DESIGN AND ANALYSIS OF ALGORITHM PRACTICAL MANUAL
FYCS SEM II

www.profajaypashankar.com

PROF.AJAY PASHANKAR

www.profajaypashankar.com

1)Programs on 1-d arrays like - sum of elements of array, searching an element inarray, finding minimum and maximum element in array, count the number ofeven and odd numbers in array. For all such programs, also find the timeComplexity, compare if there are multiple methods

Program 1 A: sum of elements of array

```
# PROGRAM 1 A
# sum of elements in given array
def _sum(arr):
    sum=0
    for i in arr:
        sum = sum + i
    return(sum)
arr=[]
arr = [12, 3, 4, 15,45,78,450]
n = len(arr)
ans = _sum(arr)
print ('Sum of the array is ', ans)
```

OUTPUT:

```
====================== RESTART: C:/Python34/program1a.py ===========
==========
Sum of the array is  607
>>>
```

PROGRAM 1B : finding an element in an array

```
#program 1B finding an element in an array
#finding character value
x = ['p','y','t','h','o','n']
print(x.index('o'))

# finding numerical value
x = [5,1,7,0,3,4]
print(x.index(7))
```

```
>>>
====================== RESTART: C:/Python34/progam1b.py ==========
============
4
2
>>>
```

PROGRAM 1C: program to find minimum (or maximum) element in an array

```
#program 1C:
#  program to find minimum (or maximum) element in an array
# Minimum Function
def getMin(arr, n):
    res = arr[0]
    for i in range(1,n):
        res = min(res, arr[i])
    return res

# Maximum Function
def getMax(arr, n):
    res = arr[0]
    for i in range(1,n):
        res = max(res, arr[i])
    return res

# Driver Program
arr = [12, 1234, 45, 67, 1]
n = len(arr)
print ("Minimum element of array:", getMin(arr))
print ("Maximum element of array:", getMax(arr, n))
```

```
>>>
========== RESTART: C:/Python34/program1c.py ==========
==========
Minimum element of array: 1
Maximum element of array: 1234
>>>
```

PROGRAM 1D : program to count number of even and odd elements in an array

#program 1 D

```
#  program to count number of  even and odd elements in an array

def CountingEvenOdd(arr, arr_size):
    even_count = 0
    odd_count = 0

    # loop to read all the values
    # in the array
    for i in range(arr_size):

        # checking if a number is
        # completely divisible by 2
        if (arr[i] & 1 == 1):
            odd_count += 1   #odd_count=odd_count+1
        else:
            even_count += 1  #even_count=even_count+1

    print("Number of even elements = ",
          even_count)
    print("Number of odd elements = ",
          odd_count)


arr = [2, 3, 4, 5, 6,7,8,9,10,11,12,13]
n = len(arr)

# Function Call
CountingEvenOdd(arr, n)
```

```
>>>
========== RESTART: C:/Python34/program1d.py ==========
==========
Number of even elements =  6
Number of odd elements =  6
>>>
```

PROGRAM 2A: row-sum , COLUMN-wise of 2D ARRAY

```
# PROGRAM 2 A row-sum & COLUMN-wise Sum  of 2D ARRAY
n = int(input("Enter the number of rows:"))
m = int(input("Enter the number of columns:"))
matrix = []
print("Enter values in matrix :")
# For user input
for i in range(n):
    data =[]
    for j in range(m):
        data.append(int(input()))
    matrix.append(data)
# For printing the matrix
for i in range(n):
    for j in range(m):
        print(matrix[i][j], end = " ")
    print()

# For printing row wise sum
for i in range(n):
```

```
    sum = 0
    for j in range(m):
        sum = sum + matrix[i][j]
    print('Sum of row',i+1,':',sum)

# For printing column wise sum

for i in range(m):
    sum = 0
    for j in range(n):
        sum = sum + matrix[j][i]
    print('Sum of column',i,':',sum)
```

```
>>>
================== RESTART: C:/Users/admin/Documents/prog2a.py ==================
Enter the number of rows:3
Enter the number of columns:3
Enter values in matrix :
14
45
89
95
12
35
12
23
15
14 45 89
95 12 35
12 23 15
Sum of row 1 : 148
Sum of row 2 : 142
Sum of row 3 : 50
Sum of column 0 : 121
Sum of column 1 : 80
Sum of column 2 : 139
```

#program 2 B find sum of diagonals

```
# A simple Python program to # find sum of diagonals
MAX = 100
def printDiagonalSums(mat, n):

    principal = 0
    secondary = 0;
    for i in range(0, n):
        for j in range(0, n):

            # Condition for principal diagonal
            if (i == j):
                principal += mat[i][j]

            # Condition for secondary diagonal
            if ((i + j) == (n - 1)):
                secondary += mat[i][j]

    print("Principal Diagonal:", principal)
    print("Secondary Diagonal:", secondary)

# Driver code
a = [[ 1, 2, 3, 4 ],
     [ 5, 6, 7, 8 ],
     [ 1, 2, 3, 4 ],
      [ 5, 6, 7, 8 ]]
printDiagonalSums(a, 4)
```

```
>>>
================ RESTART: C:/Users/admin/Documents/program2bb.py ===============
Principal Diagonal: 18
Secondary Diagonal: 18
>>>
```

#PROGRAM 2C two matrices using nested loop

 # Program to add two matrices using nested loop

```
X = [[1,2,3],
   [4 ,5,6],
   [7 ,8,9]]

Y = [[9,8,7],
   [6,5,4],
   [3,2,1]]
result = [[0,0,0],
       [0,0,0],
       [0,0,0]]

# iterate through rows
for i in range(len(X)):
# iterate through columns
   for j in range(len(X[0])):
       result[i][j] = X[i][j] + Y[i][j]

for r in result:
   print(r)
```

```
>>>
================= RESTART: C:/Users/admin/Documents/program2c.py =================
[10, 10, 10]
[10, 10, 10]
[10, 10, 10]
>>>
```

#program 2d multiply two matrices using nested loops

Program to multiply two matrices using nested loops

```
# take a 3x3 matrix
A = [[12, 7, 3],
   [4, 5, 6],
   [7, 8, 9]]

# take a 3x4 matrix
B = [[5, 8, 1, 2],
   [6, 7, 3, 0],
   [4, 5, 9, 1]]

result = [[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]]

# iterating by row of A
for i in range(len(A)):

   # iterating by column by B
   for j in range(len(B[0])):

       # iterating by rows of B
       for k in range(len(B)):
           result[i][j] += A[i][k] * B[k][j]

for r in result:
   print(r)
```

```
>>>
================ RESTART: C:/Users/admin/Documents/program2d.py ================
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 23]
>>>
```

#PROGRAM 3 : Program to create a list-based stack and perform various stack operations.

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class Stack:
    def __init__(self):
        self.head = None

    def push(self, data):
        if self.head is None:
            self.head = Node(data)
        else:
            new_node = Node(data)
            new_node.next = self.head
            self.head = new_node

    def pop(self):
        if self.head is None:
            return None
        else:
            popped = self.head.data
            self.head = self.head.next
            return popped

a_stack = Stack()
while True:
    print('push <value>')
    print('pop')
    print('quit')
    do = input('What would you like to do? ').split()

    operation = do[0].strip().lower()
    if operation == 'push':
        a_stack.push(int(do[1]))
    elif operation == 'pop':
        popped = a_stack.pop()
        if popped is None:
            print('Stack is empty.')
        else:
            print('Popped value: ', int(popped))
    elif operation == 'quit':
        break
```

```
>>>
================ RESTART: C:/Users/admin/Documents/program333.py ===============
push <value>
pop
quit
What would you like to do? push 15
push <value>
pop
quit
What would you like to do? push 15
push <value>
pop
quit
What would you like to do? push 112
push <value>
pop
quit
What would you like to do? pop 112
Popped value:  112
push <value>
pop
quit
What would you like to do?
```


#PROGRAM 4 A LINEAR SEARCH

```
def LinearSearch(array, n, k):

    for j in range(0, n):

        if (array[j] == k):

            return j

    return -1

array = [1, 3, 5, 7, 9]

k = 7
n = len(array)

result = LinearSearch(array, n, k)

if(result == -1):

    print("Element not found")

else:

    print("Element found at index: ", result)
```

```
>>>
================ RESTART: C:\Users\admin\Documents\program4a.py ================
Element found at index:  3
>>>
```

#PROGRAM 4B BINARY SEARCH

```
def BinarySearch(arr, k, low, high):
    if high >= low:
        mid = low + (high - low)//2
        if arr[mid] == k:
            return mid
        elif arr[mid] > k:
            return BinarySearch(arr, k, low, mid-1)
        else:
            return BinarySearch(arr, k, mid + 1, high)
```

```
    else:
        return -1
arr = [1, 3, 5, 7, 9,15,16,14,45]
k = 15
result = BinarySearch(arr, k, 0, len(arr)-1)
if result != -1:
    print("Element is present at index " + str(result))
else:
    print("Not found")
```

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Users/admin/Documents/program4b.py ================
Element is present at index 5
>>>
```

#PROGRAM 5 A BUBBLE SORT

```
# Creating a bubble sort function
def bubble_sort(list1):
    # Outer loop for traverse the entire list
    for i in range(0,len(list1)-1):
        for j in range(len(list1)-1):
            if(list1[j]>list1[j+1]):
                temp = list1[j]
                list1[j] = list1[j+1]
                list1[j+1] = temp
    return list1

list1 = [5, 3, 8, 6, 7, 2]
print("The unsorted list is: ", list1)
# Calling the bubble sort function
print("The sorted list is: ", bubble_sort(list1))
```

```
>>>
================ RESTART: C:/Users/admin/Documents/program5a.py ================
The unsorted list is:  [5, 3, 8, 6, 7, 2]
The sorted list is:  [2, 3, 5, 6, 7, 8]
>>>
```

(k+1 element with number of iteration)

```
def bubble_sort(list1):
    has_swapped = True

    total_iteration = 0
    while(has_swapped):
        has_swapped = False
        for i in range(len(list1) - total_iteration - 1):
            if list1[i] > list1[i+1]:
                # Swap
                list1[i], list1[i+1] = list1[i+1], list1[i]
                has_swapped = True
        total_iteration += 1
    print("The number of iteraton: ",total_iteration)
    return list1

list1 = [5, 3, 8, 6, 7, 2]
print("The unsorted list is: ", list1)
# Calling the bubble sort funtion
print("The sorted list is: ", bubble_sort(list1))
```

```
>>>
================ RESTART: C:/Users/admin/Documents/program5aa.py ===============
The unsorted list is:  [5, 3, 8, 6, 7, 2]
The number of iteraton:  6
The sorted list is:  [2, 3, 5, 6, 7, 8]
>>>
```

#PROGRAM 5 B: SLECTION SORT

```
def selectionSort( itemsList ):
    n = len( itemsList )
    for i in range( n - 1 ):
        minValueIndex = i
        for j in range( i + 1, n ):
            if itemsList[j] < itemsList[minValueIndex] :
                minValueIndex = j

        if minValueIndex != i :
            temp = itemsList[i]
            itemsList[i] = itemsList[minValueIndex]
            itemsList[minValueIndex] = temp
    return itemsList

el = [21,6,9,33,3]
print(selectionSort(el))
```

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Users/admin/Documents/program5b.py ================
[3, 6, 9, 21, 33]
>>>
```

PROGRAM 5C: INSERTION SORT

```
#PROGRAM 5C INSERTION SORT
# creating a function for insertion
def insertion_sort(list1):

        # Outer loop to traverse through 1 to len(list1)
        for i in range(1, len(list1)):

            value = list1[i]

            # Move elements of list1[0..i-1], that are
            # greater than value, to one position ahead
            # of their current position
            j = i - 1
            while j >= 0 and value < list1[j]:
                list1[j + 1] = list1[j]
                j -= 1
            list1[j + 1] = value
        return list1
            # Driver code to test above

list1 = [10, 5, 13, 8, 2]
print("The unsorted list is:", list1)
print("The sorted list1 is:", insertion_sort(list1))
```

```
>>>
================ RESTART: C:/Users/admin/Documents/program 5c.py ===============
The unsorted list is: [10, 5, 13, 8, 2]
The sorted list1 is: [2, 5, 8, 10, 13]
>>>
```

6) Programs to select the Nth Max/Min element in a list by using various algorithms. Compare the efficiency of algorithms.

Program 6A: to select Nth Max/Min element In a list

```
#Program 6A  To select Nth Max/Min Element in a list
# Python program of above implementation

# structure is used to return two values from minMax()

class pair:
	def __init__(self):
		self.min = 0
		self.max = 0

def getMinMax(arr: list, n: int) -> pair:
	minmax = pair()

	# If there is only one element then return it as min and max both
	if n == 1:
		minmax.max = arr[0]
		minmax.min = arr[0]
		return minmax

	# If there are more than one elements, then initialize min
	# and max
	if arr[0] > arr[1]:
		minmax.max = arr[0]
		minmax.min = arr[1]
	else:
		minmax.max = arr[1]
		minmax.min = arr[0]
	for i in range(2, n):
		if arr[i] > minmax.max:
			minmax.max = arr[i]
		elif arr[i] < minmax.min:
			minmax.min = arr[i]

	return minmax

# Driver Code
if __name__ == "__main__":
	arr = [1000, 11, 445, 1, 330, 3000]
	arr_size = 6
	minmax = getMinMax(arr, arr_size)
	print("Minimum element is", minmax.min)
	print("Maximum element is", minmax.max)
```

Output:-

```
>>> ========================
>>>
Minimum element is 1
Maximum element is 3000
>>>
```

7) Programs to find a pattern in a given string - general way and brute force technique. Compare the efficiency of algorithms.

Program7A: Program to find a pattern by general way

```
#Program 7A program to find a pattern by general way
# Python3 program for Naive Pattern
# Searching algorithm
def search(pat, txt):
	M = len(pat)
	N = len(txt)
```

```
        # A loop to slide pat[] one by one */
        for i in range(N - M + 1):
                j = 0

                # For current index i, check
                # for pattern match */
                while(j < M):
                        if (txt[i + j] != pat[j]):
                                break
                        j += 1

                if (j == M):
                        print("Pattern found at index ", i)

# Driver Code
if __name__ == '__main__':
        txt = "AABAACAADAABAAABAA"
        pat = "AABA"
        search(pat, txt)
```

Output:-

```
>>>
Pattern found at index  0
Pattern found at index  9
Pattern found at index  13
>>>
```

Program 7B: Program to find a pattern by Brute Force Technique
Definition- Brute force algorithm is a technique that guarantees solutions for problems of any domain helps in solving the simpler problems and also provides a solution that can serve as a benchmark for evaluating other design techniques.

```
#Program 7B program to find a pattern by brute force technique

def linear_search(lst, match):
                        for idx in range(len(lst)):
                          if lst[idx] == match:
                            return idx
                          else:
                            raise ValueError("{0} not in list".format(match))

                        recipe = ["nori", "tuna", "soy sauce", "sushi rice"]
                        ingredient = "avocado"
                        try:
                          print(linear_search(recipe, ingredient))
                        except ValueError as msg:
                          print("{0}".format(msg))
```

Output-

```
>>> ------------------
>>>
avocado not in list
>>>
```

```
def find_maximum(lst):
  max = None
  for el in lst:
    if max == None or el > max:
      max = el
  return max
test_scores = [88, 93, 75, 100, 80, 67, 71, 92, 90, 83]
print(find_maximum(test_scores)) # returns 100
```

Output-

```
>>>
100
>>>
```

```
def linear_search(lst, match):
  matches = []
  for idx in range(len(lst)):
    if lst[idx] == match:
      matches.append(idx)
  if matches:
    return matches
  else:
    raise ValueError("{0} not in list".format(match))

scores = [55, 65, 32, 40, 55]
print(linear_search(scores, 55))
```

Output-

```
>>>
[0, 4]
>>>
```

---

8) Programs on recursion like factorial, fibonacci, tower of hanoi. Compare algorithms to find factorial/fibonacci using iterative and recursive approaches.

Program 8A: Program on recursion- factorial recursion

```
# Python 3 program to find
# factorial of given number
def factorial(n):

        # Checking the number
        # is 1 or 0 then
        # return 1
        # other wise return
        # factorial
        if (n==1 or n==0):

                return 1

        else:

                return (n * factorial(n - 1))

# Driver Code
num = 5;
print("number : ",num)
print("Factorial : ",factorial(num))
```

Output-

```
>>>
number :  5
Factorial :  120
>>>
```

Program 8B: Program on recursion-fibonacci recursion

```
# Python Program to Display Fibonacci Sequence Using Recursion
def recur_fibo(n):
   if n <= 1:
      return n
   else:
      return(recur_fibo(n-1) + recur_fibo(n-2))

nterms = 10

# check if the number of terms is valid
if nterms <= 0:
   print("Plese enter a positive integer")
else:
   print("Fibonacci sequence:")
   for i in range(nterms):
      print(recur_fibo(i))
```

Output-

```
>>>
Fibonacci sequence:
0
1
1
2
3
5
8
13
21
34
>>>
```

program 8C: Program on recursion-Tower of Hanoi

```
#Program 8C program on recursion- Tower of Hanoi
# Recursive Python function to solve the tower of hanoi

def TowerOfHanoi(n , source, destination, auxiliary):
        if n==1:
                print ("Move disk 1 from source",source,"to destination",destination)
                return
        TowerOfHanoi(n-1, source, auxiliary, destination)
        print ("Move disk",n,"from source",source,"to destination",destination)
        TowerOfHanoi(n-1, auxiliary, destination, source)

# Driver code
n = 4
TowerOfHanoi(n,'A','B','C')
# A, C, B are the name of rods

# Contributed By Dilip Jain
```

Output-

```
>>>
Move disk 1 from source A to destination C
Move disk 2 from source A to destination B
Move disk 1 from source C to destination B
Move disk 3 from source A to destination C
Move disk 1 from source B to destination A
Move disk 2 from source B to destination C
Move disk 1 from source A to destination C
Move disk 4 from source A to destination B
Move disk 1 from source C to destination B
Move disk 2 from source C to destination A
Move disk 1 from source B to destination A
Move disk 3 from source C to destination B
Move disk 1 from source A to destination C
Move disk 2 from source A to destination B
Move disk 1 from source C to destination B
>>>
```

9) Program to implement file merging, coin change problems using Greedy Algorithm and to understand time complexity

Program 9A: Program to implement file merging

```
# Python Program to implement
# Optimal File Merge Pattern

class Heap():
        # Building own implementation of Min Heap
        def __init__(self):

                self.h = []

        def parent(self, index):
                # Returns parent index for given index

                if index > 0:
                        return (index - 1) // 2

        def lchild(self, index):
                # Returns left child index for given index

                return (2 * index) + 1

        def rchild(self, index):
                # Returns right child index for given index

                return (2 * index) + 2

        def addItem(self, item):

                # Function to add an item to heap
                self.h.append(item)

                if len(self.h) == 1:

                        # If heap has only one item no need to heapify
                        return

                index = len(self.h) - 1
                parent = self.parent(index)

                # Moves the item up if it is smaller than the parent
                while index > 0 and item < self.h[parent]:
                        self.h[index], self.h[parent] = self.h[parent], self.h[parent]
                        index = parent
                        parent = self.parent(index)

        def deleteItem(self):

                # Function to add an item to heap
                length = len(self.h)
                self.h[0], self.h[length-1] = self.h[length-1], self.h[0]
                deleted = self.h.pop()

                # Since root will be violating heap property
                # Call moveDownHeapify() to restore heap property
                self.moveDownHeapify(0)

                return deleted

        def moveDownHeapify(self, index):

                # Function to make the items follow Heap property
```

```
                # Compares the value with the children and moves item down

                lc, rc = self.lchild(index), self.rchild(index)
                length, smallest = len(self.h), index

                if lc < length and self.h[lc] <= self.h[smallest]:
                        smallest = lc

                if rc < length and self.h[rc] <= self.h[smallest]:
                        smallest = rc

                if smallest != index:
                        # Swaps the parent node with the smaller child
                        self.h[smallest], self.h[index] = self.h[index], self.h[smallest]

                        # Recursive call to compare next subtree
                        self.moveDownHeapify(smallest)

        def increaseItem(self, index, value):
                # Increase the value of 'index' to 'value'

                if value <= self.h[index]:
                        return

                self.h[index] = value
                self.moveDownHeapify(index)


class OptimalMergePattern():
        def __init__(self, n, items):

                self.n = n
                self.items = items
                self.heap = Heap()

        def optimalMerge(self):

                # Corner cases if list has no more than 1 item
                if self.n <= 0:
                        return 0

                if self.n == 1:
                        return self.items[0]

                # Insert items into min heap
                for _ in self.items:
                        self.heap.addItem(_)

                count = 0
                while len(self.heap.h) != 1:
                        tmp = self.heap.deleteItem()
                        count += (tmp + self.heap.h[0])
                        self.heap.increaseItem(0, tmp + self.heap.h[0])

                return count


# Driver Code
if __name__ == '__main__':
        OMP = OptimalMergePattern(6, [2, 3, 4, 5, 6, 7])
        ans = OMP.optimalMerge()
        print(ans)
```

Output-

```
>>> ================================ RESTART ================================
>>>
68
>>>
```

Program 9B: Program to implement coin change problems using greedy algorithm

```
# Python3 program to find minimum
# number of denominations

def findMin(V):

	# All denominations of Indian Currency
	deno = [1, 2, 5, 10, 20, 50,
			100, 500, 1000]
	n = len(deno)

	# Initialize Result
	ans = []

	# Traverse through all denomination
	i = n - 1
	while(i >= 0):

		# Find denominations
		while (V >= deno[i]):
			V -= deno[i]
			ans.append(deno[i])

		i -= 1

	# Print result
	for i in range(len(ans)):
		print(ans[i], end = " ")

# Driver Code
if __name__ == '__main__':
	n = 93
	print("Following is minimal number",
		"of change for", n, ": ", end = "")
	findMin(n)

#
# Surendra_Gangwar
```

Output-

```
>>> ================================ RESTART ================================
>>>
Following is minimal number of change for 93 : 50 20 20 2 1
>>>
```

10) Program to implement merge sort, Straseen"s Matrix Multiplication using D-n-C Algorithm and to understand time complexity.

Program10A: Program to implement merge sort in python

```
# Python program for implementation of MergeSort

# Merges two subarrays of arr[].
# First subarray is arr[l..m]
# Second subarray is arr[m+1..r]


def merge(arr, l, m, r):
	n1 = m - l + 1
	n2 = r - m

	# create temp arrays
	L = [0] * (n1)
	R = [0] * (n2)

	# Copy data to temp arrays L[] and R[]
	for i in range(0, n1):
		L[i] = arr[l + i]

	for j in range(0, n2):
		R[j] = arr[m + 1 + j]

	# Merge the temp arrays back into arr[l..r]
	i = 0	# Initial index of first subarray
	j = 0	# Initial index of second subarray
	k = l	# Initial index of merged subarray

	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

	# Copy the remaining elements of L[], if there
	# are any
	while i < n1:
		arr[k] = L[i]
		i += 1
		k += 1

	# Copy the remaining elements of R[], if there
	# are any
	while j < n2:
		arr[k] = R[j]
		j += 1
		k += 1

# l is for left index and r is right index of the
# sub-array of arr to be sorted


def mergeSort(arr, l, r):
	if l < r:

		# Same as (l+r)//2, but avoids overflow for
		# large l and h
		m = l+(r-l)//2
```

```
        # Sort first and second halves
        mergeSort(arr, l, m)
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)


# Driver code to test above
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=" ")

#  Mohit Kumra
```

Output-

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

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

Program 10B:Program to implement Strassen''s matrix multiplication using D-n-C algorithm

```
import numpy as np

def strassen_algorithm(x, y):
    if x.size == 1 or y.size == 1:
        return x * y

    n = x.shape[0]

    if n % 2 == 1:
        x = np.pad(x, (0, 1), mode='constant')
        y = np.pad(y, (0, 1), mode='constant')

    m = int(np.ceil(n / 2))
    a = x[: m, : m]
    b = x[: m, m:]
    c = x[m:, : m]
    d = x[m:, m:]
    e = y[: m, : m]
    f = y[: m, m:]
    g = y[m:, : m]
    h = y[m:, m:]
    p1 = strassen_algorithm(a, f - h)
    p2 = strassen_algorithm(a + b, h)
    p3 = strassen_algorithm(c + d, e)
    p4 = strassen_algorithm(d, g - e)
    p5 = strassen_algorithm(a + d, e + h)
    p6 = strassen_algorithm(b - d, g + h)
    p7 = strassen_algorithm(a - c, e + f)
    result = np.zeros((2 * m, 2 * m), dtype=np.int32)
    result[: m, : m] = p5 + p4 - p2 + p6
```

```
    result[: m, m:] = p1 + p2
    result[m:, : m] = p3 + p4
    result[m:, m:] = p1 + p5 - p3 - p7

    return result[: n, : n]

if __name__ == "__main__":

    x = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
    y = np.array([[-1, 0, 0], [0, -1, 0], [0, 0, -1]])
    print('Matrix multiplication result: ')
    print(strassen_algorithm(x, y))
```

Output-

```
Matrix multiplication result:
[[-1  0  0]
 [ 0 -1  0]
 [ 0  0 -1]]
>
```

The above Output is from Online Python Compiler(Programiz)

Note:-

Don't try this program in python 3.4.3 As Numpy is not defined in that version.

So use latest version i.e python 3.10.2, otherwise use online python compiler.

---

11) Program to implement fibonacci series, Longest Common Subsequence using dynamic programming and to understand time complexity. Compare it with the general recursive algorithm.

Program11A: Program to implement Fibonacci series in python

```
# Program to display the Fibonacci sequence up to n-th term

nterms = int(input("How many terms? "))

# first two terms
n1, n2 = 0, 1
count = 0

# check if the number of terms is valid
if nterms <= 0:
   print("Please enter a positive integer")
# if there is only one term, return n1
elif nterms == 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
# generate fibonacci sequence
else:
   print("Fibonacci sequence:")
   while count < nterms:
       print(n1)
       nth = n1 + n2
       # update values
       n1 = n2
       n2 = nth
       count += 1
```

Output-

```
>>>
How many terms? 10
Fibonacci sequence:
0
1
1
2
3
5
8
13
21
34
```

Program11B: Program to implement longest common subsequence using dynamic programming

```
# Dynamic Programming implementation of LCS problem

def lcs(X, Y):
        # find the length of the strings
        m = len(X)
        n = len(Y)

        # declaring the array for storing the dp values
        L = [[None]*(n + 1) for i in range(m + 1)]

        """Following steps build L[m + 1][n + 1] in bottom up fashion
        Note: L[i][j] contains length of LCS of X[0..i-1]
        and Y[0..j-1]"""
        for i in range(m + 1):
                for j in range(n + 1):
                        if i == 0 or j == 0 :
                                L[i][j] = 0
                        elif X[i-1] == Y[j-1]:
                                L[i][j] = L[i-1][j-1]+1
                        else:
                                L[i][j] = max(L[i-1][j], L[i][j-1])

        # L[m][n] contains the length of LCS of X[0..n-1] & Y[0..m-1]
        return L[m][n]
# end of function lcs

# Driver program to test the above function
X = "AGGTAB"
Y = "GXTXAYB"
print("Length of LCS is ", lcs(X, Y))
```

Output-

```
>>> =================
>>>
Length of LCS is  4
>>>
```

12) Program to implement N-Queen Problem, Binary String generation using

Backtracking Strategy and to understand time complexity.

Program 12A: Program to implement N-Queen Problem in Python

```
# Python program to solve N Queen
# Problem using backtracking

global N
N = 4

def printSolution(board):
	for i in range(N):
		for j in range(N):
			print (board[i][j],end=' ')
		print()


# A utility function to check if a queen can
# be placed on board[row][col]. Note that this
# function is called when "col" queens are
# already placed in columns from 0 to col -1.
# So we need to check only left side for
# attacking queens
def isSafe(board, row, col):

	# Check this row on left side
	for i in range(col):
		if board[row][i] == 1:
			return False

	# Check upper diagonal on left side
	for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
		if board[i][j] == 1:
			return False

	# Check lower diagonal on left side
	for i, j in zip(range(row, N, 1), range(col, -1, -1)):
		if board[i][j] == 1:
			return False

	return True

def solveNQUtil(board, col):
	# base case: If all queens are placed
	# then return true
	if col >= N:
		return True

	# Consider this column and try placing
	# this queen in all rows one by one
	for i in range(N):

		if isSafe(board, i, col):
			# Place this queen in board[i][col]
			board[i][col] = 1

			# recur to place rest of the queens
			if solveNQUtil(board, col + 1) == True:
				return True

			# If placing queen in board[i][col
			# doesn't lead to a solution, then
			# queen from board[i][col]
			board[i][col] = 0
```

```
        # if the queen can not be placed in any row in
        # this column col then return false
        return False

# This function solves the N Queen problem using
# Backtracking. It mainly uses solveNQUtil() to
# solve the problem. It returns false if queens
# cannot be placed, otherwise return true and
# placement of queens in the form of 1s.
# note that there may be more than one
# solutions, this function prints one of the
# feasible solutions.
def solveNQ():
        board = [ [0, 0, 0, 0],
                  [0, 0, 0, 0],
                  [0, 0, 0, 0],
                  [0, 0, 0, 0]
                  ]

        if solveNQUtil(board, 0) == False:
                print ("Solution does not exist")
                return False

        printSolution(board)
        return True

# driver program to test above function
solveNQ()

#  Divyanshu Mehta
```

Output-

```
>>>
0  0  1  0
1  0  0  0
0  0  0  1
0  1  0  0
>>>
```

Program 12B: Program to implement Binary String generation using
Backtracking Strategy

```
# Python3 implementation of the
# above approach

# Function to print the output
def printTheArray(arr, n):

        for i in range(0, n):
                print(arr[i], end = " ")

        print()

# Function to generate all binary strings
def generateAllBinaryStrings(n, arr, i):

        if i == n:
                printTheArray(arr, n)
                return

        # First assign "0" at ith position
        # and try for all other permutations
        # for remaining positions
        arr[i] = 0
```

```
        generateAllBinaryStrings(n, arr, i + 1)

        # And then assign "1" at ith position
        # and try for all other permutations
        # for remaining positions
        arr[i] = 1
        generateAllBinaryStrings(n, arr, i + 1)

# Driver Code
if __name__ == "__main__":

        n = 4
        arr = [None] * n

        # Print all binary strings
        generateAllBinaryStrings(n, arr, 0)

# This code is contributed
# by Rituraj Jain
```

Output-

```
>>> ================================ RESTART ================================
>>>
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1
>>>
```