

CS203B - Assignment2 Explanation

谢玉博 12012515

Problem1.java

```
import java.util.*;

public class Problem1 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int n = input.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = input.nextInt();
        }
        System.out.println(mergeSortCount(arr, 0, n - 1));
    }

    public static int mergeSortCount(int[] arr, int left, int right) {
        int swapCount = 0;
        if (left < right) {
            int mid = (left + right) / 2;
            swapCount += mergeSortCount(arr, left, mid);
            swapCount += mergeSortCount(arr, mid + 1, right);
            swapCount += merge(arr, left, mid, right);
        }
        return swapCount;
    }

    public static int merge(int[] arr, int left, int mid, int right) {
        int[] temp = new int[right - left + 1];
        int i = left, j = mid + 1, k = 0;
        int swapCount = 0;
        while (i <= mid && j <= right) {
            if (arr[i] <= arr[j]) {
                temp[k++] = arr[i++];
            } else {
                temp[k++] = arr[j++];
                swapCount += mid - i + 1;
            }
        }
        while (i <= mid) {
            temp[k++] = arr[i++];
        }
        while (j <= right) {
            temp[k++] = arr[j++];
        }
    }
}
```

```

        for (i = left, k = 0; i <= right; i++, k++) {
            arr[i] = temp[k];
        }
        return swapCount;
    }
}

```

Algorithms explanation

Problem1.java performs the merge sort algorithm on an array of integers, and then counts the number of swaps performed during the sorting process.

The program reads an integer `n` from the user input, which specifies the number of integers to sort. Then, it reads `n` integers from the user input and stores them in an integer array named `arr`.

The `mergeSortCount` method is called with the integer array `arr`, and the left and right indices of the array. This method recursively sorts the array using the merge sort algorithm, and returns the number of swaps performed during the sorting process.

The `merge` method is called by the `mergeSortCount` method to merge two sorted sub-arrays into a single sorted array. It uses a temporary array `temp` to store the merged array, and counts the number of swaps required to merge the sub-arrays.

Finally, the `mergeSortCount` method returns the total number of swaps performed during the entire sorting process, which is printed to the console.

Time complexity

The time complexity of this code is $O(n \log n)$, where `n` is the size of the input array.

The merge sort algorithm has a time complexity of $O(n \log n)$ for sorting an array of size `n`. The `mergeSortCount` method recursively calls itself with smaller sub-arrays, and then calls the `merge` method to merge the sorted sub-arrays, which takes $O(n)$ time. Therefore, the total time complexity of the `mergeSortCount` method is $O(n \log n)$.

Since the main method calls only the `mergeSortCount` method, the time complexity of the entire program is also $O(n \log n)$.

Correctness verification

I have this code to generate a random array for test:

```

import java.io.*;
import java.util.*;

public class RandomArrayGenerator {
    public static void main(String[] args) {
        Random rand = new Random();
        int arraySize = rand.nextInt(1000000) + 1;
    }
}

```

```

        System.out.println(arraySize);

        int[] randArray = new int[arraySize];
        for (int i = 0; i < arraySize; i++) {
            randArray[i] = rand.nextInt(1000000000 + 1);
        }
        try {
            FileWriter writer = new FileWriter("array.txt");
            writer.write(arraySize + "\n");
            for (int i = 0; i < arraySize; i++) {
                writer.write(randArray[i] + " ");
            }
            writer.close();
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}

```

For example, I run RandomArrayGenerator.java and get the random array below and input it into the given test code (801389 integers are in the array):

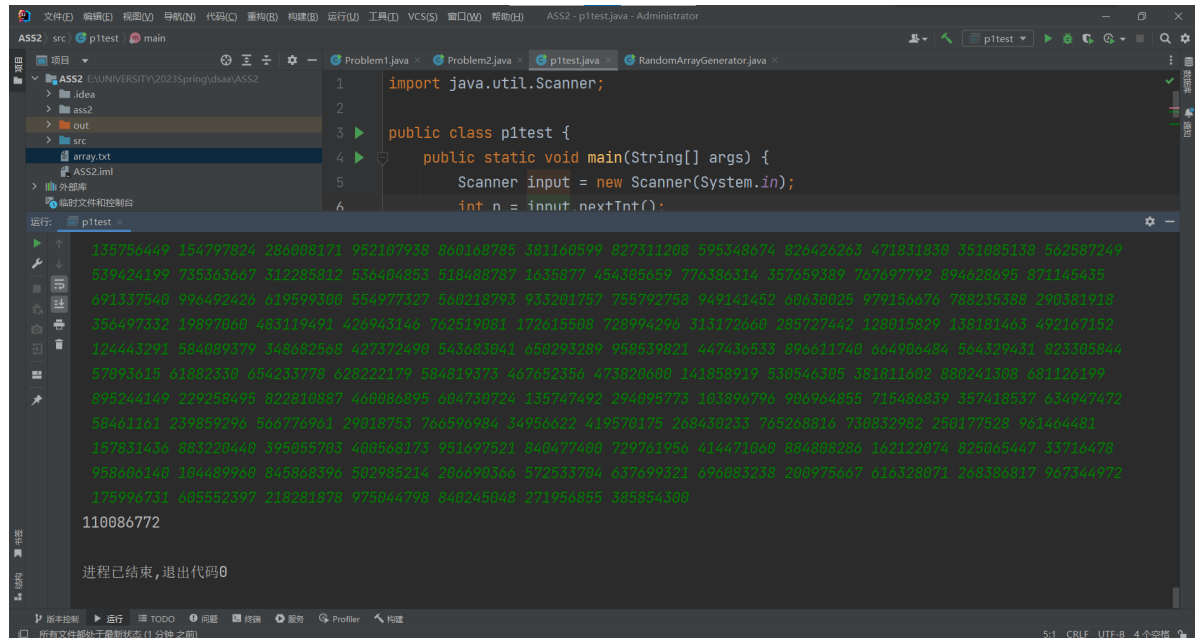
```

array.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
801389
150826706 371415797 702523825 575361980 798843683 612880310 619990153 921505902 753134268 212189563 693381933 640587777 5
1303249 968101567 825666264 216388905 553650545 273099273 418210042 613079944 642858069 166980517 785210655 983000618 294
82038053 22181876 374270349 170557996 946447071 954804384 943602482 336978715 463351893 447116195 466343125 292986164 315
06028952 504673107 314108334 27560262 897928003 990695443 729131950 15796659 328774553 641046133 379597311 385275729 5857
259932 403639328 860257752 147082119 392595057 897014101 851222148 42514561 931139920 687114322 184866798 92471149 840563
567504446 126497227 796597869 398746267 972346003 921405056 725926090 552865512 571495665 524919858 962937105 72957274 71
43 393597204 593440402 268872570 259572149 398191176 555012915 700545594 709071347 53296285 487198489 129969118 516218898
8796 816073745 293241895 572437276 996933730 58188478 432204176 130774422 669205722 851416126 52783780 862038213 20586294
0 724507623 844537647 694556313 132853917 827659085 451384507 659501401 662569487 440115085 308716826 418605194 252942475
32867085 904552612 899123045 192312787 687468908 997517458 702863471 622639803 863486328 128947596 169134010 878273282 40
957243 680549930 233616239 280800038 731088022 5124280 676199803 983797801 871441014 979612665 114386184 107751660 450094
300318405 351767035 290396653 873534754 336884999 554968834 989512805 255441877 313506935 110230554 500577432 471899211 3
671 846939084 133378315 970885531 475452306 765596957 942731873 810342534 737747016 994862737 865015529 551975957 9276535
108610086 460967666 27105322 495933305 957501395 673789665 507571235 159034290 833494442 389567958 552965776 518368644 61
70 51534407 166963511 996908208 835859500 921068488 650421952 379116477 539116873 210244177 6435542 166348231 869700386 3
8 728142816 74404793 898188906 66214418 185184874 702101779 544224024 490449720 658290264 998607760 519212447 489322231 3
20 917450827 269738417 322357497 227942258 347372226 153069878 325622570 769675244 251893451 109328872 853291539 2774898
9328 680858958 76823903 238138449 590716464 934759649 267341356 338346469 165637477 325100627 793985466 417684854 2977015
58970362 328808285 646955927 547605598 88400577 244591506 498484854 425103802 199317626 569128341 603880596 985778692 225
09467 579860725 157912171 987509472 927060979 677823244 214592426 94931726 463235185 226047129 534926065 154969433 978751
6623938 701415295 449840517 419462323 987608150 980666323 904830532 322425141 801528764 848741348 379078808 995351666 218
80266213 799384269 840591933 959546133 536753660 715706855 578722243 427331485 166268417 844496118 976594870 131663575 50

```

第 1 行, 第 1 列 100% Unix (LF) UTF-8

The result the test code gives is:

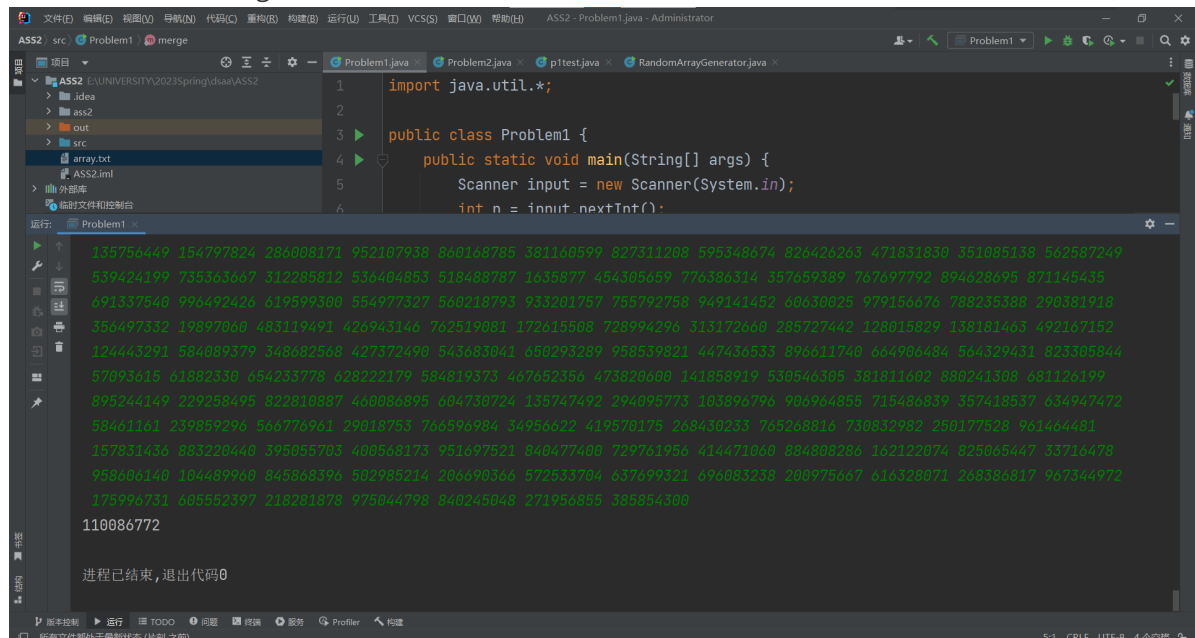


```
1 import java.util.Scanner;
2
3 public class p1test {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         int n = input.nextInt();
```

110086772

进程已结束,退出代码0

and the code I write gives result as:

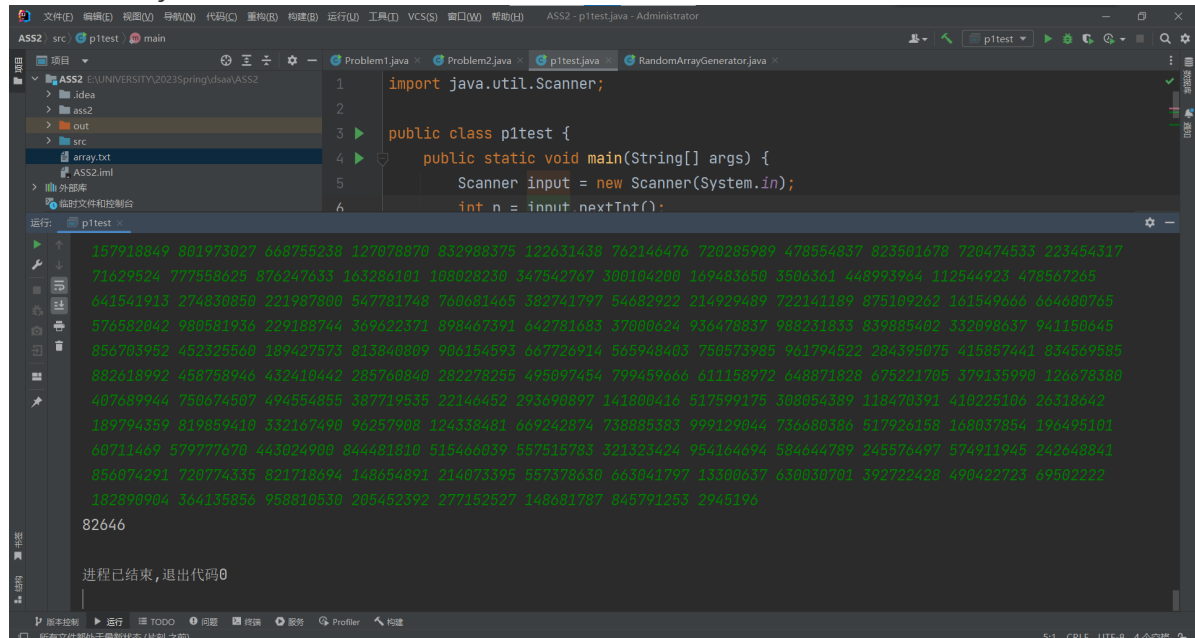


```
1 import java.util.*;
2
3 public class Problem1 {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         int n = input.nextInt();
```

110086772

进程已结束,退出代码0

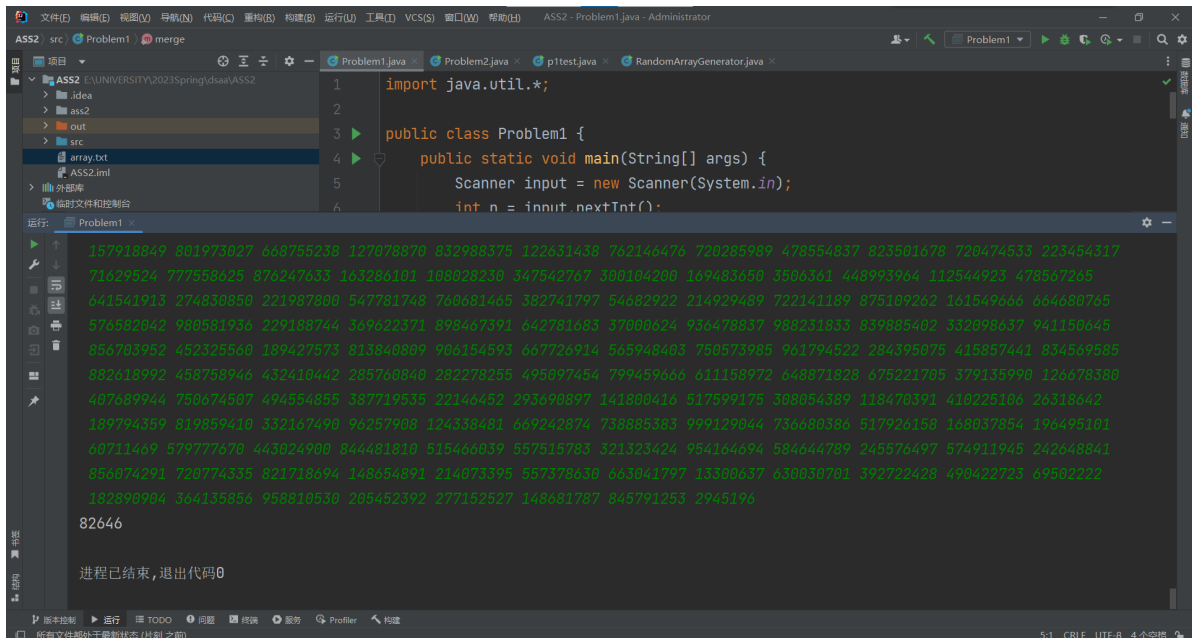
One more try:



```
1 import java.util.Scanner;
2
3 public class p1test {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         int n = input.nextInt();
```

82646

进程已结束,退出代码0



```
1 import java.util.*;
2
3 public class Problem1 {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         int n = input.nextInt();
7
8         // ... (omitted code for generating random numbers) ...
9
10        System.out.println(82646);
11
12        进程已结束,退出代码0
```

Hence, the output result of `Problem1.java` is correct.

Problem2.java

```
import java.util.*;

public class Problem2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String n = sc.nextLine();
        int k = Integer.parseInt(sc.nextLine());
        System.out.println(removeKDigits(n, k));
    }

    private static String removeKDigits(String num, int k) {
        if (num.length() == k) {
            return "0";
        }

        Stack<Character> stack = new Stack<>();
        for (int i = 0; i < num.length(); i++) {
            char digit = num.charAt(i);
            while (k > 0 && !stack.isEmpty() && stack.peek() < digit) {
                stack.pop();
                k--;
            }
            stack.push(digit);
        }

        while (k > 0) {
            stack.pop();
            k--;
        }
    }
}
```

```

    }

    StringBuilder sb = new StringBuilder();
    while (!stack.isEmpty()) {
        sb.append(stack.pop());
    }
    sb.reverse();

    while (sb.length() > 1 && sb.charAt(0) == '0') {
        sb.deleteCharAt(0);
    }
    return sb.toString();
}
}

```

Algorithms explanation

The class has a public static method named `main` which takes an array of Strings as input. The method is used to read an integer and a string from the console and then calls another private static method named `removeKDigits` passing these inputs as parameters.

The `removeKDigits` method takes two inputs, a string and an integer, and returns a string. It is used to remove `k` digits from the given number in such a way that the resulting number is the smallest possible number.

The implementation of the method involves the use of a stack data structure. The input number is iterated through, and digits are pushed into the stack. While the current digit is greater than the previous digit, the previous digit is popped from the stack. This ensures that the digits are in non-increasing order. Once the required number of digits has been popped, the remaining digits are popped from the stack and added to a `StringBuilder` object in reverse order. Finally, leading zeros are removed from the result by deleting any zeros at the beginning of the string.

Overall, this code solves a problem of finding the smallest number possible by removing `k` digits from the given number.

Time complexity

The time complexity of this code is $O(n)$, where `n` is the length of the input number.

The for loop that iterates through the input number has a time complexity of $O(n)$. Inside the loop, the stack operations (push and pop) take constant time, and their total time complexity is also $O(n)$. Finally, the while loop that removes leading zeros from the result has a time complexity of $O(n)$.

Therefore, the total time complexity of the `removeKDigits` method is $O(n)$. Since the `main` method calls this method once, the overall time complexity of the code is also $O(n)$.

Correctness verification

For the four `Test` samples given, the output results from `Problem2.java` are in complete agreement with the correct reference results. It is concluded that the correctness of `Problem2.java` is verified.