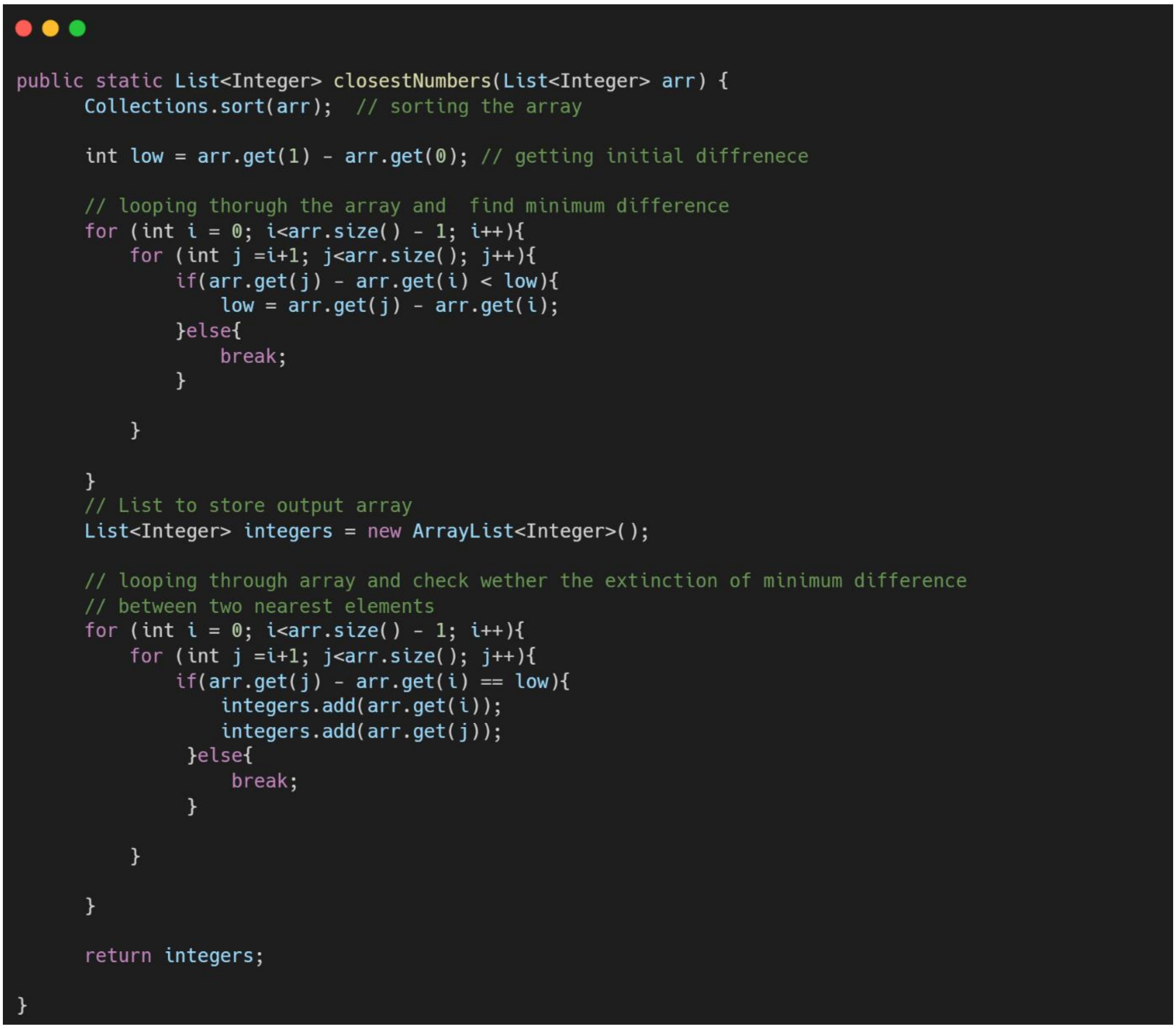
CO322: Data Structures and Algorithms

Lab 03: HR problems set 02

WIJERATHNE E.S.G

E/18/397

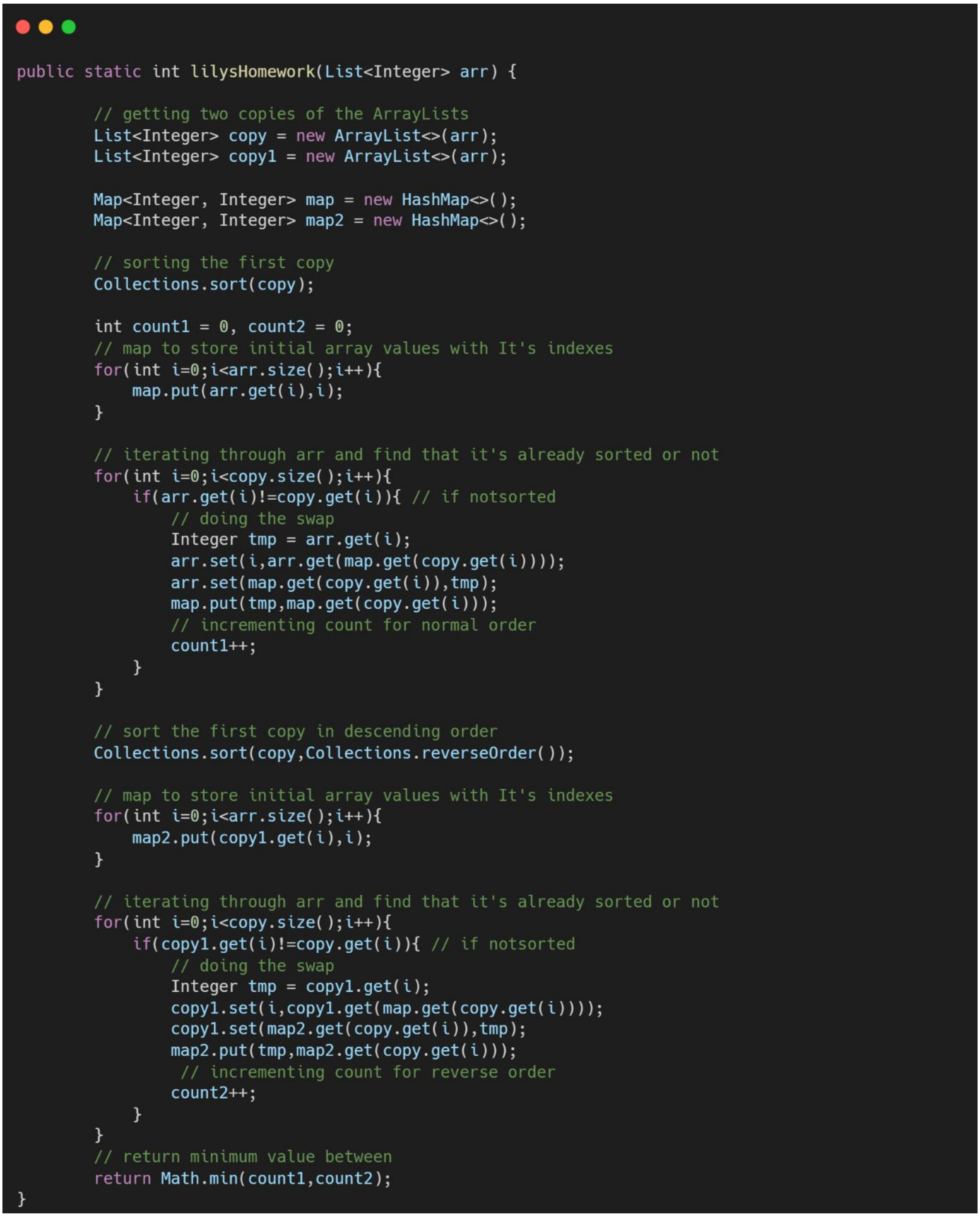
01. Closest Numbers



```
public static List<Integer> closestNumbers(List<Integer> arr) {
      Collections.sort(arr);  // sorting the array

      int low = arr.get(1) - arr.get(0); // getting initial diffrenece

      // looping thorugh the array and  find minimum difference
      for (int i = 0; i<arr.size() - 1; i++){
          for (int j =i+1; j<arr.size(); j++){
              if(arr.get(j) - arr.get(i) < low){
                  low = arr.get(j) - arr.get(i);
              }else{
                  break;
              }

          }

      }
      // List to store output array
      List<Integer> integers = new ArrayList<Integer>();

      // looping through array and check wether the extinction of minimum difference
      // between two nearest elements
      for (int i = 0; i<arr.size() - 1; i++){
          for (int j =i+1; j<arr.size(); j++){
              if(arr.get(j) - arr.get(i) == low){
                  integers.add(arr.get(i));
                  integers.add(arr.get(j));
               }else{
                   break;
               }

          }

      }

      return integers;

}
```

First, I sorted the array and searched through the list to find the shortest difference between the two elements. Used break statements to avoid unnecessary loops. Once I found the minimum difference value, I again went through the list and checked for the nearest element, which gave the same difference as the low value. Once I find an occurrence, I will add both closest values to the output Integer List.

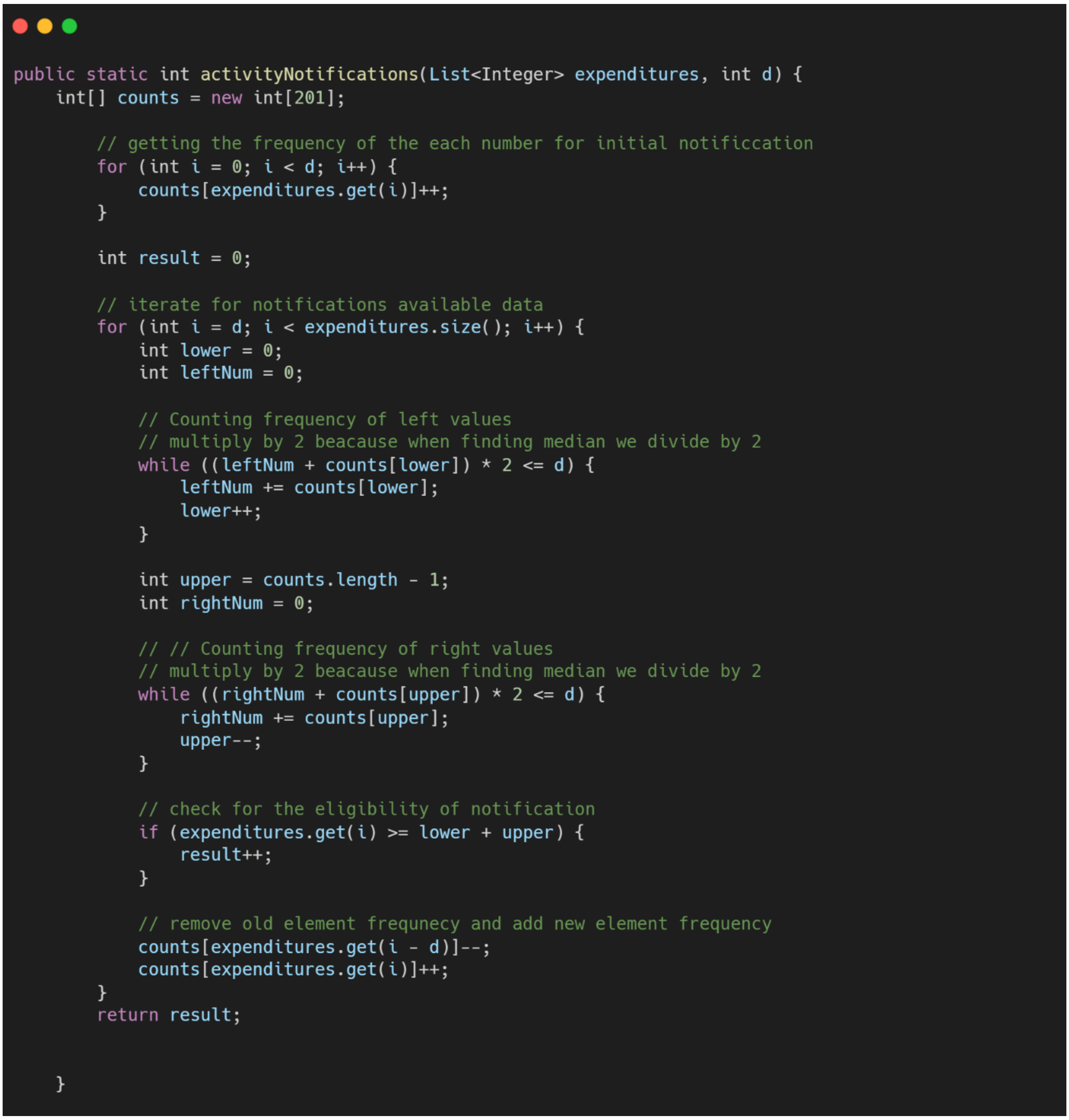
02. Lily's Homework



```
public static int lilysHomework(List<Integer> arr) {

        // getting two copies of the ArrayLists
        List<Integer> copy = new ArrayList<>(arr);
        List<Integer> copy1 = new ArrayList<>(arr);

        Map<Integer, Integer> map = new HashMap<>();
        Map<Integer, Integer> map2 = new HashMap<>();

        // sorting the first copy
        Collections.sort(copy);

        int count1 = 0, count2 = 0;
        // map to store initial array values with It's indexes
        for(int i=0;i<arr.size();i++){
            map.put(arr.get(i),i);
        }

        // iterating through arr and find that it's already sorted or not
        for(int i=0;i<copy.size();i++){
            if(arr.get(i)!=copy.get(i)){ // if notsorted
                // doing the swap
                Integer tmp = arr.get(i);
                arr.set(i,arr.get(map.get(copy.get(i))));
                arr.set(map.get(copy.get(i)),tmp);
                map.put(tmp,map.get(copy.get(i)));
                // incrementing count for normal order
                count1++;
            }
        }

        // sort the first copy in descending order
        Collections.sort(copy,Collections.reverseOrder());

        // map to store initial array values with It's indexes
        for(int i=0;i<arr.size();i++){
            map2.put(copy1.get(i),i);
        }

        // iterating through arr and find that it's already sorted or not
        for(int i=0;i<copy.size();i++){
            if(copy1.get(i)!=copy.get(i)){ // if notsorted
                // doing the swap
                Integer tmp = copy1.get(i);
                copy1.set(i,copy1.get(map.get(copy.get(i))));
                copy1.set(map2.get(copy.get(i)),tmp);
                map2.put(tmp,map2.get(copy.get(i)));
                 // incrementing count for reverse order
                count2++;
            }
        }
        // return minimum value between
        return Math.min(count1,count2);
}
```

First, I have to make two copies of the initial array for my future use. Then I initialized two HashMap also. A copy of the initial array was sorted. Then add data to one of my HashMaps. The data was the initial array element as the key and the index of that element as the value. Then iterated through the array and checked whether the array is sorted or not. If not swapping was done and incremented the swapping count. After that, redo the above procedure with the initial array and reversed version of the sorted initial value. It will also give another answer according to the reverse sorted array. From I returned the smallest value as the final answer from both answer.

03. Fraudulent Activity Notifications



```
public static int activityNotifications(List<Integer> expenditures, int d) {
    int[] counts = new int[201];

        // getting the frequency of the each number for initial notificcation
        for (int i = 0; i < d; i++) {
            counts[expenditures.get(i)]++;
        }

        int result = 0;

        // iterate for notifications available data
        for (int i = d; i < expenditures.size(); i++) {
            int lower = 0;
            int leftNum = 0;

            // Counting frequency of left values
            // multiply by 2 beacause when finding median we divide by 2
            while ((leftNum + counts[lower]) * 2 <= d) {
                leftNum += counts[lower];
                lower++;
            }

            int upper = counts.length - 1;
            int rightNum = 0;

            // // Counting frequency of right values
            // multiply by 2 beacause when finding median we divide by 2
            while ((rightNum + counts[upper]) * 2 <= d) {
                rightNum += counts[upper];
                upper--;
            }

            // check for the eligibility of notification
            if (expenditures.get(i) >= lower + upper) {
                result++;
            }

            // remove old element frequnecy and add new element frequency
            counts[expenditures.get(i - d)]--;
            counts[expenditures.get(i)]++;
        }
        return result;


    }
```

A restriction that states that no element in the input Array, including any daily debit values, may have a value greater than 200 can be seen in the function description. We can construct the frequency Array needed for counting sort thanks to this relatively tiny amount.

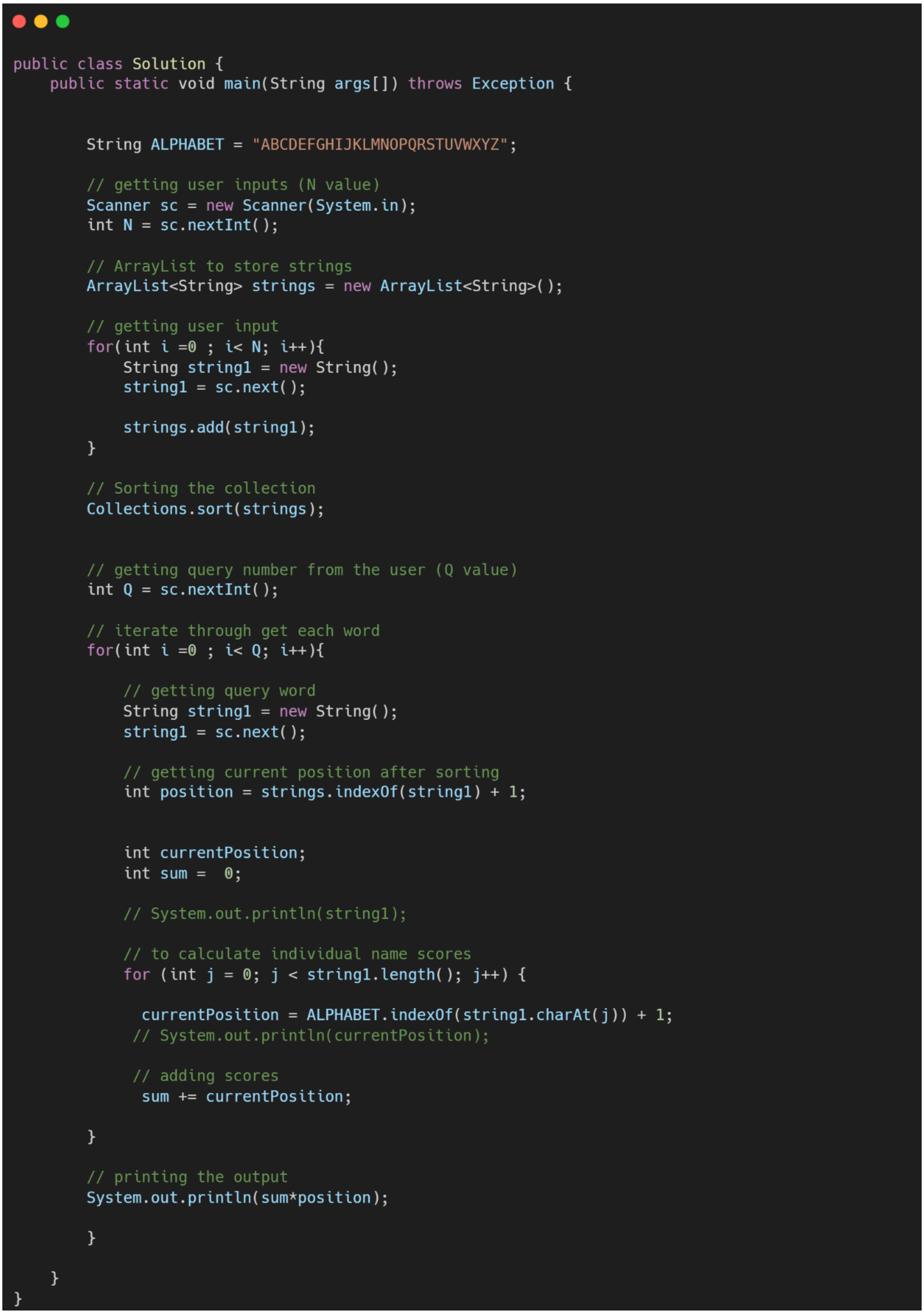
The sum is a counter variable that we have. This will show the cumulative sum over all iterations of the values in counts. For counting sort, calculating an accumulative sum is a prerequisite. Using a for loop, counts are iterated over, with each iteration adding the current number to the sum.

Within the loop, there are two conditionals—two if statements—that, if true, cause the loop to end and a return value to be generated. Multiplication is employed to detect if a day is odd or even rather than division, which is more computationally expensive and can lead to issues like floating point decimals. The multiplier we'll use is 2 since the main function checks to see if the median times two (* 2) is more than or equal to the daily debit, and we want to twice the value we're evaluating.

Days may be divided by two in an even number of equal parts, hence if total * 2 equals days on any iteration, we can conclude that sum is even. We may set days as the upper limit for sum because of the way counting sort hashes indices. This breaks the loop and aims for the current index I which corresponds to the value from the debits Array and, as a result, corresponds to the median value of the current window.

Once I have done with the calculations for single expenditures[i] element I remove the old element frequency from the array and add a new element frequency to the array.

04. Project Euler #22: Names scores



```java
public class Solution {
    public static void main(String args[]) throws Exception {


        String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

        // getting user inputs (N value)
        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();

        // ArrayList to store strings
        ArrayList<String> strings = new ArrayList<String>();

        // getting user input
        for(int i =0 ; i< N; i++){
            String string1 = new String();
            string1 = sc.next();

            strings.add(string1);
        }

        // Sorting the collection
        Collections.sort(strings);


        // getting query number from the user (Q value)
        int Q = sc.nextInt();

        // iterate through get each word
        for(int i =0 ; i< Q; i++){

            // getting query word
            String string1 = new String();
            string1 = sc.next();

            // getting current position after sorting
            int position = strings.indexOf(string1) + 1;


            int currentPosition;
            int sum =  0;

            // System.out.println(string1);

            // to calculate individual name scores
            for (int j = 0; j < string1.length(); j++) {

              currentPosition = ALPHABET.indexOf(string1.charAt(j)) + 1;
             // System.out.println(currentPosition);

             // adding scores
              sum += currentPosition;

        }

        // printing the output
        System.out.println(sum*position);

        }

    }
}
```

First, I define string for the capital alphabet, which I will use to get indexes of the characters. Then I get user strings according to the need. Once I have the string list, I will sort it using Collections.sort(). Then I get the Q value from the user and create a loop to get strings according to that Q value. Once I get a string, I will get the current position of the string in the sorted list. Then I go through that string character by character and sum up values according to each character. Once I have the sum, I will multiply it by the position of the string in the sorted list. This will be my output.