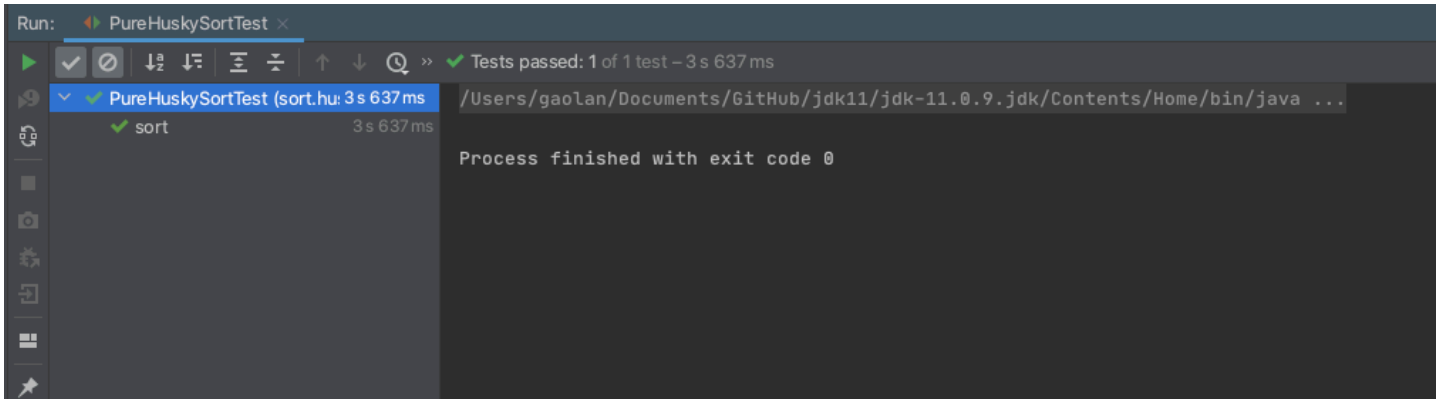
Dual-pivot Quicksort



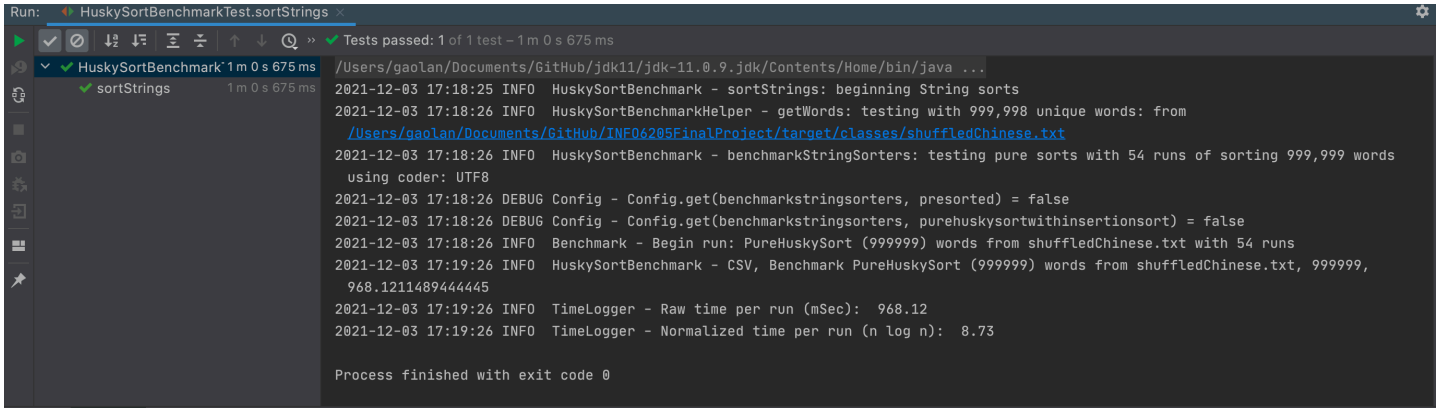
Benchmark



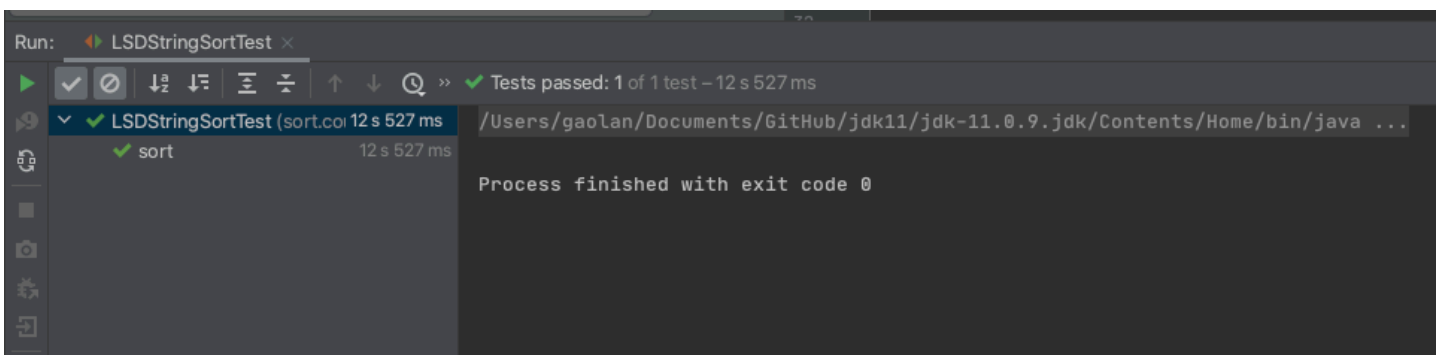
Huskysort



Benchmark

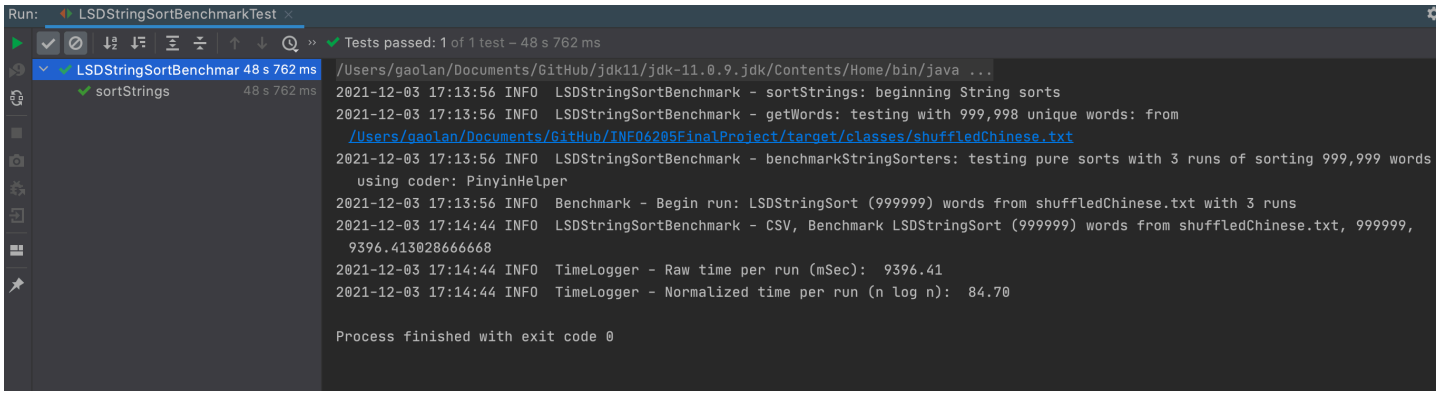


LSD radix sort



Benchmark

Benchmark


Run:
LSDStringSortBenchmarkTest
Tests passed: 1 of 1 test – 48 s 762 ms
LSDStringSortBenchmar 48 s 762 ms
sortStrings 48 s 762 ms
/Users/gaolan/Documents/GitHub/jdk11/jdk-11.0.9.jdk/Contents/Home/bin/java ...
2021-12-03 17:13:56 INFO LSDStringSortBenchmark - sortStrings: beginning String sorts
2021-12-03 17:13:56 INFO LSDStringSortBenchmark - getWords: testing with 999,998 unique words: from
/Users/gaolan/Documents/GitHub/INFO6205FinalProject/target/classes/shuffledChinese.txt
2021-12-03 17:13:56 INFO LSDStringSortBenchmark - benchmarkStringSorters: testing pure sorts with 3 runs of sorting 999,999 words
using coder: PinyinHelper
2021-12-03 17:13:56 INFO Benchmark - Begin run: LSDStringSort (999999) words from shuffledChinese.txt with 3 runs
2021-12-03 17:14:44 INFO LSDStringSortBenchmark - CSV, Benchmark LSDStringSort (999999) words from shuffledChinese.txt, 999999,
9396.413028666668
2021-12-03 17:14:44 INFO TimeLogger - Raw time per run (mSec): 9396.41
2021-12-03 17:14:44 INFO TimeLogger - Normalized time per run (n log n): 84.70
Process finished with exit code 0