# Question 1

**[20 marks]**

1 Write a Java program given the following specification and provide comments which explain how your algorithm works.

**Problem Statement**
The goal is to read in 3 numbers and output the one which is most different from the other two (i.e. the one with the greatest difference from the nearest number to it). If there is no single outlier then output "NA".

**Input Format**
Each line contains *A*, *B*, and *C* separated by a space.

**Output Format**
An integer representing the outlier of the three, or "NA" if there is no outlier.

**Constraints**
$-1000 \leq A, B, C \leq 1000$

**Sample Input**
6 3 9

**Sample Output**
NA

```
import java.util.Scanner;

public class Q1 {
    public static void main (String args[]) {
        Scanner sc = new Scanner (System.in);
        // Input three number
        int num1 = sc.nextInt();
        int num2 = sc.nextInt();
        int num3 = sc.nextInt();
        sc.close();

        int mostDiff = 0;
        boolean noDiff = false;
        // Calculate the difference between each 2 numbers
        int diff12 = Math.abs(num1 - num2);
        int diff13 = Math.abs(num1 - num3);
        int diff23 = Math.abs(num2 - num3);
```

```
        // Find the most different number
        if(diff12 > diff23 && diff13 > diff23) mostDiff = num1;
        else if (diff12 > diff13 && diff23 > diff13) mostDiff = num2;
        else if (diff23 > diff12 && diff13 > diff12) mostDiff = num3;
        // All the difference are same
        else noDiff = true;

        if(noDiff) {
            System.out.println("NA");
        }
        else {
            System.out.println(mostDiff);
        }
    }
}
```

# CS210-2017-Autumn

# Question 2

**[20 marks]**

**2** Write a Java program given the following specification and provide comments which explain how your algorithm works. Estimate the **Big O complexity** of your program and explain your reasoning clearly.

**Problem Statement**
The goal is to read in a number of fair coin tosses, and a target number of heads, and output the probability that the target number of heads in a row will be tossed at some point in the sequence, rounded to the nearest percent.

**Input Format**
An integer *N* for the number of coin tosses, followed on the next line by an integer *H* for the target number of heads.

**Output Format**
An integer from 0 to 100 representing the percentage probability that *H* heads in a row will be observed at some point during *N* tosses of a fair coin.

**Constraints**
$0 \leq N \leq 1000$
$0 \leq H \leq 100$

**Sample Input**
5
3

**Sample Output**
25

```
import java.util.Scanner;

public class Q2 {
   public static void main (String args[]) {
      Scanner sc = new Scanner(System.in);
      // Input number of coin tosses
      int N = Integer.parseInt(sc.nextLine());

      // Input target number of heads
      int H = Integer.parseInt(sc.nextLine());
      sc.close();
```

# CS210-2017-Autumn

```
        //Monte Carlo Simulation
        int ALL = 1000000;
        int count = 0;

        for(int i = 0; i < ALL ; i++) {
           int consecutiveHeads = 0;

           for (int j = 0; j < N; j ++) {
              // If toss = 0 - > Tail
              // If toss = 1 - > Head
              int toss = (int) (2 * Math.random());

              //If get a HEAD, accumulate the times
              if(toss == 1) {
                 consecutiveHeads++;
              }
              //If get a TAIL, set times back to 0
              else {
                 consecutiveHeads = 0;
              }

              // Observe H heads in a row
              if(consecutiveHeads == H) {
                 count ++ ;
              }
           }
        }
        double p = (double)(count * 100)/ (double) ALL;
        System.out.println(Math.round(p));
     }
}
```

**[Big O complexity] - O(n^2)**

I use two for loops in the program,
Outer loop is related to ALL -- the times of Monte Carlo Simulation
Inner loop is related to N -- Input number of coin tosses
So the complexity is O(n^2))

# CS210-2017-Autumn

# Question 3

**[20 marks]**

**3** Write a Java program given the following specification and provide comments which explain how your algorithm works. Estimate the **Big O complexity** of your program and explain your reasoning clearly.

**Problem Statement**
The goal is to read in a list of students and exam scores into an array and output the name of the student with the median score, that is, the one which if you put them all in order would be in the middle. For example the median of 40, 80, 20, 30 and 50 is 40, because if you put them all in order 40 would be the middle value, so output the name of the student who got 40. If there is an even number of students, then output the name of the middle two that comes first in alphabetical order.

**Input Format**
The first line contains *N*, the number of inputs. This is followed by *N* pairs of names and integers separated by a space, each on a separate line.

**Output Format**
The name of the student who represents the median.

**Constraints**
$1 \leq N \leq 1000$
$0 \leq score \leq 100$

**Sample Input**
4
Paul 18
Claire 94
David 34
John 25

**Sample Output**
David

```
import java.util.Scanner;
import java.util.Queue;
import java.util.PriorityQueue;
import java.util.Comparator;


public class Q3 {
   public static void main (String args[]) {
```

# CS210-2017-Autumn

```
//Create a priority queue that stores the information of students.
      Queue<Student> pq = new PriorityQueue<Student>(new
      Comparator<Student>() {
         @Override
         public int compare (Student s1, Student s2) {
            // Compare by the score first, low score comes first
            if(s1.score != s2.score) {
               return s1.score - s2.score;
            }
            //Same score->compare name in alphabetical order
            else {
               return s1.name.compareTo(s2.name);
            }
         }
      });

      Scanner sc = new Scanner (System.in);
      // Number of inputs
      int N = Integer.parseInt(sc.nextLine());

      // Input student information
      for (int i = 0; i < N ; i++) {
         String inputLine = sc.nextLine();
         String studentName = inputLine.split(" ")[0];
         int studentScore = Integer.parseInt(inputLine
         .split(" ")[1]);
         pq.add(new Student(studentName, studentScore));
      }

      //Get the median students
      int SIZE = pq.size();
      for (int i = 0; i < SIZE / 2 - 1; i++) {
         pq.poll();
      }


      //If students number is odd, get the median one
      if(SIZE % 2 == 1) {
         pq.poll();
         System.out.println(pq.peek().name);
      }
      /*If students number is odd, get the median two first
        And then select the name by alphabetical order */
```

```
        else {
            Student m1 = pq.poll();
            Student m2 = pq.poll();
            if(m1.name.compareTo(m2.name) < 0) {
                System.out.println(m1.name);
            }
            else {
                System.out.println(m2.name);
            }
        }
    }

}

//Create a new Class - Student
//it stores name and score
class Student{
    String name;
    int score;

    public Student(String name, int score) {
        this.name = name;
        this.score = score;
    }
}
```

**[Big O complexity] - O(n * log(n))**

I use two for loops in the program,First loop is to insert students information into a priority queue.
The loop is related to N -- Number of student inputs
The time complexity of inserting a student into priority queue is O(log(n)).
Second loop is related to N -- Number of student inputs
So the complexity is O(n) * O(logn) + O(n) = O(n * log(n))

# CS210-2017-Autumn

# Question 4

**[20 marks]**

**4** Write a Java program given the following specification and provide comments which explain how your algorithm works. Estimate the **Big O complexity** of your program and explain your reasoning clearly.

**Problem Statement**
Manipulate a queue according to the given insert and remove commands and then output the string that is in the middle of the queue. If there is an even number of strings in the queue, thus two middle strings, output the one which is nearest the front. If a remove command is issued for an empty queue then nothing should happen.

**Input Format**
A series of lines involving either INSERT or REMOVE commands. The command INSERT is followed by a space and then the string to insert (e.g. INSERT hello).

**Output Format**
Output the string that is in the middle of the queue following the given commands. If there are two middle strings, output the one nearest the front. If the queue is empty output "empty".

**Constraints**
1≤#(commands)≤20
1≤length(string)≤20

**Sample Input**
INSERT this
INSERT is
INSERT how
INSERT to
REMOVE
INSERT do
REMOVE
INSERT it

**Sample Output**
to

```
import java.util.Queue;
import java.util.LinkedList;
import java.util.Scanner;

public class Q4 {
   public static void main (String args[]) {
```

```
        Queue<String> commands = new LinkedList<String>();
        Scanner sc = new Scanner(System.in);
        // Input all the commands!
        while(true) {
            String inputLine = sc.nextLine();
            // If input empty String, close the scanner.
            if(inputLine.isEmpty()) {
                sc.close();
                break;
            }
//If input includes "INSERT" at first, add element to the queue.
            if(inputLine.split(" ")[0].toUpperCase().equals("INSERT"))
            {
                commands.add(inputLine.split(" ")[1]);
            }

            //If input is "REMOVE", remove the first element.
            if(inputLine.toUpperCase().equals("REMOVE")) {
            //  If a remove command is issued for an empty queue then
            nothing should happen.
                if(!commands.isEmpty())   commands.remove();
            }
        }

        // Get the middle string of priority queue.
        int SIZE = commands.size();
        String middleElement;
        for(int i = 0; i < SIZE / 2 - 1; i++) {
            commands.poll();
        }

        // Even number -> the middle two -> nearest the front
        if(SIZE % 2 == 0) {
            middleElement = commands.peek();
        }
        // Odd number -> the middle one
        else {
            commands.poll();
            middleElement = commands.peek();
        }
        System.out.println(middleElement);
    }
}
```

# CS210-2017-Autumn

**[Big O complexity] - O(n)**

I use two for loops in the program,
First loop is to insert Command Lines,
it is related to the number of commands.
Second line is to get the median element in the queue,
it is related to the size of queue.
So the complexity is O(n) + O(n) = O(n)

# CS210-2017-Autumn