LAB_3.py ● LAB_5.py
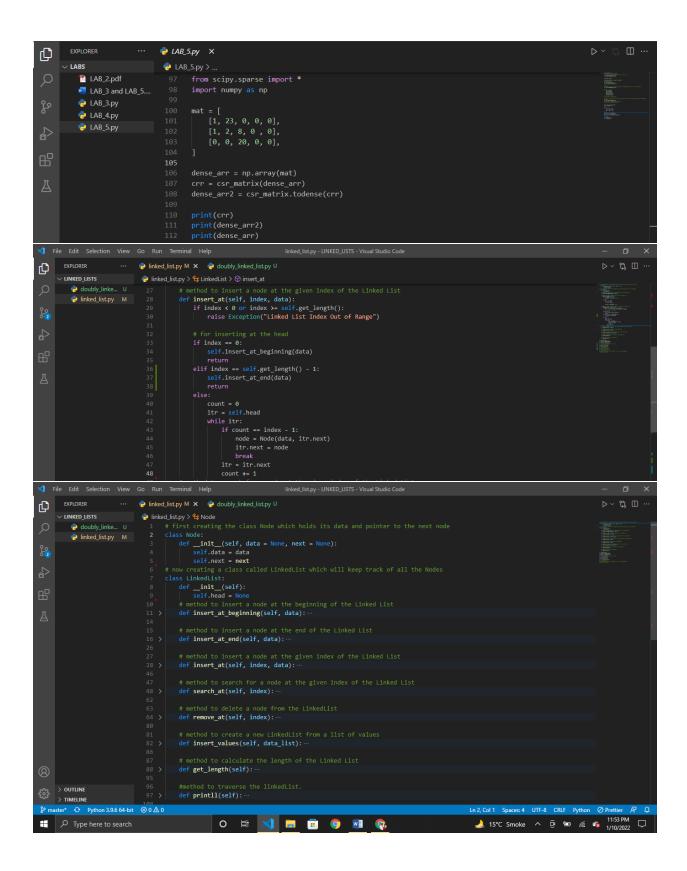
LAB_3.py > ...

```python
# part (a) -->
from timeit import *
x ='''
def binary_search(array, item):
        beg, end = 0, len(array)-1
        mid = int((beg+end)/2)

        while beg <= end and array[mid] != item:
            if item < array[mid]:
                end = mid - 1
                mid = int((beg+end)/2)
            else:
                beg = mid + 1
                mid = int((beg+end)/2)
        if array[mid] == item:
            return mid
        else:
            return None
binary_search([1,2,3,4,5,6],2)
'''
print('The time for own implementation is: ',timeit(x))
print('The time for built-in implementation is: ', timeit('''arr=[1,2,3,4,5,6]
arr.index(2)'''))
```

LAB_3.py ● LAB_5.py

LAB_3.py > ...

```python
# part (c) -->
def check_sort(arr):
    for i in range(len(arr)-1):
        if arr[i] > arr[i+1]:
            return False
    return True

def take_array():
    array_elements = input(f"Enter the numbers of the array separated by commas: ").split(',')
    array_elements = list(map(lambda x: int(x), array_elements))
    if check_sort(array_elements):
        return array_elements
    else:
        return f'The inputed array is not sorted:('

def insertion_sort(arr):
    for i in range(len(arr)-1):
        if arr[i] < arr[-1]:
            continue
        else:
            temp = arr[i]
            arr[i] = arr[-1]
            arr[-1] = temp
    return arr
```

LAB_3.py ● LAB_5.py

LAB_3.py > ...

```python
def binary_search():
    arr = take_array()
    if check_sort(arr):
        item = int(input(f'enter the number which you want to search in the above array: '))
        if isinstance(arr, list):
            beg, end = 0, len(arr)-1
            mid = (beg+end)//2
            while beg <= end and arr[mid] != item:
                if item < arr[mid]:
                    end = mid - 1
                    mid (beg+end)//2
                else:
                    beg = mid + 1
                    mid = (beg+end)//2
            if item == arr[mid]:
                return mid
            else:
                print(f'Your entered item is not in the array, adding that item in the array: ')
                arr.append(item)
                insertion_sort(arr)
                return arr
        else:
            message = f'The Inputed Array is not Sorted'
            return message
    else:
        return f'The inputed array is not sorted!!'

print(binary_search())
```

```python
 1   # Algorithm A5.1
 2   A = [[1,0,0],[2,3,0],[4,5,6]]
 3   # here A is a lower triangular matrix of order 3 x 3.
 4   def storeTriangular(array):
 5       U = []
 6       for i in range(len(A)):
 7           for j in range(len(A)):
 8               if j <= i:
 9                   U.append(A[i][j])
10       return U
11   # print(storeTriangular(A))
12
13   from math import *
14   # Algorithm A5.2
15   U = [1,1,2,1,2,3]
16   def retrieveTriangular(U):
17       a = 1
18       b = 1
19       c = -2*len(U)
20       d = b**2-4*a*c
21       x1 = (-b+sqrt(d))/(2*a)
22       x2 = (-b-sqrt(d))/(2*a)
23       if x1.real > x2.real:
24           c = int(x1.real)
25       else:
26           c = int(x2.real)
27       A = []
28       count = 0
29       for i in range(c):
30           lst = []
31           for j in range(c):
32               if j <= i:
33                   lst.append(U[count])
34                   count += 1
35               else:
36                   lst.append(0)
37           A.append(lst)
38       return A
39   # print(retrieveTriangular(U))
40
41   # part c
42   # Algo to store the non-zero values of a Tridiagonal Matrix B in a linear array U.
43   def storeTriDiagonal(array):
44       U = []
45       for i in range(len(array)):
46           for j in range(len(array)):
47               # checking for the first row
48               if i == 0 and j in [0,1]:
49                   U.append(array[i][j])
50               # checking for the last row
51               elif i == len(array)-1 and j in [len(array)-1, len(array)-2]:
52                   U.append(array[i][j])
53               # for all the middle rows
54               else:
55                   if j >= i-1 and j <= i+1:
56                       U.append(array[i][j])
57       return U
```

```python
68
69   # Algo to retrieve the non-zero values from a linear array U and place them ina triadiagonal matrix B
70   def retrieveTriDiagonal(U):
71       B = []
72       count = 0
73       # for n
74       n = int((len(U)+2)/3)
75       for i in range(n):
76           lst = []
77           for j in range(n):
78               if i == 0 and j in [0,1]:
79                   lst.append(U[count])
80                   count += 1
81               elif i == n-1 and j in [n-1, n-2]:
82                   lst.append(U[count])
83                   count += 1
84               elif j >= i-1 and j <= i+1:
85                   lst.append(U[count])
86                   count += 1
87               else:
88                   lst.append(0)
89           B.append(lst)
90       return B
91   U = [5, -7, 1, 4, 3, 9, -3, 6, 2, 4]
92   print(retrieveTriDiagonal(U))
```

Screenshot 1 — LAB_5.py

```python
 97    from scipy.sparse import *
 98    import numpy as np
 99
100    mat = [
101        [1, 23, 0, 0, 0],
102        [1, 2, 8, 0 , 0],
103        [0, 0, 20, 0, 0]
104    ]
105
106    dense_arr = np.array(mat)
107    crr = csr_matrix(dense_arr)
108    dense_arr2 = csr_matrix.todense(crr)
109
110    print(crr)
111    print(dense_arr2)
112    print(dense_arr)
```

Screenshot 2 — linked_list.py

```python
27        # method to insert a node at the given index of the Linked List
28        def insert_at(self, index, data):
29            if index < 0 or index >= self.get_length():
30                raise Exception("Linked List Index Out of Range")
31
32            # for inserting at the head
33            if index == 0:
34                self.insert_at_beginning(data)
35                return
36            elif index == self.get_length() - 1:
37                self.insert_at_end(data)
38                return
39            else:
40                count = 0
41                itr = self.head
42                while itr:
43                    if count == index - 1:
44                        node = Node(data, itr.next)
45                        itr.next = node
46                        break
47                    itr = itr.next
48                    count += 1
```

Screenshot 3 — linked_list.py

```python
 1    # first creating the class Node which holds its data and pointer to the next node
 2    class Node:
 3        def __init__(self, data = None, next = None):
 4            self.data = data
 5            self.next = next
 6    # now creating a class called LinkedList which will keep track of all the Nodes
 7    class LinkedList:
 8        def __init__(self):
 9            self.head = None
10        # method to insert a node at the beginning of the Linked List
11        def insert_at_beginning(self, data):…
14
15        # method to insert a node at the end of the Linked List
16        def insert_at_end(self, data):…
26
27        # method to insert a node at the given index of the Linked List
28        def insert_at(self, index, data):…
46
47        # method to search for a node at the given index of the Linked List
48        def search_at(self, index):…
62
63        # method to delete a node from the LinkedList
64        def remove_at(self, index):…
80
81        # method to create a new LinkedList from a list of values
82        def insert_values(self, data_list):…
86
87        # method to calculate the length of the Linked List
88        def get_length(self):…
95
96        #method to traverse the linkedList.
97        def printll(self):…
```

```python
        # method to delete a node from the LinkedList
        def remove_at(self, index):
            if index < 0 or index >= self.get_length():
                raise Exception("Linked List Index Out of Range")

            # for deleting the very first node
            if index == 0:
                self.head = self.head.next

            count = 0
            itr = self.head
            while itr:
                if count == index - 1:
                    itr.next = itr.next.next
                    break
                itr = itr.next
                count += 1
```

```python
        # method to search for a node at the given index of the Linked List
        def search_at(self, index):
            if index < 0 or index >= self.get_length():
                raise Exception("Linked List Index Out of Range")

            if index == 0:
                return self.head.data
            else:
                count = 0
                itr = self.head
                while itr:
                    if count == index:
                        return itr.data
                    itr = itr.next
                    count += 1
```

EXPLORER

LINKED_LISTS
- doubly_linke... U
- linked_list.py M

doubly_linked_list.py > DoublyLinkedList > print_forward

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

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

    def print_forward(self): ...

    def print_backward(self): ...

    def get_last_node(self): ...

    def get_length(self): ...

    def insert_at_begining(self, data): ...

    def insert_at_end(self, data): ...

    def insert_at(self, index, data): ...

    def remove_at(self, index): ...

    def insert_values(self, data_list): ...


if __name__ == '__main__':
    ll = DoublyLinkedList()
    ll.insert_values(["banana","mango","grapes","orange"])
    ll.print_forward()
```

> OUTLINE
> TIMELINE

```python
    def insert_at(self, index, data):
        if index<0 or index>self.get_length():
            raise Exception("Invalid Index")

        if index==0:
            self.insert_at_begining(data)
            return

        count = 0
        itr = self.head
        while itr:
            if count == index - 1:
                node = Node(data, itr.next, itr)
                if node.next:
                    node.next.prev = node
                itr.next = node
                break

            itr = itr.next
            count += 1

    def remove_at(self, index): …

    def insert_values(self, data_list): …
```

```python
    def remove_at(self, index):
        if index<0 or index>=self.get_length():
            raise Exception("Invalid Index")

        if index==0:
            self.head = self.head.next
            self.head.prev = None
            return

        count = 0
        itr = self.head
        while itr:
            if count == index:
                itr.prev.next = itr.next
                if itr.next:
                    itr.next.prev = itr.prev
                break

            itr = itr.next
            count+=1
```