

Advanced Data Structures with Python Laboratory (MCAC294)

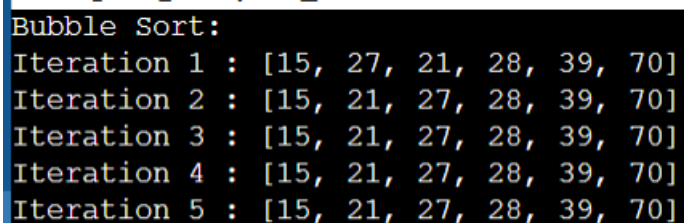
Assignment – 1

Data Set:- 27,15,39,21,28,70

1. Write a Python program to implement the concept of Bubble sort on the above data set. Print the data set after every iteration.

```
data = [27, 15, 39, 21, 28, 70]
n = len(data)
print("Bubble Sort:")
for i in range(n - 1):
    for j in range(n - 1 - i):
        if data[j] > data[j + 1]:
            data[j], data[j + 1] = data[j + 1], data[j]
    print("Iteration", i + 1, ":", data)
```

Output:-



```
Bubble Sort:
Iteration 1 : [15, 27, 21, 28, 39, 70]
Iteration 2 : [15, 21, 27, 28, 39, 70]
Iteration 3 : [15, 21, 27, 28, 39, 70]
Iteration 4 : [15, 21, 27, 28, 39, 70]
Iteration 5 : [15, 21, 27, 28, 39, 70]
```

2. Write a Python program to implement the concept of Selection sort on the above data set. Print the data set after every iteration.

```
data = [27, 15, 39, 21, 28, 70]
n = len(data)
print("Selection Sort:")
for i in range(n):
```

```

min_index = i
for j in range(i + 1, n):
    if data[j] < data[min_index]:
        min_index = j
data[i], data[min_index] = data[min_index], data[i]
print("Iteration", i + 1, ":", data)

```

Output:-

```

Selection Sort:
Iteration 1 : [15, 27, 39, 21, 28, 70]
Iteration 2 : [15, 21, 39, 27, 28, 70]
Iteration 3 : [15, 21, 27, 39, 28, 70]
Iteration 4 : [15, 21, 27, 28, 39, 70]
Iteration 5 : [15, 21, 27, 28, 39, 70]
Iteration 6 : [15, 21, 27, 28, 39, 70]

```

3. Write a Python program to implement the concept of Insertion sort on the above data set. Print the data set after every iteration.

```

data = [27, 15, 39, 21, 28, 70]
n = len(data)
print("Insertion Sort:")
for i in range(1, n):
    key = data[i]
    j = i - 1
    while j >= 0 and data[j] > key:
        data[j + 1] = data[j]
        j -= 1
    data[j + 1] = key
    print("Iteration", i, ":", data)

```

Output:-

```

Insertion Sort:
Iteration 1 : [15, 27, 39, 21, 28, 70]
Iteration 2 : [15, 27, 39, 21, 28, 70]
Iteration 3 : [15, 21, 27, 39, 28, 70]
Iteration 4 : [15, 21, 27, 28, 39, 70]
Iteration 5 : [15, 21, 27, 28, 39, 70]

```

4. Write a Python program to implement the concept of Quick sort on the above data set. Print the data set after every iteration.

```
data = [27, 15, 39, 21, 28, 70]
def quick_sort(arr, low, high):
    if low < high:
        pivot = arr[high]
        i = low - 1
        for j in range(low, high):
            if arr[j] <= pivot:
                i += 1
                arr[i], arr[j] = arr[j], arr[i]
        arr[i + 1], arr[high] = arr[high], arr[i + 1]
        pi = i + 1
        print("Partition:", arr)
        quick_sort(arr, low, pi - 1)
        quick_sort(arr, pi + 1, high)
print("Quick Sort:")
quick_sort(data, 0, len(data) - 1)
```

Output:-

```
Quick Sort:
Partition: [27, 15, 39, 21, 28, 70]
Partition: [27, 15, 21, 28, 39, 70]
Partition: [15, 21, 27, 28, 39, 70]
```

5. Write a Python program to implement the concept of Merge sort on the above data set. Print the data set after every iteration.

```
data = [27, 15, 39, 21, 28, 70]
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left = arr[:mid]
        right = arr[mid:]
        merge_sort(left)
        merge_sort(right)
        i = j = k = 0
```

```

while i < len(left) and j < len(right):
    if left[i] < right[j]:
        arr[k] = left[i]
        i += 1
    else:
        arr[k] = right[j]
        j += 1
    k += 1
while i < len(left):
    arr[k] = left[i]
    i += 1
    k += 1
while j < len(right):
    arr[k] = right[j]
    j += 1
    k += 1
print("Merged:", arr)
print("Merge Sort:")
merge_sort(data)

```

Output:-

```

Merged: [15, 39]
Merged: [15, 27, 39]
Merged: [28, 70]
Merged: [21, 28, 70]
Merged: [15, 21, 27, 28, 39, 70]

```

6. Write a Python program to show that Quick sort is better than Bubble sort.

```

import time
import sys
sys.setrecursionlimit(5000)
data = [27, 15, 39, 21, 28, 70] * 1000
def partition(arr, low, high):
    mid = (low + high) // 2
    pivot = sorted([arr[low], arr[mid], arr[high]])[1]
    i = low - 1

```

```

for j in range(low, high):
    if arr[j] <= pivot:
        i += 1
        arr[i], arr[j] = arr[j], arr[i]
arr[i + 1], arr[high] = arr[high], arr[i + 1]
return i + 1
def quick_sort(arr, low, high):
    while low < high:
        pi = partition(arr, low, high)
        if pi - low < high - pi:
            quick_sort(arr, low, pi - 1)
            low = pi + 1
        else:
            quick_sort(arr, pi + 1, high)
            high = pi - 1
start = time.time()
quick_sort(data, 0, len(data) - 1)
end = time.time()
print("Quick Sort Time:", end - start)

```

Output:-

```
Quick Sort Time: 0.8877737522125244
```

7. Write a Python program to show that merge sort is more effective than quick sort.

```

import time
import sys
sys.setrecursionlimit(5000)
data1 = [27, 15, 39, 21, 28, 70] * 1000
data2 = data1.copy()
data3 = data1.copy()

# ---- QUICK SORT ----
def partition(arr, low, high):
    mid = (low + high) // 2
    pivot = sorted([arr[low], arr[mid], arr[high]])[1]

```

```

i = low - 1
for j in range(low, high):
    if arr[j] <= pivot:
        i += 1
        arr[i], arr[j] = arr[j], arr[i]
arr[i + 1], arr[high] = arr[high], arr[i + 1]
return i + 1
def quick_sort(arr, low, high):
    while low < high:
        pi = partition(arr, low, high)
        if pi - low < high - pi:
            quick_sort(arr, low, pi - 1)
            low = pi + 1
        else:
            quick_sort(arr, pi + 1, high)
            high = pi - 1

```

---- MERGE SORT ----

```

def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    merged_arr = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            merged_arr.append(left[i])
            i += 1
        else:
            merged_arr.append(right[j])
            j += 1
    merged_arr.extend(left[i:])
    merged_arr.extend(right[j:])
    return merged_arr

```

---- PERFORMANCE COMPARISON ----

Quick Sort Timing

start = time.time()

```
quick_sort(data2, 0, len(data2) - 1)
end = time.time()
quick_time = end - start
print("Quick Sort Time:", quick_time)

# Merge Sort Timing
start = time.time()
data3 = merge_sort(data3) # Store sorted result
end = time.time()
merge_time = end - start
print("Merge Sort Time:", merge_time)
print("Merge Sort is", round(quick_time / merge_time, 2), "times
faster than Quick Sort")
```

Output:-

```
Quick Sort Time: 0.7221212387084961
Merge Sort Time: 0.022747516632080078
Merge Sort is 31.75 times faster than Quick Sort
```