# Question 1

文本

描述已自动生成

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

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

**int** noRemainder = 1;

// Aggregate the LCM from 1 to 20.

**for** (**int** i = 1 ; i <= 20; i++) {

noRemainder = *lcm* (noRemainder, i);

}

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

}

// Greatest Common Divisor

**public** **static** **int** gcd (**int** a, **int** b) {

**if** (b == 0) {

**return** a;

}

**return** *gcd*(b, a % b);

}

// Least Common Multiple

// LCM(a,b) = a X b / GCD(a,b)

**public** **static** **int** lcm (**int** a, **int** b) {

**return** (a \* b) / *gcd*(a, b);

}

}

# Question 2

文本, 信件

描述已自动生成

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

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

// Total number of Monte Carlo

**int** N = 1000000;

**int** x,y;

**int** sum = 0;

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

x = (**int**) (Integer.***MAX\_VALUE*** \* Math.*random*());

y = (**int**) (Integer.***MAX\_VALUE*** \* Math.*random*());

// Coprime number of Monte Carlo

**if** (*gcd*(x,y) == 1) sum++;

}

**double** p = (**double**)100 \* sum / (**double**) N;

// Keep 2 decimal digits

System.***out***.println(String.*format*("%.2f", p) + "%");

}

// Greatest Common Divisor

// If gcd(a,b) == 1, then a and b are coprime.

**public** **static** **int** gcd (**int** a, **int** b) {

**if** (b == 0) {

**return** a;

}

**return** *gcd*(b, a % b);

}

}

# Question 3

文本, 信件

描述已自动生成

**import** java.util.Scanner;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.LinkedList;

**public** **class** Q3\_linkedList {

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

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

LinkedList <String> priorityQueue = **new** LinkedList<String>();

**while** (**true**) {

String userInput = scanner.nextLine();

//If input includes "INSERT" at first, add element to the queue.

**if**(userInput.split(" ")[0].toUpperCase().equals("INSERT")) {

priorityQueue.add(userInput.split(" ")[1]);

}

//If input is "REMOVE", sort the queue first and remove the first element.

**if**(userInput.toUpperCase().equals("REMOVE")) {

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

@Override

**public** **int** compare(String o1, String o2) {

// 1 - More Vowel has higher priority

**if**(*countVowels*(o2)!=*countVowels*(o1)) {

**return** *countVowels*(o2) - *countVowels*(o1);

}

// 2 - If same, longer word has higher priority

**else** **if** (o2.length()!= o1.length()) {

**return** o2.length() - o1.length();

}

// 3 - else, remain the input sequence

**else** {

**return** 1;

}

}

//Method: Compare the Vowels

**public** **static** **int** countVowels(String str) {

**int** count = 0;

**for** (**int** i = 0; i < str.length(); i++) {

**char** ch = Character.*toLowerCase*(str.charAt(i));

**if** (ch == 'a' || ch == 'e' || ch == 'i'

|| ch == 'o' || ch == 'u') {

count++;

}

}

**return** count;

}

}));

// Remove command is issued for an empty queue， nothing should happen.

if(!priorityQueue.isEmpty()) priorityQueue.poll();

}

// If input empty String, close the scanner.

**if** (userInput.isEmpty()) {

scanner.close();

**break**;

}

// Make a new input line, ignore other invalid input.

}

// Get the middle string of priority queue.

**int** size = priorityQueue.size();

String middleElement;

**for** (**int** i = 0; i < size / 2 - 1; i++) {

priorityQueue.poll();

}

// Even number -> the middle two -> nearest the front

**if**(size % 2 == 0) {

middleElement = priorityQueue.peek();

// Odd number -> the middle one

} **else** {

priorityQueue.poll();

middleElement = priorityQueue.peek();

}

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

}

}

# Question 4

文本, 信件

描述已自动生成

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.LinkedList;

**import** java.util.Scanner;

**public** **class** Q4 {

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

LinkedList <String> input = **new** LinkedList<String>();

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

// Input total number of words

**int** SIZE = sc.nextInt();

sc.nextLine(); // change a new line to avoid bug

// Input each word

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

input.add(sc.nextLine());

}

sc.close();

// Sort the List after input.

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

@Override

**public** **int** compare(String o1, String o2) {

// lower sum of ASCII has priority

**if**(sumOfASCLL(o1) != sumOfASCLL(o2)) {

**return** sumOfASCLL(o1) - sumOfASCLL(o2);

}

// REVERSE alphabetical order

**else** {

**return** o2.compareTo(o1);

}

}

// Method: Calculate the total sum of the ASCII characters

**public** **int** sumOfASCLL (String str) {

**int** sum = 0;

**for**(**int** i = 0; i < str.length(); i++) {

sum = sum + (**int**)str.charAt(i);

}

**return** sum;

}

}));

// Print out each element in the LinkedList

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

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

}

}

}

**【Question】 State the complexity of your program using Big O notation.**

【Answer】 The time complexity in my program is (N\*log (N))

Because if the element amount in Collection.sort() is very high,

Java will use MergSort to change the order.

In merge sort, if you want to use BinaryTree to divide N elements into

one element, you need log2(N) times.

In each layer, every element should compare each other for N time.

Therefore the Big O Notation for mergesort is O(N\*log(N)).

System.out.println(sumOfASCLL("one")); //322

System.out.println(sumOfASCLL("ten")); //327

System.out.println(sumOfASCLL("six")); //340

System.out.println(sumOfASCLL("two")); //346

System.out.println(sumOfASCLL("nine")); //426

System.out.println(sumOfASCLL("five")); //426

System.out.println(sumOfASCLL("four")); //444

System.out.println(sumOfASCLL("eight")); //529

System.out.println(sumOfASCLL("three")); //536

System.out.println(sumOfASCLL("seven")); //545