# Question 1

文本

描述已自动生成

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

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

// Total number of Monte Carlo

**int** N = 1000000;

**int** count = 0;

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

**int** n1 = (**int**) (Integer.***MAX\_VALUE*** \* Math.*random*());

**int** n2 = (**int**) (Integer.***MAX\_VALUE*** \* Math.*random*());

// Coprime number of Monte Carlo

**if**(*gcd*(n1,n2) == 1) count++;

}

**double** probability = (**double**) 100 \* count / (**double**) N;

// Keep 2 decimal digits

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

}

// gcd = 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;

}

**else** {

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

}

}

}

# Question 2

文本, 信件

描述已自动生成

**import** java.util.Queue;

**import** java.util.PriorityQueue;

**import** java.util.Comparator;

**import** java.util.Set;

**import** java.util.HashSet;

**import** java.util.Scanner;

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

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

Queue<String> pq = **new** PriorityQueue<String> ( **new** Comparator<String>() {

@Override

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

// 1 - more unique letters first

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

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

}

// 2 - If unique letter is same -> lexicographically

**else** {

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

}

}

// Use Set to calculate the number of unique letter.

// Set can filter the duplicate letter.

**public** **int** countUniqueChar(String input) {

Set<Character> uniqueChar = **new** HashSet<Character>();

**for**(**char** c : input.toCharArray()) {

uniqueChar.add(c);

}

**return** uniqueChar.size();

}

}); // end of Queue and its Comparator

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

**int** SIZE = 10000;

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

pq.add(sc.nextLine());

}

sc.close();

// Print out all sorted String in PriorityQueue

**while**(!pq.isEmpty()) {

System.***out***.println(pq.poll());

}

}

}

# Question 3

文本

描述已自动生成

文本, 信件

描述已自动生成

**import** java.util.Queue;

**import** java.util.LinkedList;

**import** java.util.Scanner;

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

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

Queue<String> queue = **new** LinkedList<String>();

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

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

String inputLine = sc.nextLine();

// INSERT Command

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

queue.add(inputLine.split(" ")[1]);

}

// REMOVE Command

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

/\* If a remove command is issued for an

empty queue, then nothing should happen. \*/

queue.poll();

}

// End inputs with an empty string

**if**(inputLine.isEmpty()) {

sc.close();

**break**;

}

}

// Output the string in the middle of the queue

**int** SIZE = queue.size();

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

queue.poll();

}

// Two middle String, output the one nearest the front

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

System.***out***.println(queue.peek());

}

// One middle String, output that middle one.

**else** {

queue.poll();

System.***out***.println(queue.peek());

}

}

}

# Question 4

## Question 4-a

文本, 信件

描述已自动生成

**Big O complexity – O(n2)**

**Input variable -> n**

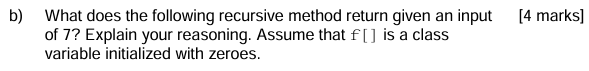
**Loop 1: for(int i = 3; i< n\*20; i++) -> O(n)**

**Loop 2: for(int j = 10; j >= 0; j--) -> O(1)**

**Loop 3: for(int k = j; k < n; k++) -> O(n)**

Considering nested loops, the time complexity is *O*(*n*×1×*n*)=*O*(*n*2).

## Question 4-b



文本

描述已自动生成

***f* [0]=1 (base case)**

***f* [1]=1 (base case)**

***f* [2]=*f* [1]+*f* [0]=1+1=2**

***f* [3]=*f* [2]+*f* [1]=2+1=3**

***f* [4]=*f* [3]+*f* [2]=3+2=5**

***f* [5]=*f* [4]+*f* [3]=5+3=8**

***f* [6]=*f* [5]+*f* [4]=8+5=13**

***f* [7]=*f* [6]+*f* [5]=13+8=21**

## Question 4-c

**文本

描述已自动生成**

Step 1: 32 | 13

|  |  |  |
| --- | --- | --- |
| **(32)10** | **=(00100000)2** |  |
| **(13)10** | **=(00100000)2** | **I** |
|  | **`(00101101)2** | **= (45)10** |

Step 2: 45 >> 2

**(00101101)2 >> 2 = (00001011)2 = (11)10**

Step 3: 11 & 9

|  |  |  |
| --- | --- | --- |
| **(11)10** | **=(00001011)2** |  |
| **(9)10** | **=(00001001)2** | **&** |
|  | **`(00001001)2** | **= (9)10** |

**Therefore, the Java Program outputs 9 when run.**

## Question 4-d

文本

描述已自动生成

**A linked list is a linear data structure consisting of a sequence of elements, each element contains a value and a reference (or a pointer) to the next element in the sequence.**

图示

描述已自动生成

**[Method : delete the tail node of the linked list]**

**public** **void** deleteTail() {

// If the list is empty, nothing to delete

**if** (head == **null**) {

**return**;

}

// If the list has only one node, set head to null

**if** (head.next == **null**) {

head = **null**;

**return**;

}

// Traverse the list to find the tail's previous node

Node currentNode = head;

**while** (currentNode.next.next != **null**) {

currentNode = currentNode.next;

}

// Update the previous node's next pointer to null to remove the tail node

currentNode.next = **null**;

}

## Question 4-e

图表

中度可信度描述已自动生成

1. **Insertion Sort**

**[35 ,86, 10, 58, 26, 96, 38, 52]**

insert 35, sorted list: **[35]**

insert 86, sorted list: **[35, 86]**

insert 10, sorted list: **[10, 35, 86]**

insert 58, sorted list: **[10, 35, 58, 86]**

insert 26, sorted list: **[10, 26, 35, 58, 86]**

insert 96, sorted list: **[10, 26, 35, 58, 86, 96]**

insert 38, sorted list: **[10, 26, 35, 38, 58, 86, 96]**

insert 52, sorted list: **[10, 26, 35, 38, 52, 58, 86, 96]**

**Finally, sorted list: [10, 26, 35, 38, 52, 58, 86, 96]**

1. **Merge Sort**

**[35 ,86, 10, 58, 26, 96, 38, 52]**

* Divide the list: **35, 86, 10, 58, 26, 96, 38, 52**
* Merge pairs and sort: **[35, 86], [10, 58], [26, 96], [38, 52]**
* Merge sublists and sort: **[26, 38, 52, 96], [26, 38, 52, 96]**
* Merge the two sorted sublists: **[10, 26, 35, 38, 52, 58, 86, 96]**

**Finally, sorted list: [10, 26, 35, 38, 52, 58, 86, 96]**

1. **Selection Sort**

**[35 ,86, 10, 58, 26, 96, 38, 52]**

* Select min 10, swap with 35: **[10, 86, 35, 58, 26, 96, 38, 52]**
* Select min 26, swap with 86: **[10, 26, 35, 58, 86, 96, 38, 52]**
* Select min 35, no swap:
* Select min 38, swap with 58, **[10, 26, 35, 38, 86, 96, 58, 52]**
* Select min 52, swap with 86: **[10, 26, 35, 38, 52, 96, 58, 86]**
* Select min 58, swap with 96: **[10, 26, 35, 38, 52, 58, 58, 96]**
* Select min 86, no swap:
* Select min 96, no swap:

**Finally, sorted list: [10, 26, 35, 38, 52, 58, 86, 96]**