Name : Sabarivasan V
Class : CSE - B
Reg no : 205001085

Assignment - 6

Qn 1: Strassen Multiplication of Matrices
Code:

```
import numpy as np
import matplotlib.pyplot as plt
import math

steps = 1

def split(matrix):
  global steps
  row, col = matrix.shape
  row2, col2 = row//2, col//2
  steps+=row*row
  return matrix[:row2, :col2], matrix[:row2, col2:], matrix[row2:, :col2],
matrix[row2:, col2:]

def strassen7(x, y):
  global steps
  if len(x) == 1:
    return x * y
  a, b, c, d = split(x)
  e, f, g, h = split(y)
  p1 = strassen7(a, f - h)
  p2 = strassen7(a + b, h)
  p3 = strassen7(c + d, e)
  p4 = strassen7(d, g - e)
  p5 = strassen7(a + d, e + h)
  p6 = strassen7(b - d, g + h)
  p7 = strassen7(a - c, e + f)

  c11 = p5 + p4 - p2 + p6
  c12 = p1 + p2
  c21 = p3 + p4
  c22 = p1 + p5 - p3 - p7
  steps+=7
```

```
  c = np.vstack((np.hstack((c11, c12)), np.hstack((c21, c22))))
  return c

def strassen(x, y):
  global steps
  if len(x) == 1:
    return x * y
  a, b, c, d = split(x)
  e, f, g, h = split(y)

  p1 = strassen(a,e)
  p2 = strassen(b,g)
  p3 = strassen(a,f)
  p4 = strassen(b,h)
  p5 = strassen(c,e)
  p6 = strassen(d,g)
  p7 = strassen(c,f)
  p8 = strassen(d,h)

  c11 = p1+p2
  c12 = p3+p4
  c21 = p5+p6
  c22 = p7+p8
  steps+=8

  c = np.vstack((np.hstack((c11, c12)), np.hstack((c21, c22))))
  return c

mat2 = [[1 for i in range(2)] for j in range(2)]
mat4 = [[1 for i in range(4)] for j in range(4)]
mat8 = [[1 for i in range(8)] for j in range(8)]
mat16 = [[1 for i in range(16)] for j in range(16)]

mat2 = np.array(mat2)
mat4 = np.array(mat4)
mat8 = np.array(mat8)
mat16 = np.array(mat16)

x = [i for i in [2,4,8,16]]
```

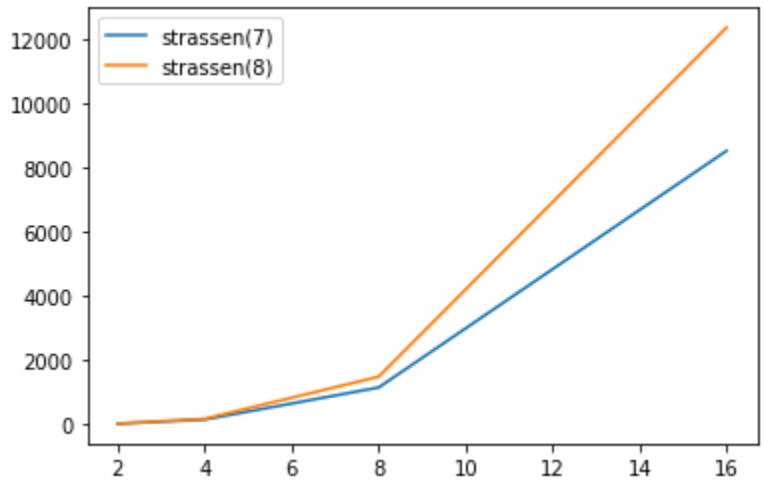
```
y1 = []
y2 = []

mat = [mat2,mat4,mat8,mat16]

for i in range(4):
  strassen7(mat[i],mat[i])
  y1.append(steps)
  steps=1
  strassen(mat[i],mat[i])
  y2.append(steps)
  steps=1


plt.plot(x,y1,label="strassen(7)")
plt.plot(x,y2,label="strassen(8)")
plt.legend()
plt.show()
```

Output:



Qn 2: Quick Sort

Code:

```
# Quick sort in Python
steps = 0
# function to find the partition position
def partition(array, low, high):
  global steps
  # choose the rightmost element as pivot
```

```
  pivot = array[high]

  # pointer for greater element
  i = low - 1

  # traverse through all elements
  # compare each element with pivot
  for j in range(low, high):
    if array[j] <= pivot:
      # if element smaller than pivot is found
      # swap it with the greater element pointed by i
      i = i + 1
      steps+=1

      # swapping element at i with element at j
      (array[i], array[j]) = (array[j], array[i])

  # swap the pivot element with the greater element specified by i
  (array[i + 1], array[high]) = (array[high], array[i + 1])
  steps+=1
  # return the position from where partition is done
  return i + 1

# function to perform quicksort
def quickSort(array, low, high):
  if low < high:

    # find pivot element such that
    # element smaller than pivot are on the left
    # element greater than pivot are on the right
    pi = partition(array, low, high)

    # recursive call on the left of pivot
    quickSort(array, low, pi - 1)

    # recursive call on the right of pivot
    quickSort(array, pi + 1, high)


def gen_arrays():
```

```
  global steps
  x=[]
  y=[]
  for i in range(1,101):
    arr = np.random.randint(0,1000,size=(i))
    quickSort(arr,0,len(arr)-1)
    y.append(i)
    x.append(steps)
    steps = 0
  plt.plot(y,x)
  plt.show()

gen_arrays()
```

Output: