# Question 1

文本

描述已自动生成

**import** java.util.Scanner;

**import** java.util.Queue;

**import** java.util.PriorityQueue;

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

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

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

Queue<Student> pq = **new** PriorityQueue<Student>();

// number of students

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

// output rank

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

// Create a new Student class, input its name and score

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

String inputName = sc.nextLine();

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

pq.add(**new** Student(inputName, inputScore));

}

sc.close();

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

pq.poll();

}

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

}

}

// Create a new Class - Student

// It has two properties - name, score

**class** Student **implements** Comparable<Student>{

String name;

**int** score;

**public** Student(String name, **int** score) {

**this**.name = name;

**this**.score = score;

}

//Higher score has priority, when use PriorityQueue

@Override

**public** **int** compareTo(Student other) {

**return** other.score - **this**.score;

}

}

# Question 2

文本

描述已自动生成

文本, 应用程序, 信件, 电子邮件

描述已自动生成

**import** java.util.Scanner;

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

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

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

// input number of coins tosses

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

// input target number of tails

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

sc.close();

//Monte Carlo

**int** N = 1000000;

**int** count = 0;

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

**int** tailToss = 0;

**for** (**int** j = 0; j < TOSSES; j ++) {

/\* create random number: 0 or 1

50% will be 0 -> coin tails

50% will be 1 -> coin head \*/

**int** status = (**int**) (2 \* Math.*random*());

**if**(status == 0) tailToss ++;

}

**if** (tailToss >= TAILS) count++;

}

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

// Round up to the nearest Integer 93.75 -> 94

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

}

}

# Question 3

文本, 信件

描述已自动生成

背景图案

低可信度描述已自动生成

**import** java.util.Scanner;

**import** java.util.Stack;

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

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

Stack<Integer> s = **new** Stack<Integer>();

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

// Number of Commands

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

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

String inputLine = sc.nextLine();

//PUSH Commands

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

**int** inputNum = Integer.*parseInt*(inputLine.split(" ")[1]);

s.add(inputNum);

}

//POP Commands

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

//When stack is empty, nothing happens.

**if**(!s.isEmpty()) s.pop();

}

}

sc.close();

// If stack is empty, then output "empty"

**if**(s.isEmpty()) {

System.***out***.println("empty");

}// output the Integer at the top of the stack

**else** {

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

}

}

}

# Question 4

## Question a

文本

描述已自动生成

**The program runs main function first, it will call compute(100)**

**1) compute(100).**

100>20=> skip if statement

**Print “Running...”,then change line**

**return (compute((100 \* 2) % 53) + 17) = compute(41) + 17**

**2) compute(41).**

41>20=> skip if statement

**Print “Running...” ,then change line**

**return (compute((41 \* 2) % 53) + 17) = compute(29) + 17**

**3) compute(29).**

29>20=> skip if statement

**Print “Running...” ,then change line**

**return (compute((29 \* 2) % 53) + 17) = compute(5) + 17**

**4) compute(5).**

5<20=> run if statement

**return 5 % 7 => compute(5) = 3**

**5) Calling compute(29)**

**compute(29) = compute(5) + 17 = 5 + 17 = 22**

**6) Calling compute(41)**

**compute(41) = compute(5) + 17 = 22 + 17 = 39**

**7) Calling compute(100)**

**compute(100) = compute(41) + 17 = 39 + 17 = 56**

**Therefore, the Java Program outputs**

**`**

**Running**

**Running**

**Running**

**56**

**`**

**when it runs.**

## Question b

文本

描述已自动生成

**The program will print out the equation**

**((4|17)|2)>>1)**

Step 1: 4 & 17

|  |  |  |
| --- | --- | --- |
| **(4)10** | **=(00000100)2** |  |
| **(17)10** | **=(00010001)2** | **|** |
|  | **`(00010101)2** | **= (21)10** |

Step 2: 21 | 2

|  |  |  |
| --- | --- | --- |
| **(21)10** | **=(00010101)2** |  |
| **(2)10** | **=(00000010)2** | **|** |
|  | **`(00010111)2** | **= (23)10** |

Step 5: 23 >> 1

**(00010111)2 >> 1 = (00001011)2 = (11)10**

**Therefore, the Java Program outputs 11 when it runs.**

## \*Question c

图片包含 表格

描述已自动生成

**[33, 63, 90, 68, 21, 96, 38, 28]**

* Divide the list:  **[33], [63], [90], [68], [21], [96], [38], [27]**
* Merge pairs and sort: **[33, 63], [68, 90], [21, 96], [27, 38]**
* Merge sublists and sort: **[33, 63, 68, 90], [21, 27, 38, 96]**
* Merge the two sorted sublists: **[21, 27, 33, 38, 63, 68, 90, 96]**

**Finally, sorted list: [21, 27, 33, 38, 63, 68, 90, 96]**

**The Big O Complexity of mergesort is O(n \* log(n))**

**The Big O Complexity of bubblesort is O(n2)**

**As we can see in the graph, if n is the same value, O(n2) > O(n \* log(n)).**

**Therefore, mergesort use less time, it is more efficient than bubblesort.**

