**Name: Muhammad Shoaib Akhter Qadri**

**Enrollment No: 02-131212-009**

**Course: DAA**

**Date: 29/03/2023**

# Question # 1: [CLO3:1.5]

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

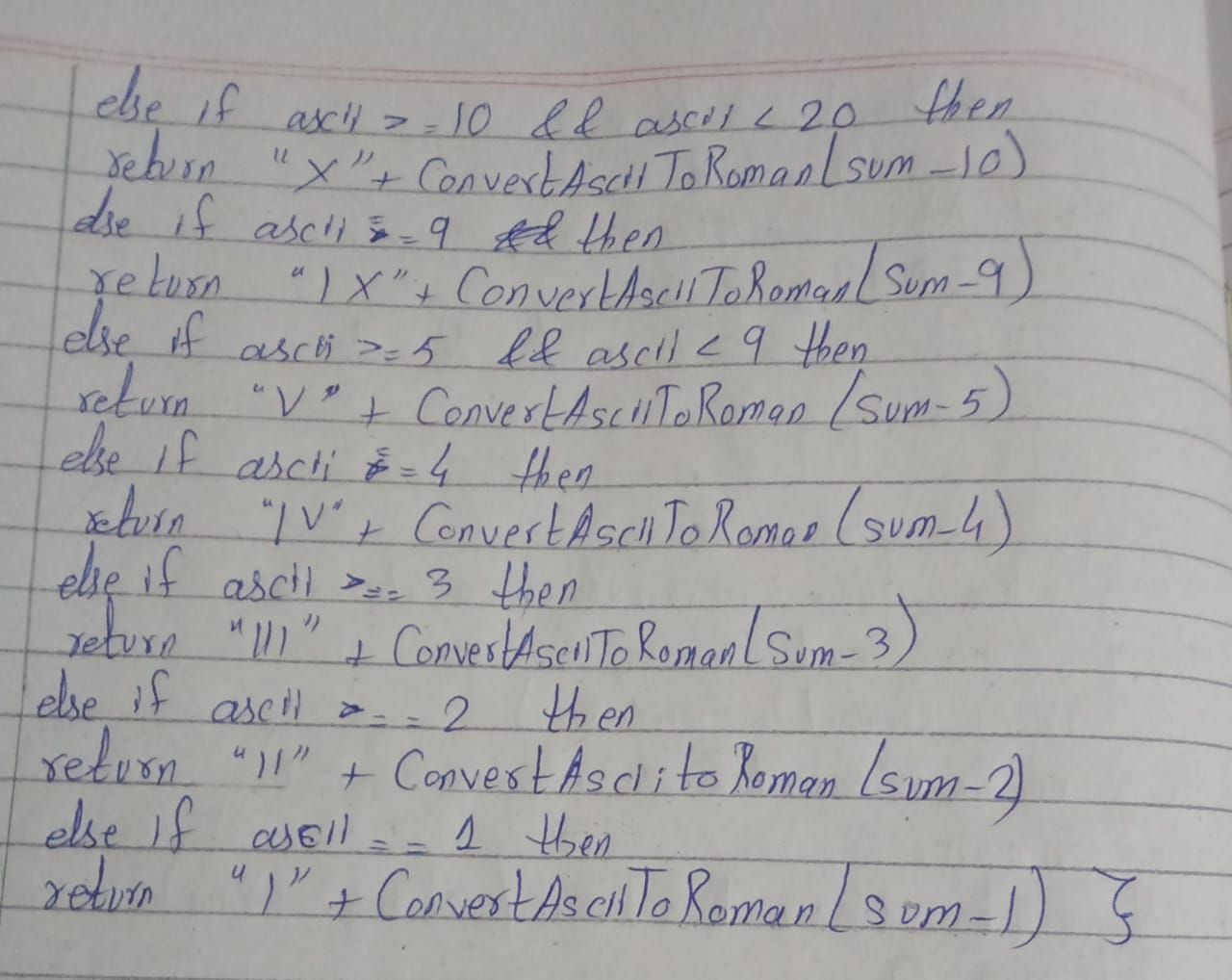
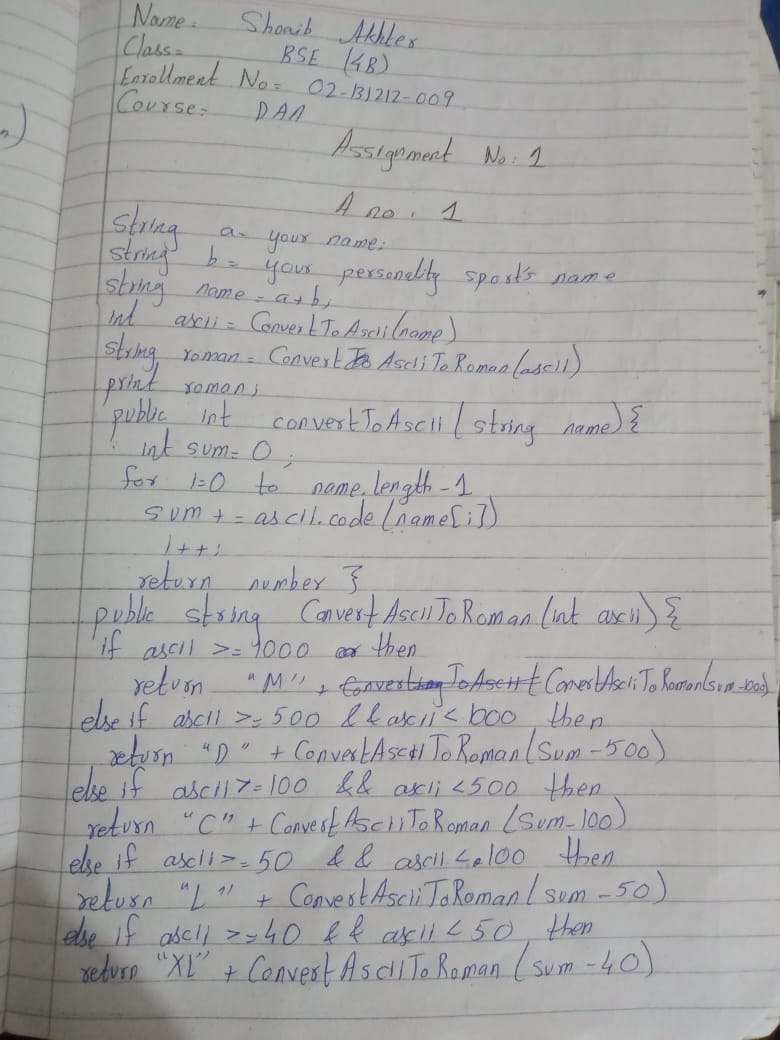
+101+114) = 2077

Recall the following conversion:

Table

Description automatically generated

**Solution:**



# Question # 2: [CLO3:1.5]

Write an algorithm in pseudo-code to find square root of a number using Babylonian square- root 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.

**Solution:**

1. Input a positive number S

2. Set an initial guess x0

3. Repeat until convergence:

a. Compute the next guess, xn+1 = (xn + S/xn) / 2

b. Check for convergence: if |xn+1 - xn| < tolerance, exit loop

c. Set xn = xn+1

4. Output the final guess as the square root of S

**Explanation:**

* The algorithm takes in a positive number S, which is the number we want to find the square root of.
* An initial guess, x0, is set.
* The algorithm iteratively computes better and better approximations of the square root of S by applying the Babylonian formula: xn+1 = (xn + S/xn) / 2.
* The algorithm checks for convergence by computing the absolute difference between xn+1 and xn and comparing it to a tolerance value. If the absolute difference is less than the tolerance value, the algorithm exits the loop.
* Once the algorithm has converged, the final guess, xn+1, is output as the square root of S.

# Question # 3: [CLO4:2]

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**

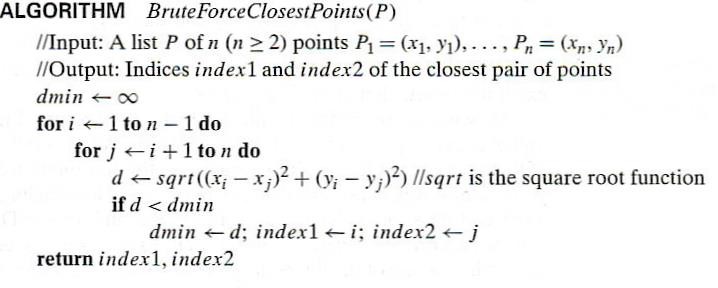
**for** *k*←*i* **to** *n* **do**

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

**Text, letter

Description automatically generated**

**2.**



**Solution:**

* The algorithm takes a list P of n (n>=2) points as input.
* The outer loop starts at i=1 and runs until i=n-1. This loop has n-1 iterations.
* For each iteration of the outer loop, the inner loop starts at j=i+1 and runs until j=n. This loop has ( n-i )iterations.
* Inside the inner loop, the algorithm calculates the Euclidean distance between the pair of points (xi,yj) and (xj,yj) using the formula d = sqrt((xi-xj)2 + (yi-yj)2). This is a constant time operation.
* The algorithm performs a constant time comparison and assignment operation for updating the minimum distance and the indices of the closest pair of points if a smaller distance is found.
* The algorithm returns the indices index1 and index2 of the closest pair of points.

Therefore, the time complexity of the algorithm can be calculated as follows:

T(n) = Summation[(n-i) \* C] from i=1 to n-1, where C is the constant time required for the operations inside the inner loop.

**Operations:**

T(n) = C \* [n(n-1)/2 - (1+2+...+(n-1))]

T(n) = C \* [n(n-1)/2 - (n(n-1)/2)/2]

T(n) = C \* [n(n-1)/4]

T(n) = O(n2)

Therefore, the time complexity of the BruteForceClosestPoints algorithm is O(n2) as it has to iterate over all n(n-1)/2 pairs of points.