Practical No. 1

JSPM

Page No.:

Title:

Write a python program to store N number of student whose play different three different games.

Aim: In second year computer engineering class, group A Write student's play cirket, group B students play badminton and group C students play football. Write a Python program using functions to compute following:

a) List of students who play both cricket and badminton.

b) List of students who play either ceicket or badminton but not both.

c) Number of students who play neither cricket nor badminton.

d) Number of students who play cricket & football but not badminton.

(Note- While realizing the group, duplicate entires should be avoided, Do not use SET built-in fun?)

Pre-Requisite: Python Programming

Objectives: To understand the use functions for N students whose play different three different games.

Theory

List are sequence containers, that allow non-contiguous memory allocation. As compared to the vector, the list has slow traversal, but once a position has

been found, insertion & deletion are quick (const time). Normally, when we say a list, we talk about a doubly linked list.

-Syntax:

```
std:: list < data_type> name_of_list;
```

Practical No. 2

JSPM

Page No.:

Title:

Write a python program to store marks for N students.

Aim:

Write a python program to store marks scored in subject "Fundamental of Data Structure" by N students in class write functions to compute following.

a) The average score of class.
b) Highest score and lowest score of class.
c) Count of students who were absent for the test.
d) display mark with highest frequency.

Pre-Requisite: Python programming

Objectives: To understand the use functions for N students record input N number of students.

Outcome: Resulting average, highest and lowest marks operation.

Theory:

Arrays - Arrays a kind of data structure that can store a fixed size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to thank of an array as a collecti

of variables of the same type.

Instead of declaring individual variables such as number 0, number 1, --- and number 99, you declare one array variable such as numbers and use numbers[0], numbers[1] and ---, numbers[99] to represent individual variables. A specific element is as array accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Declaring Arrays:

To declare an array in C, a programmer specifies the type of elements and the no. of elements required by an array as follows.

Snytax:

```
type arrayName [ArraySize];
```

This is called a single-dimensional array. The arraysize must be an integer constant greater than zero and type can be any valid C data type. For example to declare a 10 element array called balance of type double, use this element.

```
double balance [10];
```

Here balance is a variable array which sufficient to hold up to double numbers.

Initializing Array's:

You can initialize an array in C either one by one or using a single statement as follows

```
double balance[10] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

Accessing Array Elements:

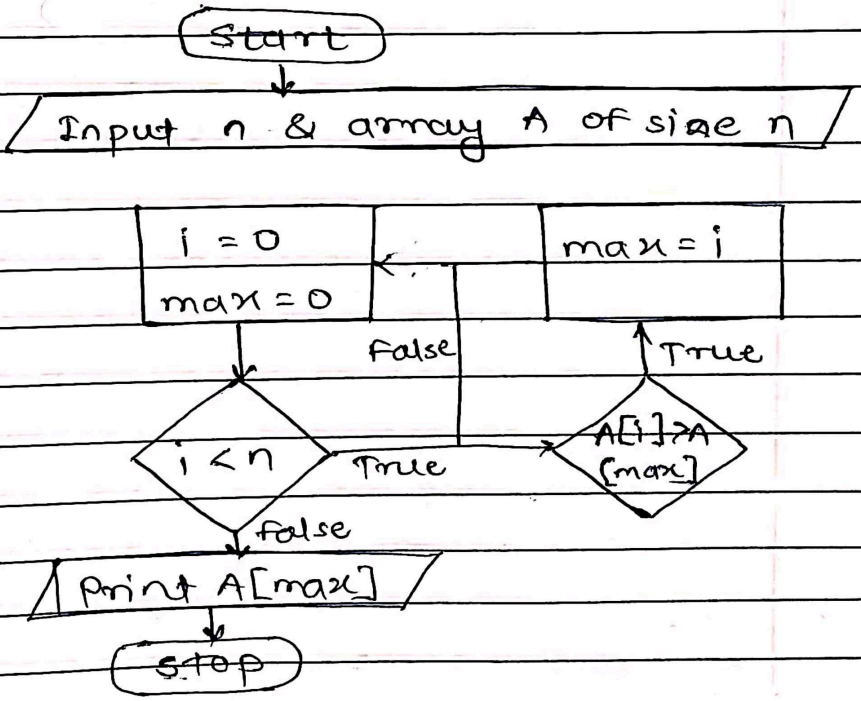
An element is accessed by indexing the array name. this is done by placing the index of the element within square brackets after the name of array.

For example:

```
double salary = balance[9];
```

function used
Are as given infront the concept.

Flowchart:



Python program to store marks of n stu

Conclusion:

By this way, we can store marks of N students successfully.

Practical No. 3

JSPM

Page No.:

Title:

Perform the different operations and matrix.

Aim: Write a python program to complete following computation on matrix

a) Addition of two matrix.
b) Substraction of two matrix.
c) Multiplication of two matrix.
d) Transpose of matrix

Pre-Requisite: Knowledge of representing matrix in python. Knowledge of different operations that can be performed on matrix.

Objectives: Compute the transpose of matrix - perform addition, substraction & multiplication of two matrix.

Input: No. of rows & columns of two matrixes - Elements of both matrices.

Output: Transpose of a matrix.
- result of addition, substraction, multiplication or both matrixes.

Theory: 2 Dimensional Array:

A two dimensional array is a data struct that contains a collections of cells laid out in a two dimensional grid. similar to a table with rows & columns. Although the

values are the still stored linearly in

Algorithm:

Step 1: Start

Step 2: Input no. of rows & columns of fi
matrix.

Step 3: Input elements of first matrix

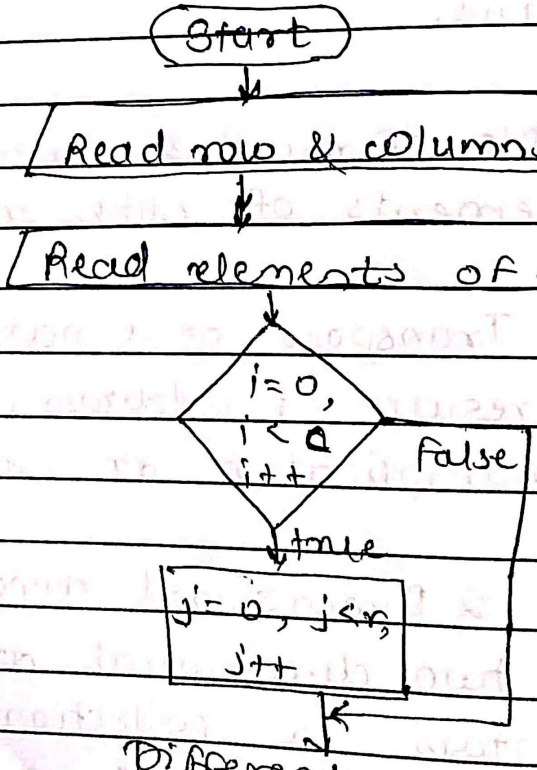
Step 4: Input no. of rows & columns of seco
matrix.

Step 5: Input the elements of second matri

Step 6: Fun^n to transpose first matrix ie
element at row^r column^c in original
palace at row^c & column^r of
transpose.

Step 7: Fun^n to add, substract, & multiply
two matrix.

Flowchart:



Conclusion: We have element the concep
of different operations on matrix.

Practical No. 5

JSPM

Page No.:

Title: Sorting of an array using selection & bubble sort.

Aim: Write a python program to store first year percentage of students in array. Write fun^n for sorting array of floating point no. in ascending order using a). selection sort b) Bubble sort & display top 5 scores of club.

Pre-Requisite: Knowledge of sorting techniques.

Objectives: To sort an array of floating point number in ascending order using
a) Selection sort
b) Bubble sort & display top 5 scores.

Input: size of array, elements of array.

Theory:
- One fundamental problems of computer science is ordering a list of items.
- Many solutions exist to this problem, known as sorting algorithms.
- Some sorting algorithms are simple such, as the bubble sort, others Quick sort are extremely complecated but produces lightning fast results.
- There are five different techniques of sorting

1) Bubble sort
2) Insertion Sort.
3) Quick sort
4) Merge sort
5) Shell sort.

1] Bubble Sort:

- Compare each element (except the last one) with its neighbour to the right.
- If they are out of order, then swap them.
- This puts the largest element at the very end.
- The last element is now in the correct & final place.
- Compare each element with its neighbour to right
- If they are out of order swap them.
- This puts the seconds largest element to last.
- The last two elements are now in their correct & final place.
- Compare each element with its neighbour to rig
- Continue above process until you have no unsorted element on the left.

2] Selection Sort:

- Selection sort divides list into two lists.
- Initially sorted sublist is empty.
- Find the smallest element from unsorted list & exchange it with element from in the 1st position
- Find the 2nd smallest element & exchange it with the element in the 2nd position.
- Continue until the array is sorted.
- Disadvantage:

  Running time depends only slightly on the amount of order in the file.

* Example on Bubble sort:
5, 10, 56, 90, 3

→ Given Data: 5, 10, 56, 90, 3
Pass 1: 5, 10, 56, 90, 3
5, 10, 56, 3, 90

Pass 2: 5, 10, 56, 3, 90
5, 10, 3, 56, 90

Pass 3: 5, 10, 3, 56, 90
5, 3, 10, 56, 90

Pass 4: 5, 3, 10, 56, 90
3, 5, 10, 56, 90 is the sorted list.

* example on selection sort:
34, 9, 78, 65, 12, -5

→ given list [34, 9, 78, 65, 12, -5]
Pass 1 result list = -5, 9, 78, 65, 12, 34

Pass 2 result list = -5, 9, 78, 65, 12, 34

Pass 3 result list = -5, 9, 12, 65, 78, 34

Pass 4 result list = -5, 9, 12, 34, 78, 65

Pass 5 result list = -5, 9, 12, 34, 65, 78

Sorted
Pass 6 result list = -5, 9, 12, 34, 65, 78

* Algorithm:

For bubble Sort:

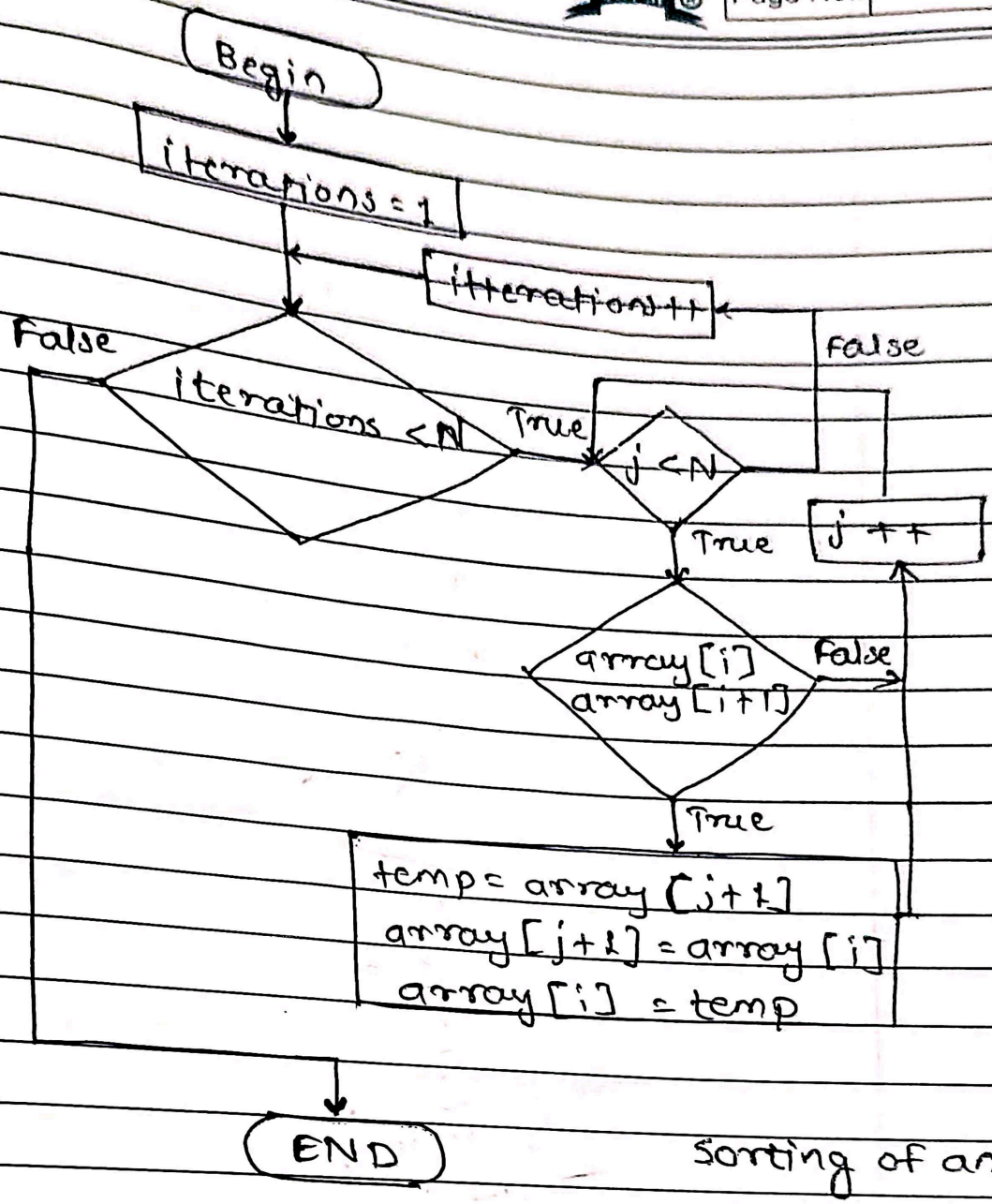
```
def bubbleSort (alist):
    for passnum in range (len(alist)-1, 0, -1):
        for i in range (passnum):
            if alist[i] > alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp
alist = [54, 26, 93, 17, 77, 31, 44, 55, 20]
bubbleSort (alist)
print (alist)
```

For selectionSort:

```
def selectionSort (alist):
    for fillslot in range (len(alist)-1, 0, -1):
        positionOfMax = 0
        for location in range (1, fillslot+1):
            if alist [location] > alist [positionOfMa
                positionOfMax = location
                temp = alist [fillslot]
                alist [fillslot] = alist [positionOfM
                alist [positionOfMax] = temp

alist = [54, 26, 93, 17, 77, 31, 44, 55, 20]
selectionSort (alist)
print (alist)
```

* Flowchart:



Sorting of an array using insertion & selection.

Conclusion:

We have studied to implement this concept sorting an array using selection & bubble sort.

Practical No. 6

JSPM
Page No.:

Title: Sorting array of floating point no's in ascending order using quick sort.

Aim: Write a python program to store first year percentage of student in array. Write fun^n for sorting array of floating point no.'s in ascending order.

Pre-requisite: Knowledge of sorting techniques.

Objectives: To sort array using quick sort.

Input: size of array,
first year percentage of students.

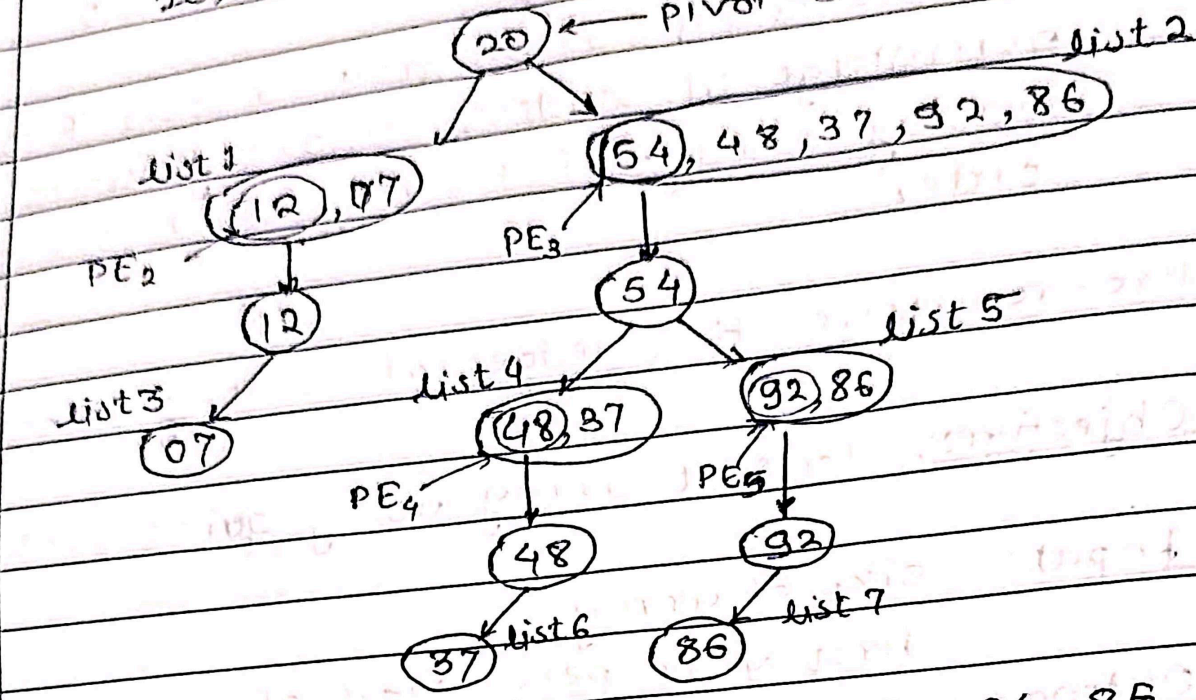
Outpome: To sort array using quick sort
To display top five scores.

Theory:

Quick Sort:

1) Quick sort works as follows.
2) Select an element in the array as a 'pivot'
3) Place the pivot element at its proper place such that the array gets partitioned into two subparts.
3) All elements in the first sub part are smaller than the pivot.
4) All elements in the second sub part are greater than the pivot element.
5) Procedure is repeated respeel recursively for both subparts.

* example on quick sort:
20, 54, 48, 37, 12, 92, 86, 07



Sorted list: 07, 12, 20, 37, 48, 54, 86, 92

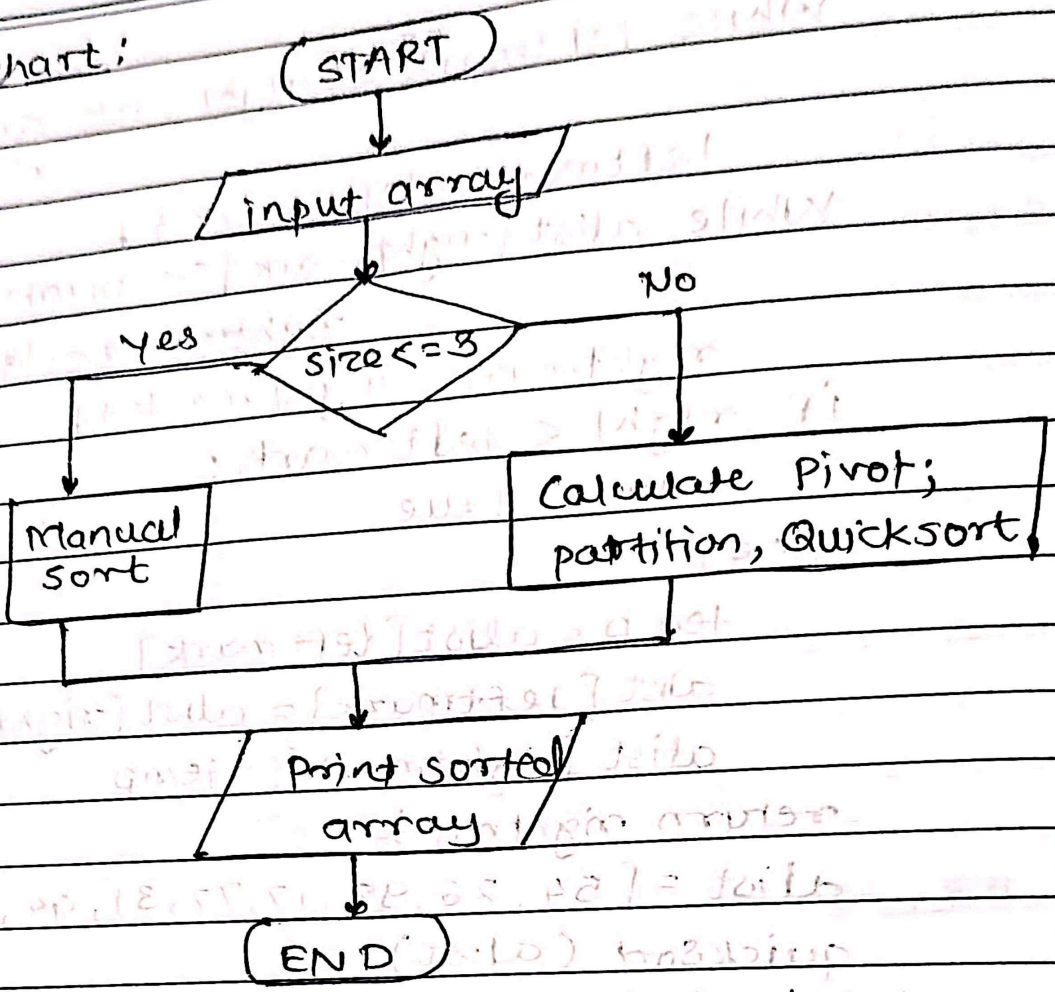
* Algorithm:

```
def quickSort(alist):
    quicksortHelper(alist, 0, len(alist)-1)
def quicksortHelper(alist, first, last):
    if first < last:
        splitpoint = partition(alist, first, las
quicksortHelper(alist, first, splitpoint-1)
quicksortHelper(alist, splitpoint+1, last)
def partition(alist, first, last):
    privatvalue = alist[first] leftmark = first
                        right mark = last.

    done = false
    While not
```

```
done:
    while leftmark <= rightmark and alist[leftmark]
                                    <=pivotvalue:
        leftmark = leftmark + 1
    while alist[rightmark] >= pivotvalue and
                        rightmark >= leftmark :
        rightmark = rightmark + 1
    if right < left mark :
        done = True
    else:
        temp = alist[leftmark]
        alist[leftmark] = alist[rightmark]
        alist[rightmark] = temp
    return rightmark
alist = [54, 26, 93, 17, 77, 31, 44, 55, 20]
quicksort (alist)
print(alist)
```

* flowchart:



QuickSort C++ program to maintain club me
using singly linked list.

* Conclusion:

By this way, we can perform standing arr
of floating no. is ascending order using quick S