Question 1

[25 marks]

1 (a) **Problem Statement** [25 marks]

The goal is to read in a number and output the distance to the nearest prime, either before it or after it. If the number itself happens to be prime, then output 0.

Sample Input
10

Sample Output
1

```
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

[25 marks]

2 (a) **Problem Statement** [25 marks]

The goal is to read in a list of integers into an array and output the one which is the median, that is, the one which if you put them all in order would be in the middle. If there are two in the middle then take the average of those two. For example the median of 4, 8, 2, 3 and 5 is 4, because if you put them all in order 4 would be the middle value. The median of 2, 7, 4, and 9 is 5.5 because 5.5 is the average of 4 and 7.
State the **Big O complexity** of your algorithm, and explain your reasoning clearly.

Sample Input
7
15 18 3 2 -5 6 2

Sample Output
3.0

```
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

[25 marks]

3 (a) **Problem Statement** [25 marks]

You are given a set of instructions to get from a home location to a destination location. You have to give the instructions which begin at the destination location and return to the home location. In other words, you need to reverse the directions. The possible set of directions is "Go North", "Go South", "Go East" and "Go West". Another possible instruction is "Go Back" which means to undo the latest step. The instruction "Arrived" means the sequence of directions is complete.
The way to solve this problem is to push all the instructions onto the stack, pop when you get "Go Back", and then pop them all off at the end (flipping North <-> South and East <-> West) to have a reversed set of instructions to get you home.

You should provide a full stack class as part of your answer.

Sample Input
Go North
Go North
Go West
Go Back
Arrived

Sample Output
Go South
Go South

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);
```

CS210-2019-Autumn

```
        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;
        }
    }
}
```

CS210-2019-Autumn

```
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 String peek() { // peek at top of stack
      return stackArray[top];
   }

   public boolean isEmpty() { // true if stack is empty
      return (top == -1);
   }

   public boolean isFull() { // true if stack is full
      return (top == maxSize - 1);
   }

   public void makeEmpty() { // empty stack
      top = -1;
   }
}
```

CS210-2019-Autumn

Question 4

Question a

[25 marks]

4 (a) Identify the output that the following Java code produces and explain your reasoning clearly. [8 marks]

```
public class Recursion{

 public static void main(String[] args){
    System.out.println(function(83));
 }

 public static int function(int input){
      input++;
      if(input%13==2){
          return 8;
      }
      return (function(input+2)-3);
 }
}
```

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

CS210-2019-Autumn

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

(b) Identify the output that the following Java code produces and explain your reasoning clearly. [8 marks]

```
public class BitManipulation{

   public static void main(String[] args){
      System.out.println(((5|7|16|11)&167)>>1);
   }
}
```

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.

CS210-2019-Autumn

Question c

(c) Show how the following numbers would be sorted by the following algorithms, and identify the Big O complexity of each [9 marks]

i) Mergesort
ii) Insertion sort
iii) Selection sort

69 24 10 72 96 22 18 38

i) * Merge Sort

[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]

ii) Insertion Sort

[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, 96]

iii) Selection Sort

[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]

CS210-2019-Autumn