Write a python program to perform matrix multiplication

In [1]:
```
matrix1 = [[1, 2], [3, 4]]
matrix2 = [[5, 6], [7, 8]]
result_matrix = [[0, 0], [0, 0]]

for i in range(len(matrix1)):
   for j in range(len(matrix2[0])):
       for k in range(len(matrix2)):
           result_matrix[i][j] += matrix1[i][k] * matrix2[k][j]
print(result_matrix)
```

```
[[19, 22], [43, 50]]
```

Write a python program for implementing Strassen's Matrix multiplication using Divide and Conquer method.

In [2]:
```
def strassen_matrix_multiply(matrix1, matrix2):
    n = len(matrix1)

    # Base case: if the matrices are 1x1
    if n == 1:
        return [[matrix1[0][0] * matrix2[0][0]]]

    # Split matrices into quadrants
    size = n // 2
    A = [matrix1[i][:size] for i in range(size)]
    B = [matrix1[i][size:] for i in range(size)]
    C = [matrix1[i][:size] for i in range(size, n)]
    D = [matrix1[i][size:] for i in range(size, n)]
    E = [matrix2[i][:size] for i in range(size)]
    F = [matrix2[i][size:] for i in range(size)]
    G = [matrix2[i][:size] for i in range(size, n)]
    H = [matrix2[i][size:] for i in range(size, n)]

    # Recursive steps
    P1 = strassen_matrix_multiply(A, sub(F, H))
    P2 = strassen_matrix_multiply(add(A, B), H)
    P3 = strassen_matrix_multiply(add(C, D), E)
    P4 = strassen_matrix_multiply(D, sub(G, E))
    P5 = strassen_matrix_multiply(add(A, D), add(E, H))
    P6 = strassen_matrix_multiply(sub(B, D), add(G, H))
    P7 = strassen_matrix_multiply(sub(A, C), add(E, F))

    # Compute the result matrix
    result_matrix = [[0 for _ in range(n)] for _ in range(n)]
    for i in range(size):
        for j in range(size):
            result_matrix[i][j] = P5[i][j] + P4[i][j] - P2[i][j] + P6[i][j]
            result_matrix[i][j+size] = P1[i][j] + P2[i][j]
            result_matrix[i+size][j] = P3[i][j] + P4[i][j]
            result_matrix[i+size][j+size] = P5[i][j] + P1[i][j] - P3[i][j] - P7[i][j]

    return result_matrix


# Helper functions to add and subtract matrices
def add(matrix1, matrix2):
    return [[matrix1[i][j]+matrix2[i][j]
             for j in range(len(matrix1))] for i in range(len(matrix1))]

def sub(matrix1, matrix2):
    return [[matrix1[i][j]-matrix2[i][j]
             for j in range(len(matrix1))] for i in range(len(matrix1))]


matrix1 = [[1, 2], [3, 4]]
matrix2 = [[5, 6], [7, 8]]

result_matrix = strassen_matrix_multiply(matrix1, matrix2)
print(result_matrix)
```

```
[[19, 22], [43, 50]]
```

Write Python program to sort n numbers using Merge sort algorithm.

In [3]:
```
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m
    L = [0] * (n1)
    R = [0] * (n2)
    for i in range(0, n1):
        L[i] = arr[l + i]
    for j in range(0, n2):
        R[j] = arr[m + 1 + j]
    i = 0
    j = 0
    k = l
    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1
def mergeSort(arr, l, r):
    if l < r:
        m = l+(r-l)//2
        mergeSort(arr, l, m)
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)
arr = [12, 11, 13, 5, 6, 7]
n = len(arr)
print("Given array is")
for i in range(n):
    print("%d" % arr[i],end=" ")

mergeSort(arr, 0, n-1)
print("\n\nSorted array is")
for i in range(n):
    print("%d" % arr[i],end=" ")
```

```
Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13
```

Write Python program to sort n names using Quick sort algorithm.

In [4]:
```
data = ["Rohan","Siddhesh","Sagar","Omkar"]
print("Unsorted Array")
print(data)
size = len(data)

def partition(array, low, high):
    pivot = array[high]
    i = low - 1
    for j in range(low, high):
        if array[j] <= pivot:
            i = i + 1
            (array[i], array[j]) = (array[j], array[i])
    (array[i + 1], array[high]) = (array[high], array[i + 1])
    return i + 1

def quickSort(array, low, high):
    if low < high:
        pi = partition(array, low, high)
        quickSort(array, low, pi - 1)
        quickSort(array, pi + 1, high)

quickSort(data, 0, size - 1)
print('Sorted Array in Ascending Order:')
print(data)
```

```
Unsorted Array
['Rohan', 'Siddhesh', 'Sagar', 'Omkar']
Sorted Array in Ascending Order:
['Omkar', 'Rohan', 'Sagar', 'Siddhesh']
```

Write Python program for inserting an element into binary tree.

In [5]:
```
class Node:
    def __init__(self,data):
        self.left=None
        self.right=None
        self.data=data

    def insert(self,data):
        if self.data is None:
            self.data=data
        else:
            if data < self.data:
                if self.left is None:
                    self.left=Node(data)
                else:
                    self.left.insert(data)
            elif data > self.data:
                if self.right is None:
                    self.right=Node(data)
                else:
                    self.right.insert(data)
def inOrderPrint(r):
    if r is None:
        return
    else:
        inOrderPrint(r.left)
        print(r.data,end=" ")
        inOrderPrint(r.right)

def preOrderPrint(r):
    if r is None:
        return
    else:
        print(r.data,end=" ")
        preOrderPrint(r.left)
        preOrderPrint(r.right)

if __name__=="__main__":
    root=Node("g")
    root.insert("c")
    root.insert("b")
    root.insert("a")
    root.insert("e")
    root.insert("d")
    root.insert("f")
    root.insert("i")
    root.insert("h")
    root.insert("j")
    root.insert("k")

preOrderPrint(root)
```

```
g c b a e d f i h j k
```

Write Python program for deleting an element(assuming data is given) from binary tree.

In [6]:
```
from queue import Queue
def deleteTree(root):
    if root:
        deleteTree(root.left)
        deleteTree(root.right)
        print("Deleting Node:", root.data)
        del root.data


deleteTree(root)
```

```
Deleting Node: a
Deleting Node: b
Deleting Node: d
Deleting Node: f
Deleting Node: e
Deleting Node: c
Deleting Node: h
Deleting Node: k
Deleting Node: j
Deleting Node: i
Deleting Node: g
```

Write Python program for finding the smallest and largest elements in an array A of size n using Selection algorithm

In [1]:
```
array=[12,22,66,12,53,2,32,3,2,2,23,6,5,3,6,65,56,36]
size =len(array)
def selection_sort(array):
    for i in range(size):
        min_index=i
        for j in range(i+1,size):
            if array[j]<array[min_index]:
                min_index=j
            array[i],array[min_index]=array[min_index],array[i]
    return array
print(selection_sort(array))
print(f"The smallest element in an array is {array[0]}.")
print(f"The largest element in an array is {array[-1]}.")
```

```
[2, 2, 2, 3, 3, 5, 6, 12, 12, 23, 22, 32, 36, 6, 53, 56, 65, 66]
The smallest element in an array is 2.
The largest element in an array is 66.
```

Write Python program for implementing Huffman Coding Algorithm.

In [4]:
```
import heapq

class node:

    def __init__(self, freq, symbol, left=None, right=None):
        self.freq = freq
        self.symbol = symbol
        self.left = left
        self.right = right
        self.huff = ''

    def __lt__(self, nxt):
        return self.freq < nxt.freq


def printNodes(node, val=''):
    newVal = val + str(node.huff)
    if(node.left):
        printNodes(node.left, newVal)
    if(node.right):
        printNodes(node.right, newVal)
    if(not node.left and not node.right):
        print(f"{node.symbol} -> {newVal}")

chars = ['a', 'b', 'c', 'd', 'e']
freq = [3,5,6,4,2]
nodes = []

for x in range(len(chars)):
    heapq.heappush(nodes, node(freq[x], chars[x]))

while len(nodes) > 1:
    left = heapq.heappop(nodes)
    right = heapq.heappop(nodes)
    left.huff = 0
    right.huff = 1
    newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)
    heapq.heappush(nodes, newNode)

printNodes(nodes[0])
```

```
d -> 00
b -> 01
e -> 100
a -> 101
c -> 11
```

In [ ]: