# Question 1

## Question a

文本

描述已自动生成

1. **Bubble Sort**

**[29, 85, 32, 69, 40, 12]**

Round 1: **[29, 32, 69, 40, 12, 85]**

Round 2: **[29, 32, 40, 12, 69, 85]**

Round 3: **[29, 32, 12, 40, 69, 85]**

Round 4: **[29, 12, 32, 40, 69, 85]**

Round 5: **[12, 29, 32, 40, 69, 85]**

**Finally, sorted list: [12, 29, 32, 40, 69, 85]**

1. **Selection Sort**

**[29, 85, 32, 69, 40, 12]**

Select min 12, swap with 29: **[12, 85, 32, 69, 40, 29]**

Select min 29, swap with 85: **[12, 29, 32, 69, 40, 85]**

Select min 32, no swap:

Select min 40, swap with 69: **[12, 29, 32, 40, 69, 85]**

Select min 69, no swap

Select min 85, no swap

**Finally, sorted list: [12, 29, 32, 40, 69, 85]**

1. **Insertion Sort**

**[29, 85, 32, 69, 40, 12]**

insert 29, sorted list: **[29, 85, 32, 69, 40, 12]**

insert 85, sorted list: **[29, 85, 32, 69, 40, 12]**

insert 32, sorted list: **[29, 32, 85, 69, 40, 12]**

insert 69, sorted list: **[29, 32, 69, 85, 40, 12]**

insert 40, sorted list: **[29, 32, 40, 69, 85, 12]**

insert 12, sorted list: **[12, 29, 32, 40, 69, 85]**

**Finally, sorted list: [12, 29, 32, 40, 69, 85]**

## Question b

文本

描述已自动生成

**The program runs main function first, it will print out the equation**

**((23|45)^(18&26)<<4)**

Step 1: 23 | 45

|  |  |  |
| --- | --- | --- |
| **(23)10** | **=(00010111)2** |  |
| **(45)10** | **=(00101101)2** | **I** |
|  | **`(00111111)2** | **= (63)10** |

Step 2: 18 & 26

|  |  |  |
| --- | --- | --- |
| **(18)10** | **=(00010010)2** |  |
| **(26)10** | **=(00011010)2** | **&** |
|  | **`(00010010)2** | **= (18)10** |

Step 3: 18 << 4

**(00010010)2 << 4 = (100100000)2 = (288)10**

Step 4: 63 ^ 288

|  |  |  |
| --- | --- | --- |
| **(63)10** | **=(000111111)2** |  |
| **(288)10** | **=(100100000)2** | **^** |
|  | **`(100011111)2** | **= (287)10** |

**Therefore, the Java Program outputs 287 when it runs.**

## Question c

文本

描述已自动生成

**The program runs main function first, it will call the function recursion(26)**

**1) Calling recursion(26).**

26 % 9 = 8, 8 > 3 => execute else branch

=> return **26% recursion(26 - 2) + 1**

**2) Calling recursion(24).**

24 % 9 = 6, 6 > 3 => execute else branch

=> return **24% recursion(24 - 2) + 1**

**3) Calling recursion(22).**

22 % 9 = 4, 4 > 3 => execute else branch

=> return **22% recursion(22 - 2) + 1**

**4) Calling recursion(20).**

20 % 9 = 2, 2 < 3 => execute if branch

=> return **8**

**5) Calculating recursion(22).**

**22 % recursion(22 - 2) + 1** = 22 % 8 + 1 = 6 + 1 = 7

=> return **7**

**6) Calculating recursion(24).**

**24 % recursion(24 - 2) + 1** = 24 % 7 + 1 = 3 + 1 = 4

=> return **2**

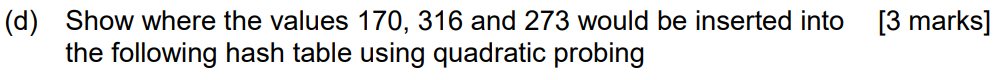
**7) Calculating recursion(26).**

**26 % recursion(26 - 2) + 1** = 26 % 4 + 1 = 2 + 1 = 3

=> return **3**

**Therefore, the Java Program outputs 3 when it runs.**

## Question d



表格

描述已自动生成

**Hash Table size = 11.**

1. **Insert 170.**

170 % 11 = 5, **5** is vacant.

Insert at slot 5.

1. **Insert 316.**

316 % 11 = 8, 8 is occupied.

***Quaduatic Probing*** 8-9-2-**0**

Insert at slot 0.

1. **Insert 273.**

273 % 11 = 9 9 s occupied.

***Quaduatic Probing*** 9-10-**3**

Insert at slot 3.

## Question e

***文本

描述已自动生成***

**Directed Graph**

|  |  |  |  |
| --- | --- | --- | --- |
| **Depth-First Search** | | **Breadth-First Search** | |
| **Events** | **Stack** | **Events** | **Queue** |
| Push A | A | Add A | A |
| Push B | A B | Remove A | - |
| Push C | A B C | Add B | B |
| Pop C | A B | Remove B | - |
| Push D | A B D | Add C | C |
| Push F | A B D F | Add D | C D |
| Push G | A B D F G | Remove C | D |
| Pop G | A B D F | Remove D | - |
| Pop F | A B D | Add F | F |
| Push E | A B D E | Add E | F E |
| Pop E | A B D | Remove F | E |
| Pop D | A B | Remove E | - |
| Pop B | A | Add G | G |
| Pop A | - | Remove G | - |
| **Traverse the Graph** | | | |
| **Stack** | | **Queue** | |
| **A B C D F G E** | | **A B C D F E G** | |

## Question f

文本

中度可信度描述已自动生成

The Big O Complexity of this program is determined by the input **`int n`.**

There are two for loops in this program:

1. Outer Loop **for**(**int** i = n; i > 500; i--)

**It runs ` n – 500 ` times**

1. Inner Loop **for**(**int** j = 6; j < 6\*n; j = j + n){

**It runs ` (6\*n - 6) / n ` times**

**Calculate the Total Iterations:**

**Simplify the expression, ignore constant terms and lower-order terms:**

**Use Big O notation to the Total Iterations:**

**Therefore, the Big O Complexity of the Java Program is O(n).**

## Question g

表格

描述已自动生成文本

描述已自动生成

In this program, highest value was given highest priority, therefore the contents’ adjustment would be:

|  |  |
| --- | --- |
| **Events** | **Priority Queue** |
| Insert 35 | 35 |
| Insert 19 | 35 19 |
| Insert 43 | 43 35 19 |
| Remove | 35 19 |
| Insert 98 | 98 35 19 |
| Remove | 35 19 |
| Remove | 19 |
| Insert 69 | 69 19 |
| Remove | 19 |

# Question 2

文本

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***);

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

sc.close();

**int** distance = 0;

**int** upperNum = input;

**int** lowerNum = input;

// If neither upperNum nor lowerNum is Prime,

distance = distance + 1.

**while**(!*checkPrime*(upperNum) && !*checkPrime*(lowerNum)) {

upperNum ++;

lowerNum --;

distance ++;

}

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

}

**public** **static** **boolean** checkPrime (**int** input) {

**if**(input <= 1) **return** **false**;

**for**(**int** i = 2; i < input; i++) {

**if** (input % i == 0) **return** **false**;

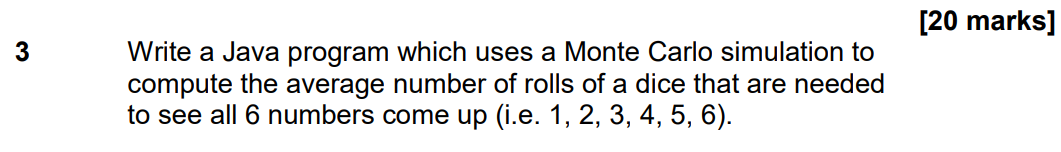
}

**return** **true**;

}

}

# Question 3



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

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

**int** N = 10000000;

**int** sumRolls = 0;

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

// Index 0 - 5 => Dice Number 1 - 6

**boolean** dice[] = {**false**, **false**, **false**,

**false**, **false**, **false**};

**int** count = 0;

**while**(!*checkDice*(dice)) {

**int** roll = (**int**)(Math.*random*() \* 6);

dice[roll] = **true**;

count ++;

}

sumRolls = sumRolls + count;

}

**double** averageNumber = (**double**)sumRolls /(**double**) N;

System.***out***.printf("%.2f\n", averageNumber);

}

// Check if all 6 numbers have come up

**public** **static** **boolean** checkDice (**boolean** dice[]) {

**for**(**boolean** number : dice) {

**if** (number == **false**) **return** **false**;

}

**return** **true**;

}

}

# Question 4

文本, 信件

描述已自动生成

**import** java.util.Scanner;

**import** java.util.List;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.Set;

**import** java.util.HashSet;

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

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

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

List<String> list = **new** ArrayList<String>();

// Input a list of n English words

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

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

String inputLine = sc.nextLine();

list.add(inputLine);

}

// Sort the list

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

@Override

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

**if**(*getUniqueChar*(s1) != *getUniqueChar*(s2)) {

// 1 - Fewer unique Char first

**return** *getUniqueChar*(s1) - *getUniqueChar*(s2);

}

**else** {

// 2 - Alphabetically

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

}

}

});

// Print the output

**for**(String s : list) {

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

}

}

//Use HashSet to get the number of Unique Char in a word

**public** **static** **int** getUniqueChar(String input) {

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

**char**[] charArray = input.toCharArray();

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

s.add(charArray[i]);

}

**int** uniqueChar = s.size();

**return** uniqueChar;

}

}