**Ex.3**

baseNumber <- 1

while baseNumber\* baseNumber <= parameter

baseNumber <- baseNumber +1

else

answer <- baseNumber -1

**Ex.5**

ADDMATRICES(X,Y)

answer <- [[0,0],[0,0]]

for n in length(X) //to go through rows

for m in length(X[0]) //to go through columns

answer[n][m] <- X[n][m] + Y[n][m]

SUBMATRICES(X,Y)

answer <- [[0,0],[0,0]]

for n in length(X) //to go through rows

for m in length(X[0]) //to go through columns

answer[n][m] <- X[n][m] + Y[n][m]

MULTMATRICES(X,Y)

answer <- [[0,0],[0,0]]

for n in length(X) //to go through rows

for m in length(X[0]) //to go through columns

for l in length(Y)

answer[n][m] <- X[n][m] + Y[n][m]

//A= N-M=N-(L+L)

L=addMatrices(B,C)

M=addMatrices(L,L)

N=multMatrices(B,C)

A=subMatrices(N,M)

//run time, due to triple nested loop, is O(n^3)

**Ex.6**

REVERSESENTECE(sentence)

sentence <- sentence.split

length <- len(sentence)

lastPostion <- length – 1

iterations <- length//2

for n in length(0, iterations)

replacement <- sentence[lastPostion]

sentence[lastPostion] <- sentence[n]

sentence[n] = replacement

lastPostion <- lastPostion - 1

**Ex.7**

PRIMECHECK(VAR, DIVISOR)

divisor <- divisor – 1 //Divisor is always smaller than dividend and we want it to go lower with each iteration

if divisor = 0 or 1 //when during iteration divisor reaches 1 we know that nothing earlier able to divide our variable

and is divisor is 0 it means variable was 1 which is a prime number

“this is prime number”

else if var%divisor = 0 //if our variable is dividable by divisor without any reminder it means it’s not a prime number

“this is not prime number”

else

return primecheck(var, divisor)

**Ex. 8**

VOWELSREMOVAL(INPT)

If not inpt

return inpt

elif inpt[0] in vowelsString

return vowelsRemoval(inpt[1:])

else

return inpt[0] + vowelsRemoval(inpt[1:])

**Ex. 9**

BINARYSEARCH(LST, X, Y)

if length(lst) = 0 //safe case

return False

else

middle =length(lst)//2 //amount of iterations

if lst[middle]>x and lst[middle]<y

return True

else

if lst[middle]<x

return binarySearch(lst[middle+1:],x,y) //if lower number is bigger than middle value of the list higher is being recursively called

else

return binarySearch(lst[:middle],x,y) //opposite to comment above