NAMAL UNIVERSITY MIANWALI
DEPARTMENT OF ELECTRICAL ENGINEERING

DATA STRUCTURE AND ALGORITHM

LAB # 13
REPORT

Title : Implementation of Sorting Algorithms in Python

Name	Fahim-Ur-Rehman Shah
Roll No	NIM-BSEE-2021-24
Intructor	Ms. Naureen Shaukat
Date	06-July-2023
Marks	

1. Lab Objectives

The objective of this lab is to implement sorting algorithms in the Python.

2. Lab Outcomes

- CLO 1: Recognize the usage of fundamental Data structure using Python Programming Language.
- CLO 2: Analyzing computation cost of different algorithms.
- CLO 3: Demonstrate solution to real life problems using appropriate data structures.

3. Equipment

• Software o IDLE (Python 3.11)

4. Instructions

1. This is an individual lab. You will perform the tasks individually and submit a report.
2. Some of these tasks (marked as 'Example') are for practice purposes only while others (marked as 'Task') have to be answered in the report.
3. When asked to display an output in the task, either save it as jpeg or take a screenshot, in order to insert it in the report.
4. The report should be submitted on the given template, including:
    a. Code (copy and pasted, NOT a screenshot)
    b. Output figure (as instructed in 3)
    c. Explanation where required
5. The report should be properly formatted, with easy to read code and easy to see figures.
6. Plagiarism or any hint thereof will be dealt with strictly.
7. Late submission of report is allowed within 03 days after lab with 20% deduction of marks every day.
8. You have to submit report in pdf format (Reg.X_DSA_LabReportX.pdf).

5. Background

Sorting Algorithm

A Sorting Algorithm is used to rearrange a given array or list of elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of elements in the respective data structure.
There are five basic sorting Algorithms we will see in this lab.

1. Merge Sort

Merge Sort is a recursive algorithm that works on the basis of divide and conquer. It divides the array into subarray, and then merge them in the specific order. The python code for the merge sort is as follows,

```
def merge_sort(S):
    n = len(S)
    if n < 2:
        return

    # Divide
    mid = n // 2
    S1 = S[0:mid]
    S2 = S[mid:n]

    # Conquer (with recursion)
    merge_sort(S1)
    merge_sort(S2)

    # Merge results
    merge(S1, S2, S)

def merge(S1, S2, S):
    i = j = 0
    while i + j < len(S):
        if j == len(S2) or (i < len(S1) and S1[i] < S2[j]):
            S[i+j] = S1[i]
            i += 1
        else:
            S[i+j] = S2[j]
            j += 1

my_list = [5, 2, 9, 1, 7]
merge_sort(my_list)
print(my_list)
```

Home Task 1 : ***Write the values of variables and the numerical values involved in each line for every iteration of the function, given the array***

ANS :

Iteration 1:

- S = [5, 2, 9, 1, 7]
- n = 5
- mid = 2
- S1 = [5, 2]
- S2 = [9, 1, 7]

Recursive Iteration 1 (S1):

- S = [5, 2]
- n = 2
- mid = 1
- S1 = [5]
- S2 = [2]

Recursive Iteration 2 (S1):

- S = [5]
- n = 1

Recursive Iteration 3 (S2):

- S = [2]

- n = 1

Merge S1 and S2 into S:

- S1 = [5]
- S2 = [2]
- S = [5, 2]

Recursive Iteration 4 (S2):

- S = [9, 1, 7]
- n = 3
- mid = 1
- S1 = [9]
- S2 = [1, 7]

Recursive Iteration 5 (S1):

- S = [9]
- n = 1

Recursive Iteration 6 (S2):

- S = [1, 7]
- n = 2
- mid = 1
- S1 = [1]
- S2 = [7]

Recursive Iteration 7 (S1):

- S = [1]
- n = 1

Recursive Iteration 8 (S2):

- S = [7]
- n = 1

Merge S1 and S2 into S:

- S1 = [1]
- S2 = [7]
- S = [1, 7]

Merge S1 and S2 into S:

- S1 = [5, 2]
- S2 = [1, 7]
- S = [1, 2, 5, 7]

Final result: [1, 2, 5, 7, 9]

2. Quick Sort

QuickSort is a sorting algorithm based on the Divide and Conquer algorithm that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array. The python code for the Quick Sort is given as,

```
def partition(array, low, high):
    pivot = array[high]
    i = low - 1
    for j in range(low, high):
        if array[j] <= pivot:
            i = i + 1
            (array[i], array[j]) = (array[j], array[i])
    (array[i + 1], array[high]) = (array[high], array[i + 1])
    return i + 1

def quickSort(array, low, high):
    if low < high:
        print(array)
        pi = partition(array, low, high)
        quickSort(array, low, pi - 1)
        quickSort(array, pi + 1, high)

data = [8, 7, 2, 1, 0, 9, 6]
print("Unsorted Array")
print(data)
size = len(data)
quickSort(data, 0, size - 1)
print('Sorted Array in Ascending Order:')
print(data)
```

Home Task 2 : ***Write the values of variables and the numerical values involved in each line for every iteration of the function, given the array***

ANS :

1. Initial call: quickSort(data, 0, 6)
   - array = [8, 7, 2, 1, 0, 9, 6]
   - pivot = 6
   - partition swaps elements to get array = [2, 1, 0, 6, 8, 7, 9]
   - Recursive calls: quickSort(array, 0, 2) and quickSort(array, 4, 6)
2. Recursive call: quickSort(array, 0, 2)
   - array = [2, 1, 0, 6, 8, 7, 9]
   - pivot = 0
   - partition does not swap elements (already sorted)
   - Recursive call: quickSort(array, 0, 0)

3. Recursive call: quickSort(array, 0, 0)
    - Base case, no further calls

The final sorted array is [0, 1, 2, 6, 7, 8, 9].

3. *Insertion Sort:*

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Take help of lecture slides and notes to implement insertion sort in Python.

Lab Task 1: Implement the python code for insertion sort of an array.

Code :

```python
def insertion_sort(arr):
    for i in range (1,len(arr)):
        left_move = arr[i]
        # print(f"left move : {left_move}")
        j =  i-1
        while j>=0 and left_move < arr[j]:
            arr[j+1] = arr[j]

            j = j-1

        arr[j+1] =left_move
        # print(arr)

arr  = [5,9,3,1,2,8,4,7,6]
insertion_sort(arr)
print(arr)
```

Output :

```
\Data Structure and Algorithm\Lab\Lab 13\insertion_sorting_final.py"


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

PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 13>
```

4. Selection Sort

Selection sort is an effective and efficient sort algorithm based on comparison operations. It adds one element in each iteration. You need to select the smallest element in the array and move it to the beginning of the array by swapping with the front element.

Lab Task 2: Implement the python code for selection sort of an array.

Code :

```
def selection_sort(arr):
    size = len(arr)
    for i in range(size-1):
        min_index = i
        for j in range(min_index+1,size):
            if arr[j] < arr[min_index]:
                min_index = j

        if i != min_index:
            arr[i],arr[min_index] = arr[min_index] , arr[i]


arr  = [5,9,3,1,2,8,4,7,6]
selection_sort(arr)
print(arr)
```

Output :

```
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 13> python -u "e:\Semester 4\Data
Structure and Algorithm\Lab\Lab 13\selection_sort.py"

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

PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 13>
```

5. Bubble Sort :

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order.

Home Task 3: Implement the bubble sorting algorithm in Python.

Code :

```
def bubble_sort(arr):
    size =len(arr)

    for i in range(size-1):
        swapped = False
        for j in range(size-1-i):
            if arr[j] > arr[j+1]:
                tmp = arr[j]
                arr[j] = arr[j+1]
                arr[j+1] = tmp
                swapped =True
        if not swapped:
            break


arrr = [5,9,3,1,2,8,4,7,6]
print(f"before sorting : {arrr}")
bubble_sort(arrr)
print(f"before sorting : {arrr}")
```

Output :

```
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 13> python -u "e:\Semester 4\Data Structure and Algorithm\Lab\Lab 13\bubble_sort.py"


    before sorting : [5, 9, 3, 1, 2, 8, 4, 7, 6]

    After  sorting : [1, 2, 3, 4, 5, 6, 7, 8, 9]


PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 13>
```

Algorithm Analysis:

Conduct algorithm analysis in form of O(n) for each sorting scheme.

Sorting Algorithm	*Best Case*	*Average Case*	*Worst Case*
Bubble Sort	**O(n)**	**O(n^2)**	**O(n^2)**
Selection Sort	**O(n^2)**	**O(n^2)**	**O(n^2)**
Insertion Sort	**O(n)**	**O(n^2)**	**O(n^2)**

Lab Evaluation Rubrics							
Domain	**CLOs/ Rubric**	**Performance Indicator**	**Unsatisfactory 0-2**	**Marginal 3-5**	**Satisfactory 6-8**	**Exemplary 9-10**	**Allocate d Marks**
Psychomotor	**CLO:1 R2**	Implementation with Results **(P)**	Does not try to solve problems. Many mistakes in code and difficult to comprehend for the instructor. There is not result of the problem.	Does not suggests or refine solutions but is willing to try out solutions suggested by others. Few mistakes in code, but done along with comments, and easy to comprehend for the instructor. Few mistake in result.	Refines solutions suggested by others. Complete and error-free code is done. No comments in the code, but easy to comprehend for the instructor. Results are correctly produced.	Actively looks for and suggests solution to problems. Complete and error free code is done, easy to comprehend for the instructor. Results are correctly produced. Student incorporated comments in the code.	
Affective	**CLO:3 R3**	Lab Report **(A)**	Code of the problem is not given. Outputs are not provided. Explanation of the solution is not stated.	Code of the problem is given. Output is not complete. Explanation of the solution is not satisfactory.	Code of the problem is given. Output is completely given. Explanation of the solution is not satisfactory.	Code of the problem is given. Output is completely given. Explanation of the solution is satisfactory.	
	CLO:1 R5	Discipline and Behavior **(A)**	Got and wandered around. More than two incidents of talking non-lab related stuff in lab and/or any talk with other groups, voice level exceeding the appropriate level, use of cell phones and involvement in any non-lab activity.	Got out of seat and wander around for some time. No more than two incidents of talking non-lab related stuff in lab. Voice level exceeding the appropriate level, use of cell phones and involvement in any non-lab related activity.	Stayed in seat and got up for a specific lab related reason, but took more time than required to do the job. No more than one incidents of talking non-lab related stuff in lab. Voice level exceeding the appropriate level, use of cell phones and involvement in any non-lab related activity.	Stayed in seat and got up for a specific lab related reason. Took care of lab related business and sat down right away. Voice level kept appropriate. Not used cell phones or involved in any non- lab related activity.	