# Question 1

图形用户界面, 文本, 应用程序

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***);

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

sc.close();

**int** nearestPrime = *findNearestPrime*(number);

**int** distance = Math.*abs*(nearestPrime - number);

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

}

**public** **static** **boolean** isPrime(**int** number) {

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

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

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

}

**return** **true**;

}

**public** **static** **int** findNearestPrime(**int** input) {

**int** largerPrime = input;

**int** smallerPrime = input;

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

**if**(*isPrime*(largerPrime)) **return** largerPrime;

**if**(*isPrime*(smallerPrime)) **return** smallerPrime;

largerPrime++;

smallerPrime--;

}

}

}

# Question 2

文本

描述已自动生成

**import** java.util.Queue;

**import** java.util.PriorityQueue;

**import** java.util.Scanner;

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

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

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

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

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

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

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

pq.add(input); // Input number into PriorityQueue

}

sc.close();

**double** average;

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

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

pq.poll();

}

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

average = (**double**)(pq.poll() + pq.poll()) / 2.0;

}

**else** {

pq.poll();

average = (**double**) pq.poll();

}

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

}

}

/\*

\* Bio O Complexity: O (n \* log(n))

\* PriorityQueue is actually a heap(Binary Tree),

\* insert an element into a tree takes O(logn)

\* There is N element to be inserted, it takes O(n)

\* Therefore, the time complexity is O(n) \* O(logn)

\*/

# Question 3

文本

描述已自动生成

文本

中度可信度描述已自动生成

## Answer 1 – Interface

**import** java.util.Stack;

**import** java.util.Scanner;

**public** **class** Q3\_Interface {

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

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

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

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

String inputLine = sc.nextLine();

**if**(inputLine.toUpperCase().contains("NORTH")) {

s.add("Go North");

}

**if**(inputLine.toUpperCase().contains("SOUTH")) {

s.add("Go South");

}

**if**(inputLine.toUpperCase().contains("WEST")) {

s.add("Go West");

}

**if**(inputLine.toUpperCase().contains("EAST")) {

s.add("Go East");

}

**if**(inputLine.toUpperCase().contains("BACK")) {

s.pop();

}

**if**(inputLine.toUpperCase().contains("ARRIVED")) {

sc.close();

**break**;

}

}

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

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

}

}

**public** **static** String overturn (String direction) {

**switch**(direction) {

**case** "Go North" : **return** "Go South";

**case** "Go South" : **return** "Go North";

**case** "Go West" : **return** "Go East";

**case** "Go East" : **return** "Go West";

**default** : **return** **null**;

}

}

}

## Answer 2 – Full Stack Class

**import** java.util.Scanner;

**public** **class** Q3\_FullStackClass {

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

FullStack s = **new** FullStack(100);

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

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

String inputLine = sc.nextLine();

**if**(inputLine.toUpperCase().contains("NORTH")) {

s.push("Go North");

}

**if**(inputLine.toUpperCase().contains("SOUTH")) {

s.push("Go South");

}

**if**(inputLine.toUpperCase().contains("WEST")) {

s.push("Go West");

}

**if**(inputLine.toUpperCase().contains("EAST")) {

s.push("Go East");

}

**if**(inputLine.toUpperCase().contains("BACK")) {

s.pop();

}

**if**(inputLine.toUpperCase().contains("ARRIVED")) {

sc.close();

**break**;

}

}

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

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

}

}

**public** **static** String overturn (String direction) {

**switch**(direction) {

**case** "Go North" : **return** "Go South";

**case** "Go South" : **return** "Go North";

**case** "Go West" : **return** "Go East";

**case** "Go East" : **return** "Go West";

**default** : **return** **null**;

}

}

}

**class** FullStack{

**private** **int** maxSize; // size of stack array

**private** String[] stackArray;

**private** **int** top; // top of stack

**public** FullStack(**int** s) { // constructor

maxSize = s; // set array size

stackArray = **new** String[maxSize]; // create array

top = -1; // no items yet

}

**public** **void** push(String j) { // nput item on top of stack

top++;

stackArray[top] = j; // increment top, insert item

}

**public** String pop() { // take item from top of stack

**return** stackArray[top--]; //access item, decrement top

}

**public** **boolean** isEmpty() { // true if stack is empty

**return** (top == -1);

}

}

# Question 4

## Question a

图形用户界面, 文本, 应用程序

描述已自动生成

**The program runs main function first, it will call function(83)**

**1) function(83).**

**input++ => input = 84**

84 % 13 == 6, 6 != 2 => skip if statement

**return(function(84+2)-3)**

**2) function(86).**

**input++ => input = 87**

87 % 13 == 9, 9 != 2 => skip if statement

**return(function(87+2)-3)**

**3) function(89).**

**input++ => input = 90**

90 % 13 == 12, 12 != 2 => skip if statement

**return(function(90+2)-3)**

**4) function(92).**

**input++ => input = 93**

93 % 13 == 2, 2 == 2 => run if statement

**return 8 => function(92) = 8**

**5) Calling function(89)**

**function(89) = function(92) – 3 = 8 – 3 = 5**

**6) Calling function(86)**

**function(86) = function(89) – 3 = 5 – 3 = 2**

**7) Calling function(83)**

**function(83) = function(86) – 3 = 2 – 3 = -1**

**Therefore, the Java Program outputs -1 when it runs.**

## Question b

文本

描述已自动生成

**The program will print out the equation**

**(((5|7|16|11)&167)>>1)**

Step 1: 5 & 7

|  |  |  |
| --- | --- | --- |
| **(5)10** | **=(00000101)2** |  |
| **(7)10** | **=(00000111)2** | **|** |
|  | **`(00000111)2** | **= (7)10** |

Step 2: 7 | 16

|  |  |  |
| --- | --- | --- |
| **(7)10** | **=(00000111)2** |  |
| **(16)10** | **=(00010000)2** | **|** |
|  | **`(00010111)2** | **= (23)10** |

Step 3: 23 | 11

|  |  |  |
| --- | --- | --- |
| **(23)10** | **=(00010111)2** |  |
| **(11)10** | **=(00001011)2** | **|** |
|  | **`(00011111)2** | **= (31)10** |

Step 4: 31 | 167

|  |  |  |
| --- | --- | --- |
| **(31)10** | **=(00011111)2** |  |
| **(167)10** | **=(10100111)2** | **&** |
|  | **`(00000111)2** | **= (7)10** |

Step 5: 7 >> 1

**(00000111)2 >> 1 = (00000011)2 = (3)10**

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

## Question c

文本, 信件

描述已自动生成

1. **\* Merge Sort – Big O Complexity = O(n \* log(n))**

**[69, 24, 10, 72, 96, 22, 18, 38]**

* Divide the list: [**69], [24], [10], [72], [96], [22], [18], [38]**
* Merge pairs and sort: **[24, 69], [10, 72], [22, 96], [18, 38]**
* Merge sublists and sort: **[10, 24, 69, 72], [18, 22, 38, 96]**
* Merge the two sorted sublists: **[10, 18, 22, 24, 38, 69, 72, 96]**

**Finally, sorted list: [10, 18, 22, 24, 38, 69, 72, 96]**

1. **Insertion Sort - Big O Complexity = O(n2)**

**[69, 24, 10, 72, 96, 22, 18, 38]**

insert 69, sorted list: **[69, 24, 10, 72, 96, 22, 18, 38]**

insert 24, sorted list: **[24, 69, 10, 72, 96, 22, 18, 38]**

insert 10, sorted list: **[10, 24, 69, 72, 96, 22, 18, 38]**

insert 72, sorted list: **[10, 24, 69, 72, 96, 22, 18, 38]**

insert 96, sorted list: **[10, 24, 69, 72, 96, 22, 18, 38]**

insert 22, sorted list: **[10, 22, 24, 69, 72, 96, 18, 38]**

insert 18, sorted list: **[10, 18, 22, 24, 69, 72, 96, 38]**

insert 38, sorted list: **[10, 18, 22, 24, 38, 69, 72, 96]**

**Finally, sorted list: [10, 18, 22, 24, 38, 69, 72, 9**

1. **Selection Sort - Big O Complexity = O(n2)**

**[69, 24, 10, 72, 96, 22, 18, 38]**

* Select min 10, swap with 69: **[10, 24, 69, 72, 96, 22, 18, 38]**
* Select min 18, swap with 86: **[10, 18, 69, 72, 96, 22, 24, 38]**
* Select min 22, swap with 69: **[10, 18, 22, 72, 96, 69, 24, 38]**
* Select min 24, swap with 72, **[10, 18, 22, 24, 96, 69, 72, 38]**
* Select min 38, swap with 96: **[10, 18, 22, 24, 38, 69, 72, 96]**
* Select min 69, no swap:
* Select min 72, no swap:
* Select min 96, no swap:

**Finally, sorted list: [10, 18, 22, 24, 38, 69, 72, 96]**