# Question 1

文本, 信件

描述已自动生成

**import** java.util.Scanner;

**public** **class** Q1 {

**public** **static** **void** main (String args[]) {

Scanner sc = **new** Scanner(System.***in***);

// Number of inputs

**int** N = Integer.*parseInt*(sc.nextLine());

// Input the number and calculate their average

**int** intArray [] = **new** **int** [N];

**double** sum = 0.0;

**double** average = 0.0;

**for** (**int** i = 0 ; i < N ; i++) {

**int** input = Integer.*parseInt*(sc.nextLine());

intArray [i] = input;

sum = sum + (**double**) input;

}

average = sum / (**double**) N;

// Find the number closest to the average

**double** minDifference = Integer.***MAX\_VALUE***;

**int** outputElement = 0;

**for** (**int** i = 0 ; i < N ; i++) {

**double** difference = Math.*abs*(average - (**double**)intArray[i]);

/\* If two values are equally close to the average, choose the one which was earliest in the list, so we ignore '=' in the case . \*/

**if**(difference < minDifference) {

outputElement = intArray[i];

minDifference = difference;

}

}

System.***out***.println(outputElement);

}

}

**[Big O complexity] - O(n)**

I use two for loops in the program,

each loop is related to the input number N;

So the complexity is O(N) + O（N） = O(N) ;

# Question 2

文本

描述已自动生成

文本, 信件

描述已自动生成

**import** java.util.Scanner;

**import** java.util.Set;

**import** java.util.LinkedHashSet;

**import** java.util.List;

**import** java.util.ArrayList;

**public** **class** Q2 {

**public** **static** **void** main (String args[]) {

Scanner sc = **new** Scanner(System.***in***);

// Input the quantity number used in the lottery

**int** N = Integer.*parseInt*(sc.nextLine());

// Input the quantity number drawn in a lottery draw

**int** D = Integer.*parseInt*(sc.nextLine());

sc.close();

// Use Monte Carlo Model

**int** ALL = 1000000;

**int** count = 0;

MONTECARLO:

**for** (**int** i = 0; i < ALL; i++) {

//Create a new drawbox and put it into Set.

Set<Integer> uniqueNumbers = **new** LinkedHashSet<Integer>();

// Avoid duplicate and remember the input sequence.

**while**(uniqueNumbers.size() < D) {

// Select one number from Lottery [1,2....,N]

**int** randomNumber = (**int**) (N \* Math.*random*() + 1);

uniqueNumbers.add(randomNumber);

}

// Check whether the drawn numbers are ASCending

List<Integer> list = **new** ArrayList<>(uniqueNumbers);

**for** (**int** j = 1; j < list.size(); j++) {

// Not ascending, continue to the next Monte Carlo

**if** (list.get(j - 1) > list.get(j)) {

**continue** MONTECARLO;

}

}

// Ascending, count a time

count++;

}

// Calculate the probability into percentage

**double** p = (**double**)(count \* 100)/ (**double**) ALL;

// Round up to the nearest Integer

System.***out***.println(Math.*round*(p));

}

}

**[Big O complexity] - O(n²)**

I use two for loops in the program. The outer loop is related to ALL -- the simulation times of Monte Carlo Simualtion. The inner loop is related to D -- the quantity number drawn from lottery.

So the complexity is O(N) \* O（N） = O(N²);

# Question 3

文本, 信件

描述已自动生成

图片包含 图形用户界面

描述已自动生成

**import** java.util.Scanner;

**import** java.util.List;

**import** java.util.LinkedList;

**import** java.util.Collections;

**import** java.util.Comparator;

**public** **class** Q3 {

**public** **static** **void** main(String[] args) {

List<String> words = **new** LinkedList<String>();

Scanner sc = **new** Scanner(System.***in***);

// number of words followed by N lines

**int** N = Integer.*parseInt*(sc.nextLine());

// Each line contains a String

**for**(**int** i = 0; i < N; i++) {

String inputLine = sc.nextLine();

words.add(inputLine);

}

sc.close();

Collections.*sort*(words, **new** Comparator<String>() {

@Override

**public** **int** compare (String s1, String s2) {

String []alphabet = {"a","b","c","d","e","f","g",

"h","i","j","k","l","m","n","o","p","q",

"r","s","t","u","v","w","x","y","z"};

// Sort the words by the earliest character

**for** (**int** i = 0; i < 26; i++) {

**if**(s1.contains(alphabet[i]) && !s2.contains(alphabet[i]))

{

**return** -1;

}

**else** **if**(!s1.contains(alphabet[i])

&& s2.contains(alphabet[i]))

{

**return** 1;

}

**else**

{

**continue**;

}

}

// If two words use same characters -> alphabetical order

**return** s1.compareTo(s2);

}

}); // End of Collection and its Comparator.

// Print out the sorted words

**for**(String element:words) {

System.***out***.print(element + " ");

}

}

}

**[Big O complexity] - O(n \* log(n))**

1) When I add elements into LinkedList or output the element from the LinkedList, they are related to N -- the number of words, so the time complexity is O(n)

2) When I use Collection.sort(), it will use merge sort if the number of input is very high. The time complexity of merge sort is O(n \* log(n)).

Therefore, the time complexity of this program is

O(n) + O(n \* log(n)) + O(n) = O(n \* log(n))

# Question 4

文本, 信件

描述已自动生成文本

描述已自动生成文本

描述已自动生成

**Floyd's cycle-finding Algorithm:**

**FAST pointer moves 2 steps in one time**

**SLOW pointer moves 1 steps in one time**

**If there has a loop, FAST will finally catch up with SLOW.**

**public** **static** **int** findLoopLength(LinkedList mylist){

Link slow = mylist.first;

Link fast = mylist.first;

**boolean** hasLoop = **false**;

//Find whether has a LOOP

**while**(fast != **null** && fast.next != **null**){

slow = slow.next;

fast = fast.next.next;

**if**(slow == fast) {

hasLoop = **true**;

**break**;

}

}

// Calculate the length of Loop

// No LOOP/ Empty List -> default value 0

**int** loopLength = 0;

**if** (hasLoop) {

**do** {

fast = fast.next;

loopLength++;

} **while** (fast != slow);

}

**return** loopLength;

}

**[Big O complexity] - O(n)**

In the method, there is one while loop.

The loop is related with the num of Link.

***int*** *num = Integer.parseInt(myscanner.nextLine());*

So the Big O Complexity is O(n).