AHSAN SAJJAD
BSE-4B

ASSIGNMENT NO 01
DESIGN AND ANALYSIS OF ALGORITHM

1. Write an algorithm in pseudo-code for converting your name along with your favorite sports personality name (your name and favorite sports personality name) without spaces into roman letters. Firstly, the names should be converted into numbers through ASCII codes then covert that name into roman letters and print as an outcome.

For example, if your name is "Ahmed Khan" and your favorite personality name is "Roger Federer", the ASCII code for 'A' is 65; 'h' is 104 and so on. The result will be (65+104+109+101+100+75+104+97+110)
+(82+111+103+101+114+70+101+100+101+114+101+114) = 2077

Recall the following conversion:

Letter	Value	Letter	Value
I	1	X	10
II	2	XL	40
III	3	L	50
IV	4	C	100
V	5	D	500
IX	9	M	1000

Answer:

```
Console.WriteLine("ENTER NAME :  ")
string name = Console.ReadLine();
foreach( char c in name)
{
   i=i+(System.Convert.ToInt32(c));
              }
Console.WriteLine("ENTER YOUR FAVOURITE PERSONALITY NAME : ")
String fav= Console.ReadLine();
foreach( char c in fav)
{
   j=j+(System.Convert.ToInt32(c));
}
Console.WriteLine("Result is: "+(i+j))

String roman= IntToRoman(i+j);

Console.WriteLine(" the Roman are  !!!!!!!");     }

   public string IntToRoman(int num) {

      string romanResult = "";

      Dictionary < string, int > romanNumDict= new()

 {
```

```
 { "I", 1 },
{"II",2},
{"III",3},
{ "IV", 4},
 { "V", 5},
{ "VI", 6},
{ "VII", 7},
{ "VIII", 8},
 { "IX", 9 },
{ "X", 10 },
{ "XL", 40 },
{ "L", 50},
{ "C",100 },
{ "D", 500 },
{ "M", 1000 }
}
      foreach(var item in romanNumDict.Reverse()) {
         if (num <= 0) break;
         while (num >= item.Value)
{
            romanResult += item.Key;
            num -= item.Value;
}
}
      return romanResult;
}
```

The following program convert any name into letters.

2. Write an algorithm in pseudo-code to find square root of a number using Babylonian squareroot method. Suppose you are given any positive number S. To find the square root of S, do the following:

1. Make an initial guess. Guess any positive number x0.

2. Improve the guess. Apply the formula x1 = (x0 + S / x0) / 2. The number x1 is a better approximation to sqrt(S)

3. Iterate until convergence. Apply the formula xn+1 = (xn + S / xn) / 2 until the process converges. Convergence is achieved when the digits of xn+1 and xn agree to as many decimal places as you desire.

Let's use this algorithm to compute the square root of S = 20 to at least two decimal places.

1. An initial guess is x0 = 10.

2. Apply the formula: x1 = (10 + 20/10)/2 = 6. The number 6 is a better approximation to sqrt(20).

3. Apply the formula again to obtain x2 = (6 + 20/6)/2 = 4.66667.

The next iterations are x3 = 4.47619 and x4 = 4.47214. Because x3 and x4 agree to two decimal places, the algorithm ends after four iterations. An estimate for sqrt(20) is 4.47214.

ANSWER:

Step 1: Take input of the number S whose square root needs to be found: **input S**
Step 2: Set the initial guess for square root: let x0 = S / 2
Step 3: Set the desired level of precision: let precision = 0.0001
Step 4: Iterate until

```
while (true){
let xn = (x0 + S / x0) / 2
if abs(xn - x0) < precision
{
  Break
}
x0 = xn
}
```

Step 5: Print the final approximation of the square root write("SQUARE ROOT of ", S, "is approximately", xn)

The square root of 20 is approximately 4.472135

3. Consider the following version of an important algorithm find time complexity for this algorithm

1. ALGORITHM GE(A[0..n − 1, 0..n])

//Input: An n × (n + 1) matrix A[0..n − 1, 0..n] of real numbers

for i ←0 to n − 2 do

for j ←i + 1 to n − 1 do

for k←i to n do

A[j, k]←A[j, k]− A[i, k] * A[j, i]/ A[i, i]

ANSWER:

The outermost loop runs n-1 times. For each iteration of the outer loop, the second loop runs n-i-1 times, and the third loop runs n-i times. Inside the innermost loop, there are two subtractions, one multiplication, and one division. In detail given below

The outer loop iterates from i=0 to i=n-2, which executes n-1 times.

The second loop iterates from j=i+1 to j=n-1, which executes n-i-1 times in each iteration of the outer loop. The total number of iterations for the second loop can be calculated as follows:

Σ(n-i-1) from i=0 to n-2

= (n-1) + (n-2) + ... + 1

= n(n-1)/2

The third loop iterates from k=i to k=n, which executes n-i times in each iteration of the second loop. The total number of iterations for the third loop can be calculated as follows:

Σ(n-i) from i=0 to n-2

= n-1 + n-2 + ... + 1

= n(n-1)/2

Therefore, the total number of iterations for the entire algorithm can be calculated as follows:

n-1 * n(n-1)/2 * n(n-1)/2

= O(n^3)

Therefore, the time complexity of the algorithm is O(n^3), which means that the algorithm takes polynomial time in terms of the input size n.

2.

ALGORITHM *BruteForceClosestPoints*(P)

```
//Input: A list P of n (n ≥ 2) points P1 = (x1, y1), . . . , Pn = (xn, yn)
//Output: Indices index1 and index2 of the closest pair of points
dmin ← ∞
for i ← 1 to n − 1 do
    for j ← i + 1 to n do
        d ← sqrt((xi − xj)^2 + (yi − yj)^2) //sqrt is the square root function
        if d < dmin
            dmin ← d; index1 ← i; index2 ← j
return index1, index2
```

ANSWER:

The outer loop iterates from i = 1 to i = n - 1, which executes n - 1 times.

The second loop iterates from j = i + 1 to j = n, which executes n - i times in each iteration of the outer loop.

Within the nested loops, there is a distance calculation that takes constant time.

Therefore, the total number of distance calculations can be calculated as follows:

Σ(n-i) from i=1 to n-1

= (n-1) + (n-2) + ... + 1

= n(n-1)/2

Each distance calculation requires a constant amount of time, so the total time complexity of the algorithm can be expressed as:

Therefore, the Brute Force Closest Points algorithm takes quadratic time in terms of the input size n.

Time complexity of this algorithm is **O(n^2).**

Because of the nested loop and the time complexity of other line of code is constant.