DS PRACTICALS

## NAME: - MITALI VISHWEKAR

## ROLL NO: - 4020

## CLASS: - SYCS

## Practical 1a.

'''

Write a program to store the elements in 1-D array and provide an option to perform

the operatons like searching, sorting, merge\_eling, reversing the elements.

'''

class pract1D:

def \_init\_(self, a):

self.array = a

def search(self, e):

if e in self.array:

return True

return False

def sort(self):

for i in range(len(self.array)):

lowest\_index = i

for j in range(i+1, len(self.array)):

if self.array[j] < self.array[lowest\_index]:

lowest\_index = j

self.array[i], self.array[lowest\_index] = self.array[lowest\_index], self.array[i]

return self.array

def merge\_el(self,l):

self.array = self.array + l

return self.array

def reverse(self):

return self.array[::-1]

a = [5,6,7,89,2,5,6,1]

o = pract1D(a)

print(o.sort())

print(o.search(89))

print(o.merge\_el([8,5,7,9,3]))

print(o.reverse())

## Practical 1b

class Node:

def \_init\_ (self, element, next = None ):

self.element = element

self.next = next

def display(self):

print(self.element)

class LinkedList:

def \_init\_(self):

self.head = None

self.size = 0

def \_len\_(self):

return self.size

def is\_empty(self):

return self.size == 0

def display(self):

if self.size == 0:

print("No element")

return

first = self.head

print(first.element)

first = first.next

while first:

print(first.element)

first = first.next

def add\_head(self,e):

temp = self.head

self.head = Node(e)

self.head.next = temp

self.size += 1

def get\_tail(self):

last\_object = self.head

while (last\_object.next != None):

last\_object = last\_object.next

return last\_object

def remove\_head(self):

if self.is\_empty():

print("Empty Singly linked list")

else:

print("Removing")

self.head = self.head.next

self.size -= 1

def add\_tail(self,e):

new\_value = Node(e)

self.get\_tail().next = new\_value

self.size += 1

def find\_second\_last\_element(self):

#second\_last\_element = None

if self.size >= 2:

first = self.head

temp\_counter = self.size -2

while temp\_counter > 0:

first = first.next

temp\_counter -= 1

return first

else:

print("Size not sufficient")

return None

def remove\_tail(self):

if self.is\_empty():

print("Empty Singly linked list")

elif self.size == 1:

self.head == None

self.size -= 1

else:

Node = self.find\_second\_last\_element()

if Node:

Node.next = None

self.size -= 1

def get\_node\_at(self,index):

element\_node = self.head

counter = 0

if index > self.size-1:

print("Index out of bound")

return None

while(counter < index):

element\_node = element\_node.next

counter += 1

return element\_node

def remove\_between\_list(self,position):

if position > self.size-1:

print("Index out of bound")

elif position == self.size-1:

self.remove\_tail()

elif position == 0:

self.remove\_head()

else:

prev\_node = self.get\_node\_at(position-1)

next\_node = self.get\_node\_at(position+1)

prev\_node.next = next\_node

self.size -= 1

def add\_between\_list(self,position,element):

if position > self.size:

print("Index out of bound")

elif position == self.size:

self.add\_tail(element)

elif position == 0:

self.add\_head(element)

else:

prev\_node = self.get\_node\_at(position-1)

current\_node = self.get\_node\_at(position)

prev\_node.next = element

element.next = current\_node

self.size -= 1

def search (self,search\_value):

index = 0

while (index < self.size):

value = self.get\_node\_at(index)

print("Searching at " + str(index) + " and value is " + str(value.element))

if value.element == search\_value:

print("Found value at " + str(index) + " location")

return True

index += 1

print("Not Found")

return False

def merge(self,linkedlist\_value):

if self.size > 0:

last\_node = self.get\_node\_at(self.size-1)

last\_node.next = linkedlist\_value.head

self.size = self.size + linkedlist\_value.size

else:

self.head = linkedlist\_value.head

self.size = linkedlist\_value.size

# 1,2, (size , 2 , SLE = 0)

# 1,2,3,5 (size , 3 , SLE = 1)

'''

l1 = 1,2,3,5

l2 = 3,5,6,7

l1.copy(l2)

l1 = 3,5,6,7

l1.head = l2.head

l1.size = l2.size

l1 = None

l2 = 1,2,3

l1.merge(l2) ? => l1 = 3,1,2,14,5,4,2,9,1,2,14,5,4,2,9,1,2,14,5,4,2,9,1,2,14,5,4,2,9

'''

## Practical 2

#Python program to perform matrix operations, matrix addition,matrix subtraction,matrix multiplication - addition

mat1 = [[1, 2], [3, 4]]

mat2 = [[1, 2], [3, 4]]

mat3 = [[0, 0], [0, 0]]

for i in range(0, 2):

for j in range(0, 2):

mat3[i][j] = mat1[i][j] + mat2[i][j]

print("Addition of two matrices")

for i in range(0, 2):

for j in range(0, 2):

print(mat3[i][j], end = "")

print()

#Python program to perform matrix operations, matrix addition,matrix subtraction,matrix multiplication - subtraction

mat1 = [[9, 2], [5, 3]]

mat2 = [[8, 1], [4, 2]]

mat3 = [[0, 0], [0, 0]]

for i in range(0, 2):

for j in range(0, 2):

mat3[i][j] = mat1[i][j] - mat2[i][j]

print("Subtraction of two matrices")

for i in range(0, 2):

for j in range(0, 2):

print(mat3[i][j], end = "")

print()

#Python program to multiply two matrices

mat1 = [[9, 2], [5, 3]]

mat2 = [[8, 1], [4, 2]]

mat3 = [[0, 0], [0, 0]]

for i in range(0, 2):

for j in range(0, 2):

mat3[i][j] = mat1[i][j] \* mat2[i][j]

print("Multiplication of two matrices")

for i in range(0, 2):

for j in range(0, 2):

print(mat3[i][j], end = "")

print()