SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur-603203

DEPARTMENT

OF

MASTER OF COMPUTER APPLICATION

ACADEMIC YEAR: 2024-2025 (ODD SEMESTER)

LAB MANUAL

(Phase I)

(REGULATION - 2025)

MC4167 – PYTHON PROGRAMMING LABORATORY

FIRST SEMSTER

MCA – COMPUTER APPLICATIONS

Prepared By

Dr.R.THENMOZHI, Associate Professor

Department of Computer Applications

VISION OF THE DEPARTMENT

To educate students with conceptual knowledge and technical skills in the field of Computer Applications with moral and ethical values to achieve excellence in academic, industry, and research-centric environments.

MISSION OF THE DEPARTMENT

1. To inculcate in students a firm foundation in theory and practice of computer application skillscoupled with the thought process for disruptive innovation and research methodologies, to keep pace with emerging technologies.
2. To provide a conducive environment for all academic, administrative, and interdisciplinary research activities using state-of-the-art technologies.
3. To stimulate the growth of graduates and doctorates, who will enter the workforce as productive software professionals, researchers, and entrepreneurs with necessary soft skills, and continue higher professional education with competence in the global market.
4. To enable seamless collaboration with the IT industry and Government for consultancy and sponsored research.
5. To cater, to the cross-cultural, multinational, and demographic diversity of students.
6. To educate the students on the social, ethical, and moral values needed to make significantcontributions to society.

INDEX

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

- To prepare students with a breadth of knowledge to comprehend, analyze, design, and createcomputing solutions to real-life problems and to excel in industry / technical profession.
- To provide students with a solid foundation in mathematical and computing fundamentals and techniques required to solve technology-related problems and to pursue higher studies and research.
- To inculcate a professional and ethical attitude in students, to enable them to work towards a broad social context.
- To empower students with skills required to work as members and leaders in multidisciplinaryteams and with continuous learning ability on technology and trends needed for a successfulcareer.

• .

PROGRAMME OUTCOMES (POs)

After going through the four years of study, Information Technology Graduates will exhibit ability to:

PO #	Graduate Attribute	Program Outcome
1	Engineering knowledge	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.
2	Problem analysis	Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3	Design/development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4	Conduct investigations of complex problems	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions
5	Modern tool usage	Create, select, and apply appropriate techniques, resources, and

		modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.
6	The engineer and society	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7	Environment and sustainability	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8	Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
9	Individual and team work	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings
10	Communication	Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions
11	Project management and finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments
12	Life-long learning	Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

MC4167 **PYTHON PROGRAMMING LABORATORY** **L T P C**
0 0 4 2

LIST OF EXPERIMENTS:

1. Python programming using simple statements and expressions (exchange thevalues of two variables, circulate the values of n variables, distance between twopoints).
2. Scientific problems using Conditionals and Iterative loops.
3. Linear search and Binary search
4. Selection sort, Insertion sort
5. Merge sort, Quick Sort
6. Implementing applications using Lists, Tuples.
7. Implementing applications using Sets, Dictionaries.
8. Implementing programs using Functions.
9. Implementing programs using Strings.
10. Implementing programs using written modules and Python Standard Libraries (pandas,numpy, Matplotlib, scipy)
11. Implementing real-time/technical applications using File handling.
12. Implementing real-time/technical applications using Exception handling.
13. Creating and Instantiating classes

Total: 60 Periods

COURSE OUTCOMES:

On completion of the laboratory course, the student should be able to

CO1: Apply the Python language syntax including control statements, loops and functions tosolve a wide variety of problems in mathematics and science.

CO2: Use the core data structures like lists, dictionaries, tuples and sets in Python to store,process and sort the data

CO3: Create files and perform read and write operations

CO4: Illustrate the application of python libraries.

CO5: Handle exceptions and create classes and objects for any real time applications

LIST OF EQUIPMENTS FOR A BATCH OF 30 STUDENTS

HARDWARE/SOFTWARE REQUIREMENTS

1: Processors: Intel Atom® processor Intel®Core™i3 processor2:
Disk space: 1GB.
3: Operating systems: Windows 7,macOS and Linux
4: Python versions:2.7, 3.6, 3.8

COURSE OUTCOMES

MC4167.1	Apply the Python language syntax including control statements, loops and functions tosolve a wide variety of problems in mathematics and science.
MC4167.2	Use the core data structures like lists, dictionaries, tuples and sets in Python to store,process and sort the data
MC4167.3	Create files and perform read and write operations
MC4167.4	Illustrate the application of python libraries.
MC4167.5	Handle exceptions and create classes and objects for any real time applications

CO- PO MATRIX

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
MC4167.1	2	1	3	3	2	2	-	-	-	-	-	-
MC4167.2	2	1	3	3	2	2	-	-	-	-	-	-
MC4167.3	1	1	3	2	2	2	-	-	-	-	-	-
MC4167.4	2	1	3	2	2	2	-	-	-	-	-	-
MC4167.5	2	1	3	3	2	3	-	-	-	-	-	-
Average	1.8	1	3	2.6	2	2.2	-	-	-	-	-	-

EVALUATION PROCEDURE FOR EACH EXPERIMENT

S.No	Description	Mark
1.	Aim & Pre-Lab discussion	20
2.	Observation	20
3.	Conduction and Execution	30
4.	Output & Result	10
5.	Viva	20
Total		**100**

INTERNAL ASSESSMENT FOR LABORATORY

S.No	Description	Mark
1.	Conduction & Execution of Experiment	30
2.	Record	10
3.	Model Test	20
Total		**60**

Ex.No:1

PYTHON PROGRAMMING USING SIMPLE STATEMENTS AND EXPRESSIONS

Ex.No:1a

AIM:

To exchange the given values of two variables using python.

PRE LAB DISCUSSION:

ALGORITHM:

Step 1: Start the program.

Step 2: Get two integer inputs var1 and var2 from the user using the input() function.

Step 3: Declare third variable, c.

Step 4: c=a, a=b, b=c.

Step 5: Print the output swapped values a and b.

Step 6: Stop the program.

PROGRAM:

```
print("Swapping using temporary variable")

a = int(input("a = "))

b = int(input("b = "))

print("Before Swapping")

print("a = ", a)

print("b = ", b)

c = a

a = b

b = c

print("After Swapping")

print("a = ", a)

print("b = ", b)
```

OUTPUT:

Swapping using temporary variable

a = 10

b = 20

Before Swapping

a = 10

b = 20

After Swapping

a = 20

b = 10

RESULT

Thus the program to exchange the given values of two variables using python has been executed successfully.

Ex.No.1b.

CIRCULATE THE VALUES OF N VARIABLES

Aim:

To circulate the given values of n variables using python.

Algorithm:

Step 1: Start the program.

Step 2: Get one integer input no_of_terms from the user using the input() function.

Step 3: Read the value of no_of_terms.

Step 4: Create a list as list1.

Step 5: Validate the range of no_of_terms.

Step 6: Then, get one integer input ele from the user using input() function.

Step 7: Add a single item to the existing list using the append method.

Step 8: Print the circulative values.

Step 9: Stop the program.

PROGRAM:

```
#Circulate the values of n variables
no_of_terms=int(input("Enter number of values : "))
list1=[]
for val in range(0,no_of_terms,1):
    ele=int(input("Enter integer : "))
    list1.append(ele)
#Circulate and display values
print("Circulating the elements of list ",list)
for val in range(0,no_of_terms,1):
    ele=list1.pop(0)
    list1.append(ele)
    print(list1)
```

OUTPUT:

Enter number of values : 5

Enter integer : 8

Enter integer : 5

Enter integer : 3

Enter integer : 6

Enter integer : 7

Circulating the elements of list <class 'list'>

[5, 3, 6, 7, 8]

[3, 6, 7, 8, 5]

[6, 7, 8, 5, 3]

[7, 8, 5, 3, 6]

[8, 5, 3, 6, 7]

Result:

Thus the program has been executed successfully circulate the given values of n variables using python.

Ex.No:1c

CALCULATE DISTANCE BETWEEN TWO POINTS

AIM:

To calculate distance between two points using python

ALGORITHM:

Step 1: Start the program.

Step 2: Get two integer inputs x1 and x2 for coordinates 1 from the user using the input() function.

Step 3: Then, Get two integer inputs y1 and y2 for coordinates 2.

Step 4: Calculate using the two points (x1,y1) and (x2,y2),the distance between these points is given by the formula.

Step 5: (((x2-x1)**2+((y2-1)**2))**0.5. **Step**

Step 6: Print the distance between values. **Step**

Step 7: Stop the program.

PROGRAM:

```
#Distance between two points
print("Enter coordinates for Point 1 : ")
x1=int(input("enter x1 : "))
x2=int(input("enter x2 : "))
print("Enter coordinates for Point 2 : ")
y1=int(input("enter y1 : "))
y2=int(input("enter y2 : "))
result= ((((x2 - x1 )**2) + ((y2-y1)**2) )**0.5)
print("distance between",(x1,x2),"and",(y1,y2),"is : ",result)
```

OUTPUT

```
Enter coordinates for Point 1 :
enter x1 : 100
enter x2 : 200
Enter coordinates for Point 2 :
enter y1 : 500
enter y2 : 500
distance between (100, 200) and (500, 500) is :  100.0
```


RESULT

Thus the program to calculate distance between two points using python has been executed successfully

Ex.No:2 **SCIENTIFIC PROBLEMS USING CONDITIONALS AND ITERATIVE LOOPS**

AIM :

To write a Python Program for scientific problems using conditionals and iterative loops

PRE LAB DISCUSSION

Conditional statements

`if`, `elif`, and `else`.

Basic `if` Statement

```
x = 10
if x > 5:
    print("x is greater than 5")
```

`if-else` Statement

```
x = 4
if x > 5:
print("x is greater than 5")
else:
print("x is 5 or less")
```

`if-elif-else` Statement

```
x = 10
if x > 10:
print("x is greater than 10")
elif x == 10:
 print("x is exactly 10")
        else:
print("x is less than 10")
```

2. Iterative Loops

`for` loops and `while` loops

`for` Loop

`case:1`

```
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
```

```
print(fruit)
```

case:2

```
        for i in range(5):
print(i)
```

while Loop

```
       count = 0
while count < 5:
print(count)
 count += 1
 # Increment the count to eventually end the loop
```

Using if Inside a for Loop

```
numbers = [1, 2, 3, 4, 5]
for number in numbers:
if number % 2 == 0:
   print(f"{number} is even")
else:
   print(f"{number} is odd")
```

Using break and continue

```
# Using break
for i in range(10):
if i == 5:
  break
     print(i)
# Using continue
for i in range(10):
      if i % 2 == 0:
        continue
      print(i)
# Prints only odd numbers
```

Ex.No:2a **FIBONACCI SERIES**

AIM:

Write a python program to generate Fibonacci series using function.

ALGORITHM:

Step1:Start

Step2:Get the number of terms

Step3: Check if the number of terms is valid

Step4:If there is only one term, return n1

Step5:If it not generate the fibonacci sequence upto n terms

Step5:End

PROGRAM:

```
nterms = int(input("How many terms? "))
n1, n2 = 0, 1
count = 0
if nterms <= 0:
   print("Please enter a positive integer")
elif nterms == 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
else:
   print("Fibonacci sequence:")
   while count < nterms:
       print(n1)
       nth = n1 + n2
       # update values
       n1 = n2
       n2 = nth
       count += 1
```

OUTPUT:

How many terms? 5

Fibonacci sequence:

0

1

1

2

3

RESULT:

Thus the program is executed to find the Fibonacci series of a given number and the output is obtained.

Ex.No:2b **AMSTRONG NUMBER**

AIM :

To write a Python program to find the Armstrong number.

PRELAB DISCUSSION:

Given a number x, determine whether the given number is Armstrong number or not. A positive integer of n digits is called an Armstrong number of order n (order is number of digits) if.

abcd... = pow(a,n) + pow(b,n) + pow(c,n) + pow(d,n) + ...

ALGORITHM:

Step1:Start

Step2:Define Function to calculate x raised to the power y

Step3: Calculate order of the number

Step4:Then add the number with the sum

Step5:Define the function isArmstrong() to check the given is armstrong number or not.

Step6:If it the digit is a armstrong number

Step6:If it is not the digit is not a Armstrong number.

PROGRAM/SOURCE CODE:

```
def power(x, y):
    if y == 0:
        return 1
    if y % 2 == 0:
        return power(x, y // 2) * power(x, y // 2)
    return x * power(x, y // 2) * power(x, y // 2)
def order(x):
    n = 0
    while (x != 0):
        n = n + 1
        x = x // 10
    return n
def isArmstrong(x):
    n = order(x)
    temp = x
    sum1 = 0
```

```
    while (temp != 0):
        r = temp % 10
        sum1 = sum1 + power(r, n)
        temp = temp // 10
    return (sum1 == x)
x = 153
print(isArmstrong(x))
x = 1253
print(isArmstrong(x))
```

OUTPUT:

```
True
False
```


RESULT:

Thus the program is executed to find the given number is Armstrong number or not and the output is obtained.

Ex.No:2c **PALINDROME**

AIM:

Write a Python program to reverse the digits of a given number and add them to the original. Repeat this procedure if the sum is not a palindrome.

Note: A palindrome is a word, number, or other sequence of characters which reads the same backward as forward, such as madam or race car.

ALGORITHM:

Step1:Start

Step2:Define the function

Step3:Check the position of the digits

Step4:If the end of the string Matches with the first string then given digit is a palindrome

Step5:If it not matches the digit is not a palindrome

Step5:End

PROGRAM:

```
def rev_number(n):
  s = 0
  while True:
    k = str(n)
    if k == k[::-1]:
      break
    else:
      m = int(k[::-1])
      n += m
      s += 1
  return n
rev=int(input("enter num:"))
print(rev_number(rev))
```

OUTPUT:

Enter num: 145

Pallindrome number is

686

Enter num: 144

Pallindrome number is

585

RESULT:

Thus the program is executed to find the palindrome of a given number and the output is obtained

Ex.No:3a **LINEAR SEARCH**

AIM :

To write a python Program to perform linear search

ALGORITHM:

Step 1: Start.

Step 2: Read the number of element in the list.

Step 3: Read the number until loop n -1.

Step 4: Then Append the all element in list

Step 5: Go to STEP -3 upto n -1.

Step 6 : Read the searching element from the user

Step 7 : Assign to FALSE flag value

Step 8 : Search the element with using for loop until length of list

Step 9 : If value is found assign the flag value is true

Step10 : Then print the output of founded value an d position.

Step 11 : If value is not found then go to next step

Step 12 : Print the not found statement

PROGRAM :

```
a=[ ]
n=int(input("Enter number of
elements:")) for i in range(1,n+1):
 b=int(input("Enter element:"))
a.append(b)
```

```
x = int(input("Enter number to search: "))
 found = False
for i in range(len(a)):
if(a[i] = = x):
found = True
print("%d found at%dthposition"%(x,i))
break
if (found==False):
print("%d is not in list"%x)
```

OUTPUT 1:

```
Enter number of elements:5
Enter element:88
Enter element:11
Enter element:64
Enter element:23
Enter element:89
Enter number to search: 11
11 found at 1th position
```

OUTPUT 2:

```
 Enter number of elements:5
Enter element:47
Enter element:99
Enter element:21
Enter element:35
Enter element:61
Enter number to search: 50
50 is not in list
```

RESULT:

Thus the program to perform linear Search is executed and the output is obtained.

Ex.No. 3b. **BINARY SEARCH**

AIM:

To write a python program to perform the binary search.

ALGORITHM:

Step:1 - mid = (starting index + last index) / 2

Step:2 - If starting index > last index

Then, Print "Element not found"

Exit

Else if element >arr[mid]

Then, starting index = mid + 1

Go to Step:1

Else if element <arr[mid]

Then, last index = mid - 1

Go to Step:2

Else:

{ means element == arr[mid] }

Print "Element Presented at position" + mid

Exit

PROGRAM :

```
def Binary_search(arr, start_index, last_index, element):
    while(start_index<=last_index):
                mid=int(start_index+last_index)/2)
                if(element>arr[mid]):
```

```
                start_index=mid+1
            elif(element<arr[mid]):
                last_index=mid-1
        elif(element==arr[mid]):
            return mid
        else
        return -1
arr=[]
n=input("enter no of elements:"))
for i in range(1,n+1):
    b=int(input("enter element"))
    arr.append(b)
print(arr)
        element=int(input("enter element to be searched"))
 start_index=0
 last_index=len(arr)-1
 found = Binary_search(arr, start_index, last_index, element)
 if (found = = -1):
print ("element not present in array")
else
        print("element is present at index", found)
```

OUTPUT 1:

Enter number of elements:8

Enter element:11

Enter element:33

Enter element:44

Enter element:56

Enter element:63

Enter element:77

Enter element:88

Enter element:90

[11, 33, 44, 56, 63, 77, 88, 90]

Enter the element to be searched

63 element is present at index 4

OUTPUT 2:

Enter number of elements:7

Enter element:11

Enter element:15

Enter element:20

Enter element:25

Enter element:30

Enter element:40

Enter element:50

[11, 15, 20, 25, 30, 40, 50] Enter the element to be searched 22

element not present in array

RESULT:

Thus the program to perform Binary Search is executed and the output is obtained.

Ex.No.4a **SELECTION SORT**

AIM:

To study and Implement Selection sort using python.

PRE LAB DISCUSSION

The provided Python code demonstrates the Selection Sort algorithm. Selection Sort has a time complexity of O(n^2). In each iteration, the code finds the minimum element's index in the unsorted portion of the array and swaps it with the current index's element. This gradually sorts the array from left to right. The example initializes an array, applies the selectionSort function to sort it, and then prints the sorted array in ascending order. The sorted array is obtained by repeatedly finding the smallest element in the unsorted portion and placing it in its correct position, resulting in an ordered array

ALGORITHM:

1. Start
2. Get the length of the array.
3. length = len(array) → 6
4. First, we set the first element as minimum element.
5. Now compare the minimum with the second element. If the second element is smaller than the first, we assign it as a minimum.
6. After each iteration, minimum element is swapped in front of the unsorted array.
7. The second to third steps are repeated until we get the sorted array.
8. Stop

PROGRAM:

```
# Selection sort in Python
# time complexity O(n*n)
#sorting by finding min_index
def selectionSort(array, size):

  for ind in range(size):
    min_index = ind
```

```
    for j in range(ind + 1, size):
        # select the minimum element in every iteration
        if array[j] < array[min_index]:
            min_index = j
     # swapping the elements to sort the array
    (array[ind], array[min_index]) = (array[min_index], array[ind])

arr = [-2, 45, 0, 11, -9,88,-97,-202,747]
size = len(arr)
selectionSort(arr, size)
print('The array after sorting in Ascending Order by selection sort is:')
print(arr)
```

OUTPUT

```
The array after sorting in Ascending Order by selection sort is:

[-202, -97, -9, -2, 0, 11, 45, 88, 747]
```

RESULT:

Thus the python program is implemented by selection sort to sort the given array.

Ex.No.4b **INSERTION SORT**

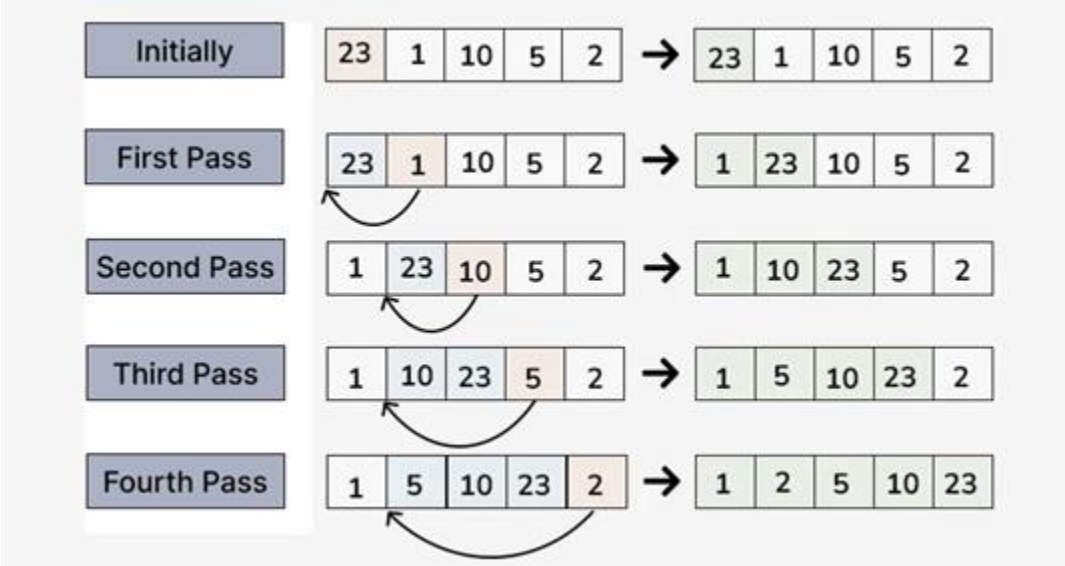
AIM:

To perform sorting using Insertion sort

PRE LAB DISCUSSION

The insertionSort function takes an array arr as input. It first calculates the length of the array (n). If the length is 0 or 1, the function returns immediately as an array with 0 or 1 element is considered already sorted.

For arrays with more than one element, the function proceeds to iterate over the array starting from the second element. It takes the current element (referred to as the "key") and compares it with the elements in the sorted portion of the array that precede it. If the key is smaller than an element in the sorted portion, the function shifts that element to the right, creating space for the key. This process continues until the correct position for the key is found, and it is then inserted in that position.



.

ALGORITHM:

1. Start
2. We start with second element of the array as first element in the array is assumed to be sorted.
3. Compare second element with the first element and check if the second element is smaller then swap them.
4. Move to the third element and compare it with the second element, then the first element and swap as necessary to put it in the correct position among the first three elements.
5. Continue this process, comparing each element with the ones before it and swapping as needed to place it in the correct position among the sorted elements.
6. Repeat until the entire array is sorted.
7. Stop

PROGRAM:

```
def insertionSort(arr):
    n = len(arr)  # Get the length of the array
        if n <= 1:
        return  # If the array has 0 or 1 element, it is already sorted, so return
     for i in range(1, n):  # Iterate over the array starting from the second element
        key = arr[i]  # Store the current element as the key to be inserted in the right position
        j = i-1
        while j >= 0 and key < arr[j]:  # Move elements greater than key one position ahead
            arr[j+1] = arr[j]  # Shift elements to the right
            j -= 1
        arr[j+1] = key  # Insert the key in the correct position
  # Sorting the array [12, 11, 13, 5, 6] using insertionSort
arr = [12, 11, 13, 5, 6]
insertionSort(arr)
print('The array after sorting in Ascending Order by insertion sort is:')
print(arr)
```

OUTPUT

```
The array after sorting in Ascending Order by insertion sort is:
[5, 6, 11, 12, 13]
```

RESULT :

Thus the python program to perform sorting using insertion sort technique is executed successfully.

EX. NO: 5a **MERGE SORT**

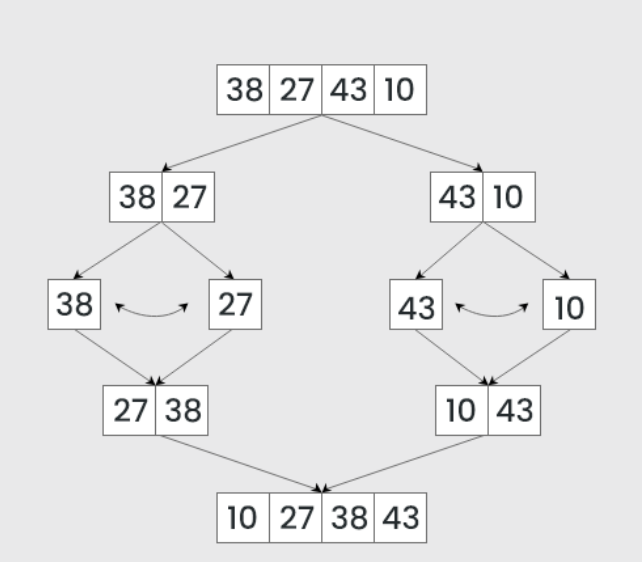
AIM

To perform sorting of the given data items using merge sort

PRE LAB DISCUSSION

Merge sort is a sorting algorithm that follows the **divide-and-conquer** approach. It works by recursively dividing the input array into smaller subarrays and sorting those subarrays then merging them back together to obtain the sorted array.

In simple terms, we can say that the process of **merge sort** is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.



ALGORITHM:

1. Start
2. Check if the left index is less than the right index.
3. Calculate the midpoint of the array if the low index is less than the high index.
4. Call the mergesort function on the left and right halves of the array.
5. Merge the two sorted halves using the merge function.
6. Merge function creates two different arrays and copies the left and right halves into these arrays.
7. Iteration and comparison of both arrays are done.
8. After merging, the arrays are sorted in ascending order.
9. Stop

PROGRAM:

```
# 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=" ")
```

OUTPUT:

Given array is 12 11 13 5 6 7

Sorted array is 5 6 7 11 12 13

RESULT

Thus the python program to perform sorting using merge sort technique is executed successfully.

EX NO : 5b **QUICK SORT**

AIM:

To sort the given list of data items using quick sort techniques.

PRE LAB DISCUSSION

One of the most effective sorting algorithms is Quicksort, which is based on the divide-and-conquer strategy. Quicksort makes some average memories complexity of O(n log n) and is generally utilized practically speaking.

ALGORITHM:

Inputs:

A: an array of n elements

lo: the index of the first element of the sub-array to be sorted

hi: the index of the last element of the sub-array to be sorted

1. If lo is less than hi, then do the following:
   - Call partition(A, lo, hi) and store the index of the pivot element in p.
   - Recursively call quicksort(A, lo, p-1).
   - Recursively call quicksort(A, p+1, hi).

Partition Algorithm:

1. Let pivot be the last element of the sub-array A[lo..hi].
2. Let i be the index of the first element of the sub-array.
3. For each j from lo to hi-1, do the following:
1. If A[j] <= pivot, then do the following:
1. Increment i.
2. Swap A[i] with A[j].
4. Swap A[i+1] with A[hi].
5. Return i+1.

PROGRAM:

```
# Python program for Quicksort
def quicksort(arr, lo, hi):
    """
    Sorts the given array in ascending order using the Quicksort algorithm.
```

```
    Parameters:
    arr (list): The array to be sorted
    lo (int): The index of the first element in the sub-array to be sorted
    hi (int): The index of the last element in the sub-array to be sorted
    """
    if lo < hi:
        # Partition the array and get the index of the pivot element
        p = partition(arr, lo, hi)

        # Recursively sort the left and right partitions
        quicksort(arr, lo, p-1)
        quicksort(arr, p+1, hi)
def partition(arr, lo, hi):
    """
    Partitions the sub-array by selecting the last element as the pivot,
    and rearranging the array so that all elements to the left of the pivot
    are less than or equal to the pivot, and all elements to the right of the
    pivot are greater than the pivot.

    Parameters:
    arr (list): The array to be partitioned
    lo (int): The index of the first element in the sub-array to be partitioned
    hi (int): The index of the last element in the sub-array to be partitioned

    Returns:
    int: The index of the pivot element after partitioning
    """
    # Select the last element as the pivot
    pivot = arr[hi]
    i = lo - 1

    # Loop through the sub-array and partition it
```

```
    for j in range(lo, hi):
        if arr[j] <= pivot:
            # Move the element to the left partition
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    # Move the pivot element to its final position in the array
    arr[i+1], arr[hi] = arr[hi], arr[i+1]
    # Return the index of the pivot element
    return i+1
# Example usage
arr = [10, 7, 8, 9, 1, 5]
n = len(arr)
quicksort(arr, 0, n-1)
print("The given array before sorting is : [10, 7, 8, 9, 1, 5] ")
print("Sorted array by quick sort is:", arr)
```

OUTPUT

```
The given array before sorting is : [10, 7, 8, 9, 1, 5]
Sorted array by quick sort is: [1, 5, 7, 8, 9, 10]
```

RESULT :

Thus the python program for sorting the given data items using quick sort is executed successfully.

EX NO: 6a IMPLEMENTING APPLICATIONS USING LISTS

Aim:

To write a python program to create, slice, change, delete and index elements using List.

PRE LAB DISCUSSION

In Python, list slicing is a common practice and it is the most used technique for programmers to solve efficient problems. Consider a Python list, in order to access a range of elements in a list, you need to slice a list. One way to do this is to use the simple slicing operator i.e. colon(:). With this operator, one can specify where to start the slicing, where to end, and specify the step. List slicing returns a new list from the existing list.

Python List Slicing Syntax

The format for list slicing is of Python List Slicing is as follows:

Lst[Initial : End : IndexJump]

If *Lst* is a list, then the above expression returns the portion of the list from index *Initial* to index *End*, at a step size *IndexJump*.

ALGORITHM :

Step 1: Create the List.

Step 2: Indexing the List using the index operator [ ].

Step3: Silicing an element from the List

Step4: Step 4: Changing an element from the List.

Step 5: Appending the List.

Step 6: Removing an element from the List.

Step 7:Deleting an element from the List.

PROGRAM:

```
print("list is created in the name:list")

list=['p','e','r','m','i','t']

print('list created",list)

print("list indexing",list[0])

print("list negative indexing",list[-1])

print("list slicing",list[1:4])

list=['p','e','r','m','i','t']
```

```
print("Given list",list)

 list[0]=2

print("List Changing",list)

list[1:4]=[1,2,3]

print("List Changing",list)

list =
['p','e','r','m','i','t']

print("Given list",list)

list[0]=2

print("List Changing",list)

list[1:4]=[1,2,3]

print("List Changing",list)

list =
['p','e','r','m','i','t']

print("Given list",list)


list.append(['add','sub'])

print("List appending",list)

list  = ['p','e','r','m','i','t'  ]

print("Given list",list)

list.remove('p')

print("List Removing",list)

list = ['p','e','r','m','i','t']

print("Given list",list)

list[2:5] = []

print("List Delete",list)
```

OUTPUT :

List is created in the name: list

List Created ['p', 'e', 'r', 'm', 'i', 't']

List indexing p

List negative indexing t

List slicing ['e', 'r', 'm']

Given list ['p', 'e', 'r', 'm', 'i', 't']

List Changing [2, 'e', 'r', 'm', 'i', 't']

List Changing [2, 1, 2, 3, 'i', 't']

Given list ['p', 'e', 'r', 'm', 'i', 't']

List appending ['p', 'e', 'r', 'm', 'i', 't', ['add', 'sub']]

Given list ['p', 'e', 'r', 'm', 'i', 't']

List Removing ['e', 'r', 'm', 'i', 't']

Given list[‘p’,’e’,’r’,’m’,’i’,’t’]

List Delete[‘p’,’e’,’t’]

RESULT:

Thus program to create, slice, change, delete and index elements using list is executed and the output is obtained**.**

Ex. No: 6b **IMPLEMENTING APPLICATIONS USING TUPLE**

AIM:

To write a python program to create, slice, delete and index elements using Tuple

ALGORITHM:

1: start

2: create a tuple with empty, having integers,objects in different data types and nested tuples

3: slicing the tuple with : operator

4. Deleting the tuple with del method

5. Indexing the elements of tuple

6.Stop.

PROGRAM1:

 # Python program to show how to create a tuple

```
# Creating an empty tuple
empty_tuple = ()
print("Empty tuple: ", empty_tuple)

# Creating tuple having integers
int_tuple = (4, 6, 8, 10, 12, 14)
print("Tuple with integers: ", int_tuple)

# Creating a tuple having objects of different data types
mixed_tuple = (4, "Python", 9.3)
print("Tuple with different data types: ", mixed_tuple)

# Creating a nested tuple
nested_tuple = ("Python", {4: 5, 6: 2, 8:2}, (5, 3, 5, 6))
print("A nested tuple: ", nested_tuple)
```

OUTPUT:

```
Empty tuple: ()
Tuple with integers: (4, 6, 8, 10, 12, 14)
Tuple with different data types: (4, 'Python', 9.3)
A nested tuple: ('Python', {4: 5, 6: 2, 8: 2}, (5, 3, 5, 6))
```

PROGRAM2:

```
# Python program to show how slicing works in Python tuples
# Creating a tuple
tuple_ = ("Python", "Tuple", "Ordered", "Immutable", "Collection", "Objects")
# Using slicing to access elements of the tuple
print("Elements between indices 1 and 3: ", tuple_[1:3])
# Using negative indexing in slicing
print("Elements between indices 0 and -4: ", tuple_[:-4])
# Printing the entire tuple by using the default start and end values.
print("Entire tuple: ", tuple_[:])
```

OUTPUT:

```
Elements between indices 1 and 3: ('Tuple', 'Ordered')
Elements between indices 0 and -4: ('Python', 'Tuple')
Entire tuple: ('Python', 'Tuple', 'Ordered', 'Immutable', 'Collection', 'Objects')
```

PROGRAM3:

```
# Python program to show how to delete elements of a Python tuple
# Creating a tuple
tuple_ = ("Python", "Tuple", "Ordered", "Immutable", "Collection", "Objects")
# Deleting a particular element of the tuple
try:
    del tuple_[3]
    print(tuple_)
except Exception as e:
    print(e)
# Deleting the variable from the global space of the program
del tuple_
# Trying accessing the tuple after deleting it
try:
    print(tuple_)
except Exception as e:
    print(e)
```

OUTPUT:

'tuple' object does not support item deletion

name 'tuple_' is not defined

PROGRAM4:

```
# Creating tuples
Tuple_data = (0, 1, 2, 3, 2, 3, 1, 3, 2)
# getting the index of 3
res = Tuple_data.index(3)
print('First occurrence of 1 is', res)
# getting the index of 3 after 4th
# index
res = Tuple_data.index(3, 4)
print('First occurrence of 1 after 4th index is:', res)
```

OUTPUT:

First occurrence of 1 is 2

First occurrence of 1 after 4th index is: 6

RESULT:

Thus the program to illustrate the applications of tuple like create, slice, delete and index elements using Tuple has been executed successfully.